# An efficient, distributed stochastic gradient descent algorithm for deep-learning applications

Guojing Cong, Onkar Bhardwaj, Minwei Feng
IBM TJ Watson Research Center
1101 Kitchawan Road, Yorktown Heights, NY, 10598
{gcong,obhardw,mfeng}@us.ibm.com

***Abstract*—Parallel and distributed processing is employed to accelerate training for many deep-learning applications with large models and inputs. As it reduces synchronization and communication overhead by tolerating stale gradient updates, asynchronous stochastic gradient descent (ASGD), derived from stochastic gradient descent (SGD), is widely used. Recent theoretical analyses show ASGD converges with linear asymptotic speedup over SGD.**

**Oftentimes glossed over in theoretical analysis are communication overhead and practical learning rates that are critical to the performance of ASGD. After analyzing the communication performance and convergence behavior of ASGD using the *Downpour* algorithm as an example, we demonstrate the challenges for ASGD to achieve good practical speedup over SGD. We propose a distributed, bulk-synchronous stochastic gradient descent algorithm that allows for sparse gradient aggregation from individual learners. The communication cost is amortized explicitly by a gradient aggregation interval, and global reductions are used instead of a parameter server for gradient aggregation. We prove its convergence and show that it has superior communication performance and convergence behavior over popular ASGD implementations such as *Downpour* and *EAMSGD* for deep-learning applications.**

## I. Introduction

To solve large-scale machine learning problems on modern computer systems, parallel and distributed processing is adopted for stochastic optimization methods (e.g., see [20], [5], [19], [28], [4], [29]). Efficient parallelization becomes critical to accelerating long-running machine learning applications. Asynchronous stochastic gradient descent (ASGD), derived from stochastic gradient descent (SGD), is popular in current deep-learning applications and studies (e.g., see [5], [20]). ASGD exploits data parallelism by employing multiple learners each computing gradient updates on their inputs. The gradients learned by each learner are aggregated typically through a central parameter server (e.g., see [11]). The parameters maintained at the central parameter server are updated asynchronously. Asynchronous gradient updates reduce synchronization and communication overhead that can otherwise become prohibitive on a cluster even with a modest number of learners.

The mathematical foundations of parallel methods including ASGD are established in recent studies. The convergence for SGD has been extensively studied (e.g., see [21], [3], [17], [22], [7]). The convergence rate is $O\left(1/\sqrt{S}\right)$ for non-convex problems and $O(1/S)$ for convex problems, with $S$ being the number of samples processed. Regarding parallel variants of SGD, Dekel *et al.* [6] extends these results to the setting of synchronous SGD with $p$ learners and show that it has convergence rate of $O\left(1/\sqrt{pK}\right)$ for non-convex objectives, with $K$ being the number of samples processed by each learner. Note that this agrees with the prior analyses for SGD since in this case $S = pK$ is the number of samples processed. Hogwild! is a lockfree implementation of ASGD, and Niu *et al.* [20] proves its convergence for strongly convex problems with theoretical linear speedup over SGD. *Downpour* is another ASGD implementation with resilience against machine failures [5]. Lian *et al.* [13] show that as long as the gradient staleness is bounded by the number of learners, ASGD converges for non-convex problems (with certain assumptions) with asymptotic linear speedup over SGD.

Unfortunately, theoretical convergence does not guarantee practical efficiency for ASGD. When deployed on a cluster, communication cost can dominate the execution time when the model is large and/or gradient update is frequent. Although ASGD has the same asymptotic convergence rate as SGD when the staleness of gradient update is bounded, the learning rate assumed for proving ASGD convergence can be too small for practical purposes. It is also difficult for an ASGD implementation to control the staleness in gradient updates as it is influenced by the relative processing speeds of learners and their positions in the communication network. Furthermore, the parameter server presents performance challenges on platforms with many GPUs. On such platforms, a single parameter server oftentimes does not serve the aggregation requests fast enough. A sharded server alleviates the aggregation speed problem but introduces inconsistencies for parameters distributed on multiple shards. Communication between the parameter server (typically on CPUs) and the learners (on GPUs) is likely to remain a bottleneck in future systems.

