# Asynchronous Mini-Batch Gradient Descent with Variance Reduction for Non-Convex Optimization

**Zhouyuan Huo**
Dept. of Computer Science and Engineering
University of Texas at Arlington
Arlington, TX, 76019, USA
zhouyuan.huo@mavs.uta.edu

**Heng Huang**
Dept. of Computer Science and Engineering
University of Texas at Arlington
Arlington, TX, 76019, USA
heng@uta.edu

## Abstract

We provide the first theoretical analysis on the convergence rate of asynchronous mini-batch gradient descent with variance reduction (AsySVRG) for non-convex optimization. Asynchronous stochastic gradient descent (AsySGD) has been broadly used for deep learning optimization, and it is proved to converge with rate of $O(1/\sqrt{T})$ for non-convex optimization. Recently, variance reduction technique is proposed and it is proved to be able to accelerate the convergence of SGD greatly. It is shown that asynchronous SGD method with variance reduction technique has linear convergence rate when problem is strongly convex. However, there is still no work to analyze the convergence rate of this method for non-convex problem. In this paper, we consider two asynchronous parallel implementations of mini-batch gradient descent method with variance reduction: one is on distributed-memory architecture and the other is on shared-memory architecture. We prove that both methods can converge with a rate of $O(1/T)$ for non-convex optimization, and linear speedup is accessible when we increase the number of workers. We evaluate our methods by optimizing multi-layer neural networks on two real datasets (MNIST and CIFAR-10), and experimental results demonstrate our theoretical analysis.

## Introduction

With the boom of data, training machine learning model with large-scale datasets becomes a challenging problem. Basing on batch gradient descent (GD) method, researchers propose stochastic gradient descent (SGD) method or mini-batch gradient descent method to relieve the complexity of computation in each iteration and reduce the total time complexity for optimization (Nemirovski et al. 2009; Lan 2012; Ghadimi and Lan 2013; Ghadimi, Lan, and Zhang 2016; Bottou 2010). Due to efficiency, SGD method has been widely used to solve different kinds of large-scale machine learning problems, including both convex and non-convex. However, because we use stochastic gradient to approximate full gradient in the process, a decreasing learning rate has to be applied to guarantee convergence, or it is very easy to diverge from the optimal solution. Thus, it leads to a sub-linear convergence rate of $O(1/T)$ on strongly convex problem. Recently, stochastic variance reduced gradient (SVRG) (Johnson and Zhang 2013) and its variants, such as SAGA(Defazio, Bach, and Lacoste-Julien 2014), m2SGD(Konečný et al. 2014), have gained much attention in stochastic optimization. Through reusing the previously computed first order gradient information, these methods are able to reduce the variance of gradient approximation in the optimization and are proved to have linear convergence rate on strongly convex problem. After that, SVRG is then applied to solve non-convex problem (Allen-Zhu 2016; Reddi et al. 2016), and it is proved to have a faster sub-linear convergence rate of $O(1/T)$. Experiments are conducted on neural networks and their results also validate that it outperforms SGD method for non-convex optimization.

Serial algorithm is not able to make good use of computation resource. Therefore, parallel algorithms are introduced to further speedup the computation task, including synchronous optimization and asynchronous optimization. Because there is no need of synchronization between workers, asynchronous methods often have better performance. Asynchronous parallelism has been successfully applied to speedup many state-of-the-art optimization algorithms, such as SGD (Recht et al. 2011; Lian et al. 2015), stochastic coordinate descent (SCD) (Liu, Wright, and Sridhar 2014), SVRG (Zhang, Zheng, and Kwok 2015) and DualFreeSDCA (Huo and Huang 2016). There are mainly two kinds of distributed architectures, one is distributed-memory architecture on multiple machines (Agarwal and Duchi 2011; Lian et al. 2015; Zhang and Kwok 2014; Dean et al. 2012; Zhang, Zheng, and Kwok 2015) and the other one is shared-memory architecture on a multi-core machine (Recht et al. 2011; Zhao and Li 2016; Langford, Smola, and Zinkevich 2009). Deep learning is a typical situation where asynchronous SGD and its variants have gained great success(LeCun, Bengio, and Hinton 2015; Dean et al. 2012; Lian et al. 2015; Ngiam et al. 2011). It is known that deep neural network always has large set of parameters and trains with large-scale datasets.

Recently, asynchronous SVRG method has been implemented and studied on both distributed-memory architecture (Zhang, Zheng, and Kwok 2015) and shared-memory architecture (Zhao and Li 2016). It is proved that asynchronous SVRG method has linear convergence rate on strongly convex problem. Mini-batch gradient is implemented in the experiments, while it is missing in their proof. Further, there is no theoretical analysis of convergence rate for asynchronous SVRG on non-convex problem yet.

In this paper, we provide the convergence analysis of asynchronous mini-batch gradient descent with variance reduction method (asySVRG) for non-convex optimization. Two different algorithms and analysis are proposed on two different distributed architectures, one is shared-memory architecture and the other is distributed-memory architecture. The key difference between these two categories lies on that distributed-memory architecture can ensure the atomicity of reading and writing the whole vector of $x$, while the shared-memory architecture can usually just ensure atomic reading and writing on a single coordinate of $x$ (Lian et al. 2015). We implement asySVRG on two different architectures and analyze their convergence rate based on the mini-batch setting. We prove that asySVRG can get convergence rate of $O(1/T)$ on both architectures. Besides, we also prove that linear speedup is accessible when we increase the number of workers until reaching an upper bound.

We list our main contributions as follows:

- We extend asynchronous shared-memory SVRG method to solve non-convex problem. Our Shared-AsySVRG on shared-memory architecture has faster convergence rate than AsySGD. We prove that Shared-AsySVRG has a convergence rate of $O(1/T)$ for non-convex optimization.
- We extend asynchronous distributed-memory SVRG method to solve non-convex problem. Our Distributed-AsySVRG on distributed-memory architecture has faster convergence rate than AsySGD. We prove that Distributed-AsySVRG has a convergence rate of $O(1/T)$ for non-convex optimization.
- Both of Shared-AsySVRG and Distributed-AsySVRG have linear speedup when we increase the number of threads in a shared-memory architecture or workers in a distributed-memory architecture until reaching an upper bound.

## Notation

In this paper, we consider the following empirical loss minimization problem:

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

where $f(x)$ and $f_i(x)$ are Lipschitz smooth, they are not necessarily convex. In this paper, we assume both of them are non-convex.

Following the proof in (Lian et al. 2015; Reddi et al. 2016; Allen-Zhu 2016) for non-convex optimization, we use the weighted average of the $\ell_2$ norm of full gradient $||\nabla f(x)||^2$ as metric to analyze its convergence property. For further analysis, throughout this paper, we make the following assumptions for problem (1). All of them are very common assumptions in the theoretical analysis for asynchronous stochastic gradient descent method.

**Assumption 1** ***Independence:*** *All random samples $i$ are selected randomly and independently to each other.*

**Assumption 2** ***Unbiased Gradient:*** *The stochastic gradient $\nabla f_i(x)$ is unbiased:*

$$\mathbb{E}\left[\nabla f_i(x)\right] = \nabla f(x) \tag{2}$$

**Assumption 3** ***Lipschitz Gradient:*** *We say $\nabla f(x)$ is Lipschitz continuous, and it holds that:*

$$||\nabla f(x) - \nabla f(y)||_2 \leq L||x-y||_2 \tag{3}$$

*Throughout, we also assume that the function $\nabla f_i(x)$ is also Lipschitz continuous, so that $||\nabla f_i(x) - \nabla f_i(y)||_2 \leq L||x-y||_2$*

**Assumption 4** ***Maximum Time Delay:*** *Time delay variable $\tau$ of parameters in each worker is upper bounded, namely $\max \tau \leq \Delta$. In practice, $\Delta$ is related with the number of workers.*

## Asynchronous Mini-Batch Gradient Descent with Variance Reduction for Shared-Memory Architecture

In this section, we propose AsySVRG method for shared-memory architecture, and prove that it converges with rate $O(1/T)$. It is proved that SVRG has a convergence rate of $O(1/T)$ on non-convex problem (Reddi et al. 2016; Allen-Zhu 2016). In this section, we follow the convergence analysis in (Reddi et al. 2016), and extends it to asynchronous optimization on shared-memory architecture.

### Algorithm Description

Following the setting in (Lian et al. 2015), we define one iteration as a modification on any single component of $x$ in the shared memory. We use $x_t^{s+1}$ to denote the value of parameter $x$ in the shared memory after $(ms+t)$ iterations, and Equation (4) represents the update rule of parameter $x$ in iteration $t$:

$$(x_{t+1}^{s+1})_{k_t} = (x_t^{s+1})_{k_t} - \eta(v_t^{s+1})_{k_t}\,, \tag{4}$$

where $k_t \in \{1,...,d\}$ is a random index of component in $x \in \mathbb{R}^d$, and learning rate $\eta$ is constant. Descent direction $v_t^{s+1}$ is defined as follows:

$$v_t^{s+1} = \frac{1}{|I_t|}\sum_{i_t\in I_t}\left(\nabla f_{i_t}(\hat{x}_t^{s+1}) - \nabla f_{i_t}(\tilde{x}^s) + \nabla f(\tilde{x}^s)\right) \tag{5}$$

where $\tilde{x}^s$ denotes a snapshot of $x$ after every $m$ iterations. $i_t$ denotes the index of a sample, and $I_t$ is index set of mini-batch samples, and mini-batch size is $|I_t|$. The definition of $\hat{x}_t^{s+1}$ follows the analysis in (Lian et al. 2015), where $\hat{x}_t^{s+1}$ is assumed to be some earlier state of $x$ in the shared memory.

$$\hat{x}_t^{s+1} = x_t^{s+1} - \sum_{j\in J(t)}(x_{j+1}^{s+1} - x_j^{s+1}) \tag{6}$$

where $J(t) \in \{t-1, ...., t-\Delta\}$ is a subset of previous iterations, $\Delta$ is the upper bound of time delay. In Algorithm 1, we summarize the Shared-AsySVRG on shared-memory architecture.