We propose a distributed, bulk-synchronous SGD algorithm that allows for sparse gradient aggregation to effectively minimize the communication overhead. We call this algorithm sparse aggregation SGD (*SASGD*). Instead of a parameter server, the learners in *SASGD* communicate the gradients learned with each other at regular intervals through global reductions. Rather than relying on asynchrony that reduces communication overhead but has adverse impact on practical convergence, we make the communication interval $T$ a parameter in *SASGD*. The communication time is amortized among the data samples processed within each interval and becomes negligible if $T$ is large enough. Compared to asynchronous updates to a parameter server, global reduction minimizes the amount of data transported in the system. Also on current and emerging computer platforms that support high bandwidth direct communication among GPUs (e.g., GPU-direct [8]), global reduction does not involve CPUs and avoids multiple costly copies through the software layers. We demonstrate the convergence behavior of *SASGD* relative to $T$, and analyze its sample complexity (measured as the number of data samples required to reach certain training quality). We show that sample complexity of *SASGD* increases with $T$, and thus the practitioners need to explicitly balance the decrease of communication time and the increase of iterations through an appropriately chosen $T$. In contrast, neither communication time nor sample complexity can be effectively controlled in most ASGD implementations, even for some recent ASGD variants that also employ a gradient update interval to tolerate more staleness in gradient updates.

Our experiments with two deep-learning applications demonstrate the superior performance of *SASGD* over two popular ASGD implementations: *Downpour* [5] and *EAMSGD* [29]. In *EAMSGD*, global gradient aggregation among learners simulates an elastic force that links the parameters they compute with a center variable stored by the parameter server. On our target platform, when $T$ is small, *SASGD* significantly reduces the communication time in comparison to *Downpour* and *EAMSGD* while achieving similar training and test accuracies. The training time reduction is up to 50%. When $T$ is large, *SASGD* achieves much better training and test accuracies than *Downpour* and *EAMSGD* after the same amount of data samples are processed. For example, with 16 GPUs and $T = 50$, *SASGD* achieves up to 3% more accuracy for the first application and 50% more accuracy for the second application than *Downpour* and *EAMSGD*.

The rest of the paper is organized as follows. Section II investigates the practical efficiency of ASGD for two deep-learning applications. It shows the communication overhead is significant and argues that in practice only sublinear speedups may be observed. Section III introduces *SASGD*, and analyzes its convergence behavior. Section IV shows the performance of *SASGD* in comparison to *Downpour* and *EAMSGD*. Section V presents our conclusion and future work.

| |
|---|
| Input: minibatch of $M$ RGB images |
| $\downarrow$ |
| Convolution: (nfeat, nkern, height, width) = $(3, 64, 5, 5)$ |
| Rectified Linear Unit (ReLU) |
| Max-Pooling: (height, width) = $(2, 2)$ |
| Dropout: prob. = $0.5$ |
| $\downarrow$ |
| Convolution: (nfeat, nkern, height, width) = $(64, 128, 3, 3)$ |
| Rectified Linear Unit (ReLU) |
| Max-Pooling: (height, width) = $(2, 2)$ |
| Dropout: prob. = $0.5$ |
| $\downarrow$ |
| Convolution: (nfeat, nkern, height, width) = $(128, 256, 3, 3)$ |
| Rectified Linear Unit (ReLU) |
| Max-Pooling: (height, width) = $(2, 2)$ |
| Dropout: prob. = $0.5$ |
| $\downarrow$ |
| Convolution: (nfeat, nkern, height, width) = $(256, 128, 2, 2)$ |
| Rectified Linear Unit (ReLU) |
| Max-Pooling: (height, width) = $(2, 2)$ |
| Dropout: prob. = $0.5$ |
| $\downarrow$ |
| Fully connected layer: $128 \times 10$ |
| $\downarrow$ |
| Cross-entropy error |

TABLE I: Convolutional Neural Network for CIFAR10. For convolutional layers nfeat denotes the number of input feature maps and nkern denotes the number of kernels.

| |
|---|
| Input: minibatch of $M$ sentences translated to their precomputed word2vec representation |
| $\downarrow$ |
| Fully connected layer: $100 \times 200$ |
| Tanh layer |
| $\downarrow$ |
| Temporal Convolution: (nkern, window size) = $(1000, 2)$ |
| Max-Pooling: (height, width) = $(2, 1)$ |
| Tanh layer |
| $\downarrow$ |
| Fully connected layer: $1000 \times 1000$ |
| Tanh layer |
| Fully connected layer: $1000 \times 311$ |
| $\downarrow$ |
| Cross-entropy error |

TABLE II: Neural Network for *NLC-F*. For temporal convolutional layer nkern denotes the number of kernels.