### Convergence Analysis

In this section, we prove that our proposed Shared-AsySVRG method has a sub-linear convergence rate of $O(1/T)$ on non-convex problem. Different from AsySGD method, we are able to bound the variance of gradient update $v_t^{s+1}$ because of the variance reduction technique. And it is crucial for our convergence analysis.

**Algorithm 1** Shared-AsySVRG

```
Initialize $x^0 \in \mathbb{R}^d$.
for $s = 0, 1, 2, ,.., S-1$ do
    $\tilde{x}^s \leftarrow x^s$;
    Compute full gradient $\nabla f(\tilde{x}^s) \leftarrow \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\tilde{x}^s)$;
    Parallel Computation on Multiple Threads
    for $t = 0, 1, 2, ..., m-1$ do
        Randomly select mini-batch $I_t$ from $\{1, ....n\}$;
        Compute variance reduced gradient $v_t^{s+1}$:
        $v_t^{s+1} \leftarrow \frac{1}{|I_t|}\sum_{i_t \in I_t} \left(\nabla f_{i_t}(\hat{x}_t^{s+1}) - \nabla f_{i_t}(\tilde{x}^s) + \nabla f(\tilde{x}^s)\right)$
        Randomly select $k_t$ from $\{1, ..., d\}$;
        Update $(x_{t+1}^{s+1})_{k_t} \leftarrow (x_t^{s+1})_{k_t} - \eta(v_t^{s+1})_{k_t}$;
    end for
    $x^{s+1} \leftarrow x_m^{s+1}$;
end for
```

**Lemma 1** *As per the definition of the variance reduced gradient $v_t^{s+1}$ in Equation (5), we define,*

$$u_t^{s+1} = \frac{1}{|I_t|}\sum_{i_t \in I_t} \left(\nabla f_{i_t}(x_t^{s+1}) - \nabla f_{i_t}(\tilde{x}^s) + \nabla f(\tilde{x}^s)\right) \quad (7)$$

*We have the following inequality:*

$$\sum_{t=0}^{m-1} \mathbb{E}\left[||v_t^{s+1}||^2\right] \leq \frac{2d}{d - 2L^2\Delta^2\eta^2} \sum_{t=0}^{m-1} \mathbb{E}\left[||u_t^{s+1}||^2\right] \quad (8)$$

where $\mathbb{E}\left[||u_t^{s+1}||^2\right]$ is upper bounded in (Reddi et al. 2015).

$$\mathbb{E}\left[||u_t^{s+1}||^2\right] \leq 2\mathbb{E}\left[||\nabla f(x_t^{s+1})||^2\right] + \frac{2L^2}{b}\mathbb{E}\left[||x_t^{s+1} - \tilde{x}^s||^2\right] \quad (9)$$

From Lemma 1, we know that the variance of $v_t^{s+1}$ goes to zero when we reach the optimal solution if it exists. Thus, we can maintain learning rate as a constant in the optimization. Therefore, our Shared-AsySVRG has a faster convergence rate as follows:

**Theorem 1** *Suppose all assumptions of $f(x)$ satisfy. Let $c_m = 0$, learning rate $\eta > 0$ is constant, $\beta_t = \beta > 0$, $b = |I_t|$ denotes the size of mini-batch samples in each iteration. We define:*

$$\begin{aligned} c_t &= c_{t+1}\left(1 + \frac{\eta\beta_t}{d} + \frac{4L^2\eta^2}{(d - 2L^2\Delta^2\eta^2)b}\right) \\ &\quad + \frac{4L^2}{(d - 2L^2\Delta^2\eta^2)b}\left(\frac{L^2\Delta^2\eta^3}{2d} + \frac{\eta^2 L}{2}\right) \end{aligned} \quad (10)$$

$$\Gamma_t = \frac{\eta}{2d} - \frac{4}{d - 2L^2\Delta^2\eta^2}\left(\frac{L^2\Delta^2\eta^3}{2d} + \frac{\eta^2 L}{2} + c_{t+1}\eta^2\right) \quad (11)$$

*such that $\Gamma_t > 0$ for $0 \leq t \leq m-1$. Define $\gamma = \min_t \Gamma_t$, and $x^*$ is the optimal solution for non-convex problem. Then, Shared-AsySVRG has the following convergence rate in iteration $T$:*

$$\frac{1}{T}\sum_{s=0}^{S-1}\sum_{t=0}^{m-1} \mathbb{E}\left[||\nabla f(x_t^{s+1})||^2\right] \leq \frac{\mathbb{E}\left[f(x^0) - f(x^*)\right]}{T\gamma} \quad (12)$$

Now, we prove that our method has a convergence rate of $O(1/T)$ if problem is non-convex. If we represent $\gamma$ with known parameters, we have the following theorem.

**Theorem 2** *Suppose all assumptions of $f(x)$ satisfy. Let $\eta = \frac{u_0 b}{Ln^\alpha}$, where $0 < u_0 < 1$ and $0 < \alpha \leq 1$, $\beta = 2L$, $m = \lfloor \frac{dn^\alpha}{6u_0 b} \rfloor$ and $T$ is the number of total iterations. If the maximum time delay $\Delta$ satisfies the following condition:*

$$\Delta^2 < \min\left\{\frac{d}{2u_0 b}, \frac{3d - 28u_0 bd}{28u_0^2 b^2}\right\} \quad (13)$$

*Then there exists universal constant $u_0$ and $\sigma$, such that it holds $\gamma \geq \frac{\sigma b}{dLn^\alpha}$ and*

$$\frac{1}{T}\sum_{s=0}^{S-1}\sum_{t=0}^{m-1} \mathbb{E}\left[||\nabla f(x_t^{s+1})||^2\right] \leq \frac{dLn^\alpha \mathbb{E}\left[f(x^0) - f(x^*)\right]}{bT\sigma} \quad (14)$$

In (14), we can find out that the convergence rate has nothing to do with maximum time delay $\Delta$, if it is upper bounded. Thus in a specific domain, the negative effect of using stale information of parameter $x$ for approximating gradient evaluation vanishes, and a linear speedup is accessible when we increase the number of threads.

## Asynchronous Mini-Batch Gradient Descent with Variance Reduction for Distributed-Memory Architecture

In this section, we propose Distributed-AsySVRG algorithm for distributed-memory architecture, and prove that it converges with rate $O(1/T)$ on non-convex problem.

### Algorithm Description

In each iteration, parameter $x$ is updated through the following update rule,

$$x_{t+1}^{s+1} = x_t^{s+1} - \eta v_t^{s+1} \quad (15)$$

where learning rate $\eta$ is constant, $v_t^{s+1}$ represents variance reduced gradient, and it is defined as:

$$v_t^{s+1} = \frac{1}{|I_t|}\sum_{i_t \in I_t} \left(\nabla f_{i_t}(x_{t-\tau}^{s+1}) - \nabla f_{i_t}(\tilde{x}^s) + \nabla f(\tilde{x}^s)\right) \quad (16)$$

where $\tilde{x}^s$ means a snapshot of $x$ after every $m$ iterations, and $x_{t-\tau}^{s+1}$ denotes the current parameter used to compute gradient in the worker. $i_t$ denotes the index of a sample, $\tau$ denotes time delay of parameter in the worker, and mini-batch size is $|I_t|$. Suppose there are $K$ workers in total, and the number of dataset in worker $k$ is $n_k$. We summarize the Distributed-AsySVRG on distributed-memory architecture in the Algorithm 2 and Algorithm 3, Algorithm 2 shows operations in server node, and Algorithm 3 shows operations in worker node.

### Convergence Analysis

Similar to the convergence analysis in Shared-AsySVRG, we analyze the convergence rate for our proposed Distributed-AsySVRG in this section. It has been proved in (Reddi et al. 2015) that the variance of $v_t^{s+1}$ is upper bounded, and it goes to zero when $x$ is close to the optimal solution.

**Algorithm 2** Distributed-AsySVRG Server Node

```
Initialize $x^0 \in \mathbb{R}^d$.
for $s = 0, 1, 2, ,.., S-1$ do
    $\tilde{x}^s \leftarrow x^s$;
    flag = True
    Broadcast $\tilde{x}^s$ to all workers;
    Receive and compute: $\nabla f(\tilde{x}^s) \leftarrow \frac{1}{n}\sum_{k=1}^{K} \nabla^k f(\tilde{x}^s)$;
    Broadcast $\nabla f(\tilde{x}^s)$ to all workers;
    flag = False
    for $t = 0, 1, 2, ..., m-1$ do
        Receive variance reduced gradient $v_t^{s+1}$ from worker;
        Update $x_{t+1}^{s+1} \leftarrow x_t^{s+1} - \eta v_t^{s+1}$;
    end for
    $x^{s+1} \leftarrow x_m^{s+1}$;
end for
```