## II. Practical efficiency of ASGD

We evaluate the practical efficiency of ASGD using *Downpour* as an example for two deep-learning applications. Training in machine learning applications typically takes many passes of the input data before convergence. One pass of the input is called an epoch. To take advantage of the parallelism in the architecture (e.g., SIMD units or threads in GPUs), the training samples are typically processed in groups called minibatches.

**Experiment setup**: The two applications in our experiments both employ ASGD for training. They use different deep-learning network models, but the training algorithm is the same. So we consider them as ASGD with two different data sets. One is the CIFAR data set (*CIFAR-10*) [9]. The other is an in-house natural language processing data set from the finance industry, and we call it *NLC-F*. For the *CIFAR-10* data set, the application trains a model to recognize the input images, and for the *NLC-F* data set, the application detects the sentiments expressed in the input sentences. *CIFAR-10* contains $50,000$ training images and $10,000$ test images, each associated with 1 out of 10 possible labels, whereas *NLC-F* consists of 2500 input sentences and 311 output labels.

The multi-layer convolutional neural networks used for

*CIFAR-10* and *NLC-F* are shown in Tab. I and Tab. II, respectively. The network used for *CIFAR-10* is a fairly standard convolutional network design [1] consisting of a series of convolutional layers interspersed by max-pooling layers. We choose this network instead of other networks with deeper structures such as AlexNet [10] or GoogLeNet [24] to limit the amount of training time. The approaches discussed in this paper work for these networks also. The outputs of convolutional layers in this network are filtered with rectified linear units before max-pooling is applied. Additionally, *Dropout* is applied as regularization [23]. The last layer is a fully connected layer. The network for *NLC-F* contains a temporal convolution layer [2] and a few fully connected layers. *Tanh* units are used for non-linearities instead of rectified linear units. The number of parameters is about $0.5$ million in the *CIFAR-10* network and about 2 million in the *NLC-F* network. Both networks use cross-entropy error between the input labels and the predicted labels as the error measure. In both applications, gradient descent is implemented with Torch [25] and the communication is implemented using CUDA-aware openMPI 2.0 through the mpiT library [15].

We run our experiments on an IBM Power8 host with an OSS high-density compute accelerator [18]. The OSS accelerator contains 8 NVIDIA Tesla K80 GPUs connected by PCIe switches forming a binary tree. The host contains two Power8 chips, each with 12 cores running at 3.1 GHz. In the ASGD implementations, to fully utilize both the host and the accelerators, the learners are run on the GPUs, while the (sharded) parameter server is run on the host Power8 CPUs.

## A. ASGD communication overhead

We experiment with 1, 2, 4, and 8 learners (each learner on one GPU) for *CIFAR-10* and *NLC-F*. All learners in *Downpour* have similar behavior. The amount of computation and communication in each learner remains constant between epochs. Fig. 1 shows the breakdown of epoch time into computation and communication for one learner. Minibatch sizes 64 and 1[1] are used for training on *CIFAR-10* and *NLC-F*, respectively. Of the two bars in each group, the one on the left is for *NLC-F* and the one on the right is for *CIFAR-10*.

In Fig. 1, communication dominates for *NLC-F*, accounting for more than 60% of the epoch time. For *CIFAR-10*, with 1 learner the communication time is around 20%, and increases to about 30% with 8 learners. In *Downpour*, from the perspective of a learner, communication includes sending its computed gradients to the parameter server, waiting for the server to aggregate the gradients, and receiving parameters from the parameter server. Communication time becomes significant as the number of learners increases and/or the network size increases.

## B. ASGD convergence relative to SGD

We analyze the convergence behavior of ASGD in comparison to SGD. We make standard assumptions about the surface properties of the objective function.

[1]Minibatch size 1 is used because it gives the best test accuracy for *NLC-F*

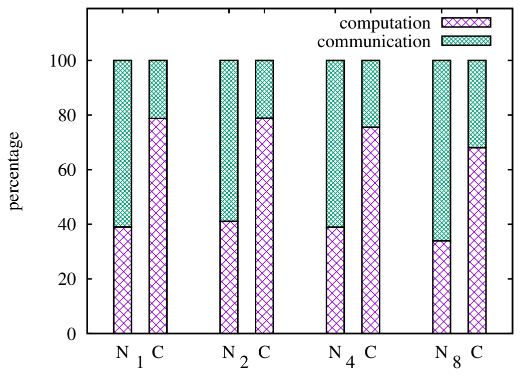


Fig. 1: Breakdown of epoch time

- Unbiased gradient: We assume that the partial gradient $G(x, z)$ of $f(\cdot)$ is an unbiased estimator of true gradient, where $x$ is any parameter vector and $z$ is a mini-batch of randomly selected $M$ samples. In other words, we assume $\mathbb{E}(G(x, z)) = \nabla f(x)$ where the expectation is with respect to randomly selected mini-batches.
- Bounded variance: We assume that the variance of partial gradient with respect to randomly selected mini-batches is bounded, i.e., $\mathbb{E}(\|G(x, z) - \nabla f(x)\|^2) \leq \sigma^2$.
- Lipschitzian gradient: We assume that there exists a constant $L$ such that $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$ for any two parameter vectors $x, y$.

The notations used in the convergence behavior analysis and analyses in later sections are introduced in Tab. III. Lian *et al.* [13] show that with a small enough constant learning rate $\gamma$ and ignoring communication overhead, linear asymptotic speedup may be achieved for ASGD over SGD after sufficient number of iterations if the staleness of the gradient is bounded by the number of learners. As it is not measured by comparing wall-clock times, we refer to the speedup as convergence speedup. We analyze the convergence speedup of ASGD and SGD for a finite number of iterations, and show that it can be sublinear for practical purposes.

Let $\bar{R}_K$ denote the average expected gradient norm after the first $K$ updates of ASGD. Then from Theorem 1 in [13] the convergence rate guarantee for ASGD expressed as average gradient norm is

$$\bar{R}_K \leq \frac{2D_f}{MK\gamma} + \sigma^2 L\gamma + 2\sigma^2 L^2 Mp\gamma^2 \quad (1)$$

$$\text{s.t.} \quad LM\gamma + 2L^2M^2p^2\gamma^2 \leq 1 \quad (2)$$

The terms independent of the number of updates $K$ in Equation 1 indicate that with a constant learning rate, there is a limit on how close the algorithm can reach to the optimum without lowering the learning rate.

Analyzing average gradient norm, we get the following theorem about the gap between the guarantee for one learner and multiple learners

| | |
|---|---|
| $f(\cdot) :=$ | A non-convex objective function |
| $x :=$ | Parameter vector |
| $x_1 :=$ | Initial parameter vector |
| $x^* :=$ | A local optima towards which the algorithm converges |
| $D_f :=$ | $f(x_1) - f(x^*)$ |
| $L :=$ | The constant corresponding to Lipschitzian gradient |
| $M :=$ | Minibatch size |
| $T :=$ | No. of local updates after which a global aggregation is done |
| $p :=$ | Number of learners |
| $\gamma :=$ | Learning rate |
| $\sigma^2 :=$ | Upper bound on $\mathbb{E}(\|G(x, z_i) - \nabla f(x)\|)^2$ where $G(x, z_i)$ is the stochastic gradient of $f(\cdot)$ with respect to $i^{th}$ sample $z_i$ |
| $S :=$ | The number of total samples processed |

TABLE III: Notation

**Theorem 1.** *Let $p > 1$ be the number of learners and let $\alpha = \sqrt{\frac{K\sigma^2}{MLD_f}} \leq p$, then the optimal ASGD convergence rate guarantee for 1 learner and $p$ learners can differ by a factor of approximately $\frac{p}{\alpha}$.*

*Proof.* We have $\gamma = c \cdot \sqrt{D_f/(MKL\sigma^2)} = \frac{c}{\alpha ML}$ from the definition of $\alpha$. Substituting this in Equation 1, we get

$$\bar{R}_K \quad \leq \quad \left(\frac{2}{c} + c + \frac{2pc^2}{\alpha}\right) \cdot \sqrt{\frac{D_f L\sigma^2}{MK}} \tag{3}$$

From the definition of $\alpha$, we have $K = \alpha^2 MLD_f/\sigma^2$. Using it in the above equation, we get

$$\bar{R}_K \quad \leq \quad \left\{\left(\frac{2}{c} + c + \frac{2pc^2}{\alpha}\right) \cdot \frac{1}{\alpha}\right\} \cdot \frac{\sigma^2}{M} \tag{4}$$

Similarly, given $\gamma = c \cdot \sqrt{\frac{D_f}{MKL\sigma^2}} = \frac{c}{\alpha ML}$, the condition in Equation 2 can also be expressed as:

$$\frac{c}{\alpha} + \frac{2p^2c^2}{\alpha^2} \leq 1 \quad \Rightarrow \quad 2p^2c^2 + \alpha c - \alpha^2 \leq 0$$