**Algorithm 3** Distributed-AsySVRG Worker Node $k$

```
if flag is True then
    Receive parameter $\tilde{x}^s$ from server;
    Compute and send full gradient $\nabla^k f(\tilde{x}^s)$:
    $\nabla^k f(\tilde{x}^s) = \sum_{i=1}^{n_k} \nabla_i f(\tilde{x}^s)$ ;
    Receive full gradient $\nabla f(\tilde{x}^s)$ from server;
else
    Receive parameter $x_{t-\tau}^{s+1}$ from server;
    Randomly select mini-batch $I_t$ from $\{1, ..., n_k\}$;
    Compute $v_t^{s+1}$ and send it to server:
    $v_t^{s+1} \leftarrow \frac{1}{|I_t|}\sum_{i_t \in I_t} \left(\nabla f_{i_t}(x_{t-\tau}^{s+1}) - \nabla f_{i_t}(\tilde{x}^s) + \nabla f(\tilde{x}^s)\right)$;
end if
```

**Theorem 3** *Suppose all assumptions of $f(x)$ satisfy. Let $c_m = 0$, learning rate $\eta > 0$ is constant, $\beta_t = \beta > 0$, $b$ denotes the size of mini-batch. We define:*

$$\begin{aligned} c_t &= c_{t+1}\left(1 + \eta\beta_t + \frac{4L^2\eta^2}{(1-2L^2\Delta^2\eta^2)b}\right) \\ &\quad + \frac{4L^2}{(1-2L^2\Delta^2\eta^2)b}\left(\frac{L^2\Delta^2\eta^3}{2} + \frac{\eta^2 L}{2}\right) \end{aligned} \quad (17)$$

$$\Gamma_t = \frac{\eta}{2} - \frac{4}{(1-2L^2\Delta^2\eta^2)}\left(\frac{L^2\Delta^2\eta^3}{2} + \frac{\eta^2 L}{2} + c_{t+1}\eta^2\right) \quad (18)$$

*such that $\Gamma_t > 0$ for $0 \le t \le m-1$. Define $\gamma = \min_t \Gamma_t$, $x^*$ is the optimal solution for non-convex problem. Then, we have the following convergence rate in iteration $T$:*

$$\frac{1}{T}\sum_{s=0}^{S-1}\sum_{t=0}^{m-1} \mathbb{E}\left[||\nabla f(x_t^{s+1})||^2\right] \le \frac{\mathbb{E}\left[f(x^0) - f(x^*)\right]}{T\gamma} \quad (19)$$

Representing $\gamma$ with known parameters, and then we have the following theorem.

**Theorem 4** *Suppose all assumptions of $f(x)$ satisfy. Let $\eta_t = \eta = \frac{u_0 b}{Ln^\alpha}$, where $0 < u_0 < 1$ and $0 < \alpha \le 1$, $\beta = 2L$, $m = \lfloor \frac{n^\alpha}{6u_0 b} \rfloor$ and $T$ is total iteration. If the maximum time delay $\Delta$ is upper bounded by:*

$$\Delta^2 < \min\left\{\frac{1}{2u_0 b}, \frac{3 - 28u_0 b}{28u_0^2 b^2}\right\} \quad (20)$$

*then there exists universal constant $u_0$, $\sigma$, such that if it holds $\gamma \ge \frac{\sigma b}{Ln^\alpha}$, we have the following inequality:*

$$\frac{1}{T}\sum_{s=0}^{S-1}\sum_{t=0}^{m-1} \mathbb{E}\left[||\nabla f(x_t^{s+1})||^2\right] \le \frac{Ln^\alpha \mathbb{E}\left[f(x^0) - f(x^*)\right]}{bT\sigma} \quad (21)$$

Therefore, it is obvious that our proposed Distributed-AsySVRG method has sub-linear convergence rate of $O(1/T)$, and is much faster than the AsySGD with convergence rate of $O(1/\sqrt{T})$ (Lian et al. 2015). From inequality (21), we know that the convergence rate has nothing to do with $\Delta$ if it is upper bounded, linear speedup is also accessible when we increase the number of workers in a cluster.

## Experiments

In this section, we perform experiments on shared-memory architecture and distributed-memory architecture respectively. One of the main purposes of our experiments is to validate the faster convergence rate of asySVRG method, and the other purpose is to demonstrate its linear speedup property. The speedup we consider in this paper is running time speedup when they reach similar performance, e.g. similar training loss function value. Given $K$ workers, running time speedup is defined as,

$$\text{Time speedup} = \frac{\text{Running time for the serial computation}}{\text{Running time of using } K \text{ workers}} \quad (22)$$

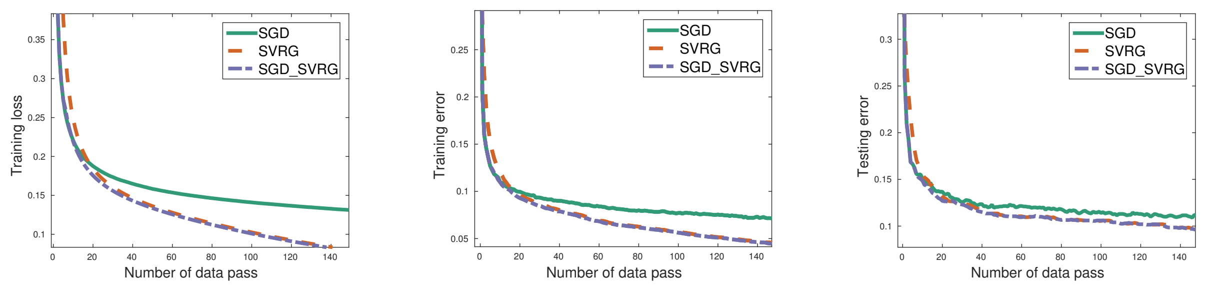
### Shared-Memory Architecture

We conduct experiments on a machine which has 2 sockets, and each socket has 18 cores. OpenMP library [1] is used to handle shared-memory parallelism. We consider the multi-class classification task on MNIST dataset (LeCun et al. 1998), and use $10,000$ training samples and $2,000$ testing samples in the experiment. Each image sample is a vector of 784 pixels. We construct a toy three-layer neural network $(784 \times 100 \times 10)$, where ReLU activation function is used in the hidden layer. We train this neural network with softmax loss function, and $\ell_2$ regularization with weight $C = 10^{-3}$. We set mini-batch size $|I_t| = 10$, and inner iteration length $m = 1,000$. Updating only one component of $x$ in each iteration is too time consuming, therefore we randomly select and update $1,000$ components.

We compare following three methods in the experiment:

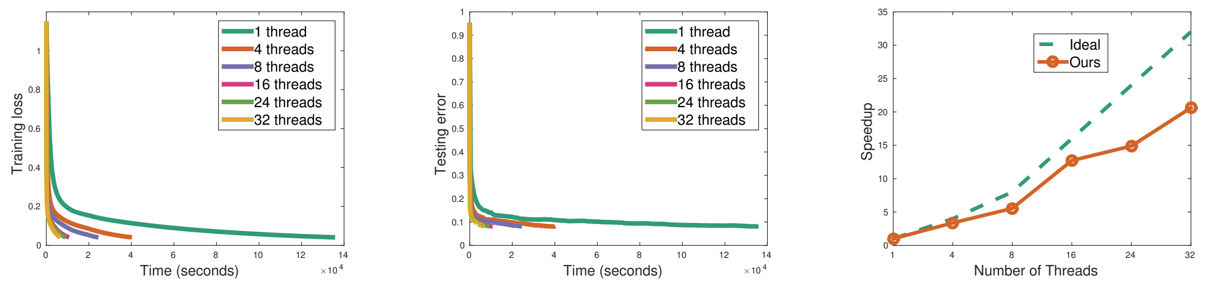
- SGD: We implement stochastic gradient descent (SGD) algorithm and train with the best tuned learning rate. In our experiment, we use polynomial learning rate $\eta = \frac{\alpha}{(1+s)^\beta}$, where $\alpha$ denotes initial learning rate and we tune it from $\{1e^{-2}, 5e^{-2}, 1e^{-3}, 5e^{-3}, 1e^{-4}, 5e^{-4}, 1e^{-5}, 5e^{-5}\}$, $\beta$ is tuned from in $\{0, 0.1, 0.2, ..., 1\}$ and $s$ denotes the epoch number.

[1] https://openmp.org



(a) (b) (c)

Figure 1: Comparison of three methods: SGD, SVRG, SGD_SVRG on MNIST dataset. Figure 1a shows the convergence of loss function value on training dataset. Figure 1b shows the convergence of training error and Figure 1c shows the convergence of test error.



(a) (b) (c)

Figure 2: We run Shared-AsySVRG on a machine with different number of threads from 1 to 32. Figure 2a shows the convergence of training loss value with respect to time. Figure 2b shows the convergence of error rate on testing data. Figure 2c represents the running time speedup when we use different threads, where the dashed line represents ideal linear speedup.

- SVRG: We implement our Shared-AsySVRG method and train with the best tuned constant learning rate $\alpha$.
- SGD_SVRG: SVRG method is sensitive to initial point, and it is slower than SGD at first few iterations. Thus, we apply Shared-AsySVRG on a pre-trained model learned by SGD. In the experiment, we use a pre-trained model after running 10 epochs of SGD method.

We evaluate three compared methods on MNIST dataset, and each method trains with the best tuned learning rate. Figure 1 shows the convergence of each method with respect to different criterion: loss function value on training dataset, training error, and testing error. Figure 1a shows the curves of training loss function value, it is clear that SGD method converges faster than SVRG method in the first 20 iterations, and after that, SVRG method outperforms SGD. SGD_SVRG method initializes with a pre-trained model, and it has the best performance. Figure 1b and Figure 1c present the performance of each method on training error and testing error respectively. We can conclude that SVRG and SGD_SVRG method have better performance on the long run, and SGD_SVRG method has the fastest convergence.

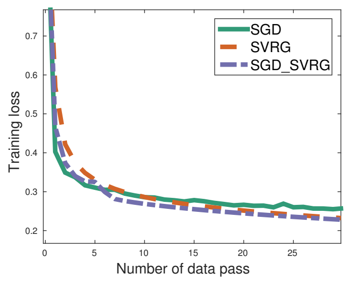
To demonstrate that our proposed Shared-AsySVRG method has linear speedup when we increase the number of workers, we also evaluate Shared-AsySVRG with different number of threads, and Figure 2 presents the result of our experiment. In Figure 2a, all curves are reaching the similar training loss value. As we can see, the more threads we use in the computation, the less time we need to achieve a similar accuracy. This result is reasonable, because when we distribute the whole work to multiple workers, each worker focuses on its own subset independently and parallelly. The ideal result of parallel computation is linear speedup, namely if we use $K$ threads, its running time should be $\frac{1}{K}$ of the time when we just use a single thread. Figure 2c shows the ideal speedup and actual speedup in our experiment. We can find out that a nearly linear speedup is accessible when we increase the thread number. When the number of threads exceeds a threshold, performance will degrade. These findings in the experiment are compatible with our theoretical analysis.

## Distributed-Memory Architecture

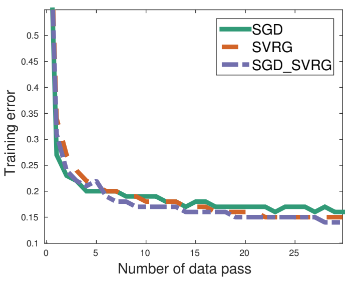
We conduct distributed-memory architecture experiment on AWS platform[2], and each node is a t2.micro instance with one virtual CPU. Each server and worker takes a single node. The point to point communication between server and workers are handled by MPICH library[3]. We use CIFAR-10 dataset (Krizhevsky and Hinton 2009) in the experiment, and this

[2]https://aws.amazon.com/

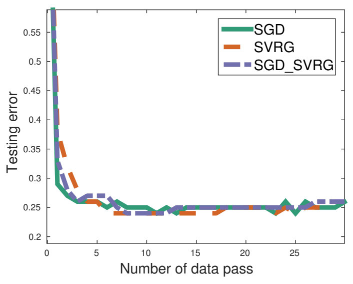
[3]http://www.mpich.org/



(a)

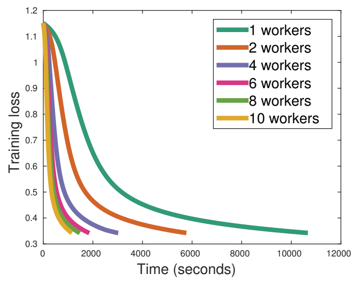


(b)

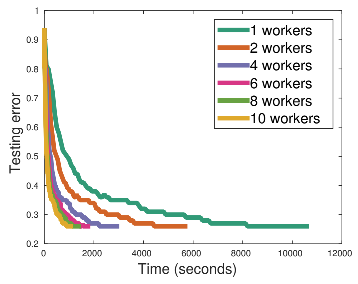


(c)

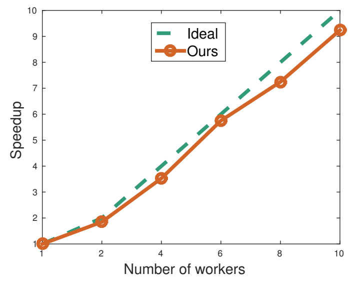
Figure 3: Comparison of three methods: SGD, SVRG, SGD_SVRG on CIFAR-10 dataset. Figure 3a shows the convergence of loss function value on training dataset. Figure 3b shows the convergence of training error and Figure 3c shows convergence of test error.



(a)



(b)



(c)

Figure 4: We run Distributed-AsySVRG on multiple machines from 1 to 10. Figure 4a shows the convergence of training loss value with respect to time. Figure 4b shows the convergence of error rate on testing data. Figure 4c represents the running time speedup when using different number of workers, where the dashed line denotes ideal linear speedup.

dataset has 10 classes of color image of size $32 \times 32 \times 3$. We use $20,000$ samples as training data and $4,000$ samples as testing data. We use a pre-trained CNN model in TensorFlow tutorial (Abadi et al. 2016), and extract features from second fully connected layer. Thus, each sample is a vector of size 384. We construct a three-layer fully connected neural network $(384 \times 50 \times 10)$. In the hidden layer, we use ReLU activation function. We train this model with softmax loss, and $\ell_2$ regularization with weight $C = 1e^{-4}$. In this experiment, mini-batch size $|I_t| = 10$, and the inner loop length $m = 2,000$. We use the same compared methods as in the last section, except that SGD_SVRG method is initialized with parameters learned after 1 epoch of SGD.

Performances of all three methods are presented in Figure 3. Curves in Figure 3a show that SGD is the fastest method in the first few iterations, after that, SVRG-based method will outperform it. It is obvious that SGD_SVRG has better convergence rate than SVRG method. We can also draw a similar conclusion from Figure 3b. In Figure 3c, it shows that the test error performance of three compared methods are comparable.

We also test our Distributed-AsySVRG method with different number of workers, and Figure 4 illustrates the results of our experiment. It is easy to know that when the number of workers increases, our method has a near linear speedup.

## Conclusion

In this paper, we propose and analyze asynchronous mini-batch gradient descent method with variance reduction for non-convex optimization on two distributed architectures: shared-memory architecture and distributed-memory architecture. We analyze their convergence rate and prove that both of them can get a convergence rate of $O(1/T)$ for non-convex optimization. Linear speedup is accessible when we increase the number of workers $K$, if $K$ is upper bounded. Experiment results on real dataset also demonstrate our analysis.

## References

Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

Agarwal, A., and Duchi, J. C. 2011. Distributed delayed stochastic optimization. In *Advances in Neural Information Processing Systems*, 873–881.

Allen-Zhu, Zeyuan; Hazan, E. 2016. Variance reduction for faster non-convex optimization. *arXiv preprint arXiv:1603.05643*.

Bottou, L. 2010. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*. Springer. 177–186.

Dean, J.; Corrado, G.; Monga, R.; Chen, K.; Devin, M.; Mao, M.; Senior, A.; Tucker, P.; Yang, K.; Le, Q. V.; et al. 2012. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, 1223–1231.

Defazio, A.; Bach, F.; and Lacoste-Julien, S. 2014. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems*, 1646–1654.

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

Ghadimi, S.; Lan, G.; and Zhang, H. 2016. Mini-batch stochastic approximation methods for nonconvex stochastic composite optimization. *Mathematical Programming* 155(1-2):267–305.

Huo, Z., and Huang, H. 2016. Distributed asynchronous stochastic dual coordinate ascent without duality. *arXiv preprint arXiv:1605.09066*.

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

Konečnỳ, J.; Liu, J.; Richtárik, P.; and Takáč, M. 2014. ms2gd: Mini-batch semi-stochastic gradient descent in the proximal setting. *arXiv preprint arXiv:1410.4744*.

Krizhevsky, A., and Hinton, G. 2009. Learning multiple layers of features from tiny images.

Lan, G. 2012. An optimal method for stochastic composite optimization. *Mathematical Programming* 133(1-2):365–397.

Langford, J.; Smola, A.; and Zinkevich, M. 2009. Slow learners are fast. *arXiv preprint arXiv:0911.0491*.

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *Nature* 521(7553):436–444.

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

Lian, X.; Huang, Y.; Li, Y.; and Liu, J. 2015. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*, 2719–2727.

Liu, J.; Wright, S. J.; and Sridhar, S. 2014. An asynchronous parallel randomized kaczmarz algorithm. *arXiv preprint arXiv:1401.4780*.

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

Ngiam, J.; Coates, A.; Lahiri, A.; Prochnow, B.; Le, Q. V.; and Ng, A. Y. 2011. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 265–272.

Recht, B.; Re, C.; Wright, S.; and Niu, F. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, 693–701.

Reddi, S. J.; Hefny, A.; Sra, S.; Póczos, B.; and Smola, A. J. 2015. On variance reduction in stochastic gradient descent and its asynchronous variants. In *Advances in Neural Information Processing Systems*, 2629–2637.

Reddi, S. J.; Hefny, A.; Sra, S.; Póczós, B.; and Smola, A. 2016. Stochastic variance reduction for nonconvex optimization. *arXiv preprint arXiv:1603.06160*.

Zhang, R., and Kwok, J. 2014. Asynchronous distributed admm for consensus optimization. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 1701–1709.

Zhang, R.; Zheng, S.; and Kwok, J. T. 2015. Fast distributed asynchronous sgd with variance reduction. *arXiv preprint arXiv:1508.01633*.

Zhao, S.-Y., and Li, W.-J. 2016. Fast asynchronous parallel stochastic gradient descent: A lock-free approach with convergence guarantee.