Since learning rate (and hence $c$) is always positive, the above equation gives us

$$0 \leq c \leq \frac{\alpha}{4p^2} \cdot (-1 + \sqrt{1 + 8p^2})$$

Thus finding the optimal learning rate (within the regime of Equation 1 and 2) is equivalent to solving the following

$$\text{minimize} \qquad \left(\frac{2}{c} + c + \frac{2pc^2}{\alpha}\right) \cdot \frac{1}{\alpha} \cdot \frac{\sigma^2}{M} \tag{5}$$

$$\text{s.t.} \qquad 0 \;\leq\; c \;\leq\; \frac{\alpha}{4p^2} \cdot (-1 + \sqrt{1 + 8p^2}) \tag{6}$$

Now, by means of solving the above optimization, we will investigate how much the convergence guarantee can differ as the number of learners increase. In particular, we will look at the difference in the guarantee for 1 learner and $p_{max}$ learners where $p_{max} \geq 16$. Taking the derivative of Equation 5 with respect to $c$ and setting it to 0, we get the following

$$4pc^3 + \alpha c^2 - 2\alpha = 0 \tag{7}$$

Let $c_1^*$ and $c_{p_{max}}^*$ denote the solutions to the above equation for 1 and $p_{max}$ learners respectively.

$$\text{For } p = 1: \qquad \bar{R}_K \lesssim 2\sqrt{2} \cdot \frac{\sigma^2}{\alpha M} \tag{8}$$

For $p = p_{max}$ and $16 \leq \alpha \leq T_{max}$, the cubic term dominates in Equation 7. $c_{p_{max}}^* = \alpha/(\sqrt{2}p_{max})$. Thus for $16 \leq \alpha \leq p_{max}$, Equation 4 becomes

$$\text{For } p = p_{max}: \qquad \bar{R}_K \lesssim \frac{2\sqrt{2}\,p_{max}}{\alpha} \cdot \frac{\sigma^2}{\alpha M} \tag{9}$$

Thus comparing Equation 8 and 9, we see that the ASGD convergence guarantee for $p = 1$ and $p = p_{max}$ learners can differ by a factor of $\frac{p_{max}}{\alpha}$ for $16 \leq \alpha \leq p_{max}$. □

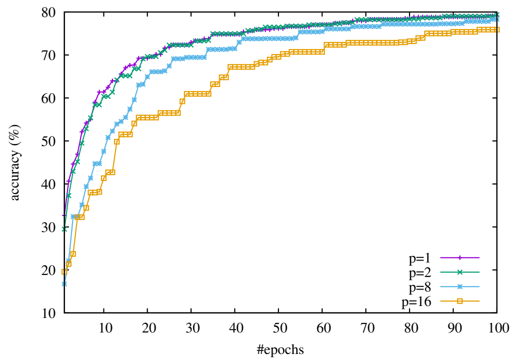


Fig. 2: ASGD convergence for CIFAR10 with $\gamma = 0.1$

Theorem 1 predicts that the convergence guarantees after processing $K$ minibatches can differ significantly between 1 learner and $p > 1$ learners. For example, when $p = 32$, $\alpha$ is roughly 16 for 50 epochs of updates with *CIFAR-10*. The convergence guarantee between SGD and ASGD with $p = 32$ can differ by 2.

We run *Downpour* with $p = 1$, 2, 8, and 16 learners to evaluate its convergence speedup over SGD. For $p = 16$ we run 2 learners per GPU using CUDA multi-process service [16]. We use the learning rate of $\gamma = 0.1$, and compare how test accuracy increases with respect to the number of epochs. The results are shown in Fig. 2.

In Fig. 2, as $p$ increases, with the same number of epochs, the accuracy gap between *Downpour* and SGD (*Downpour* with $p$=1) increases. Since the accuracy converges slower and slower as $p$ increases, linear convergence speedup is not observed. That is, to reach the same accuracy achieved by SGD, *Downpour* with $p > 1$ learners needs to process more data samples.

Linear convergence speedup is observed, however, if we use the learning rate, $\sqrt{\frac{D_f}{MKL\sigma^2}}$, derived from the convergence analysis of ASGD by Lian *et al.* [13]. We estimate the Lipschitz constant $L$ and an upper bound on gradient variance $\sigma^2$ for *CIFAR-10*. We bound $D_f$ as $f(x_1)$ and use $MK = 500,000$.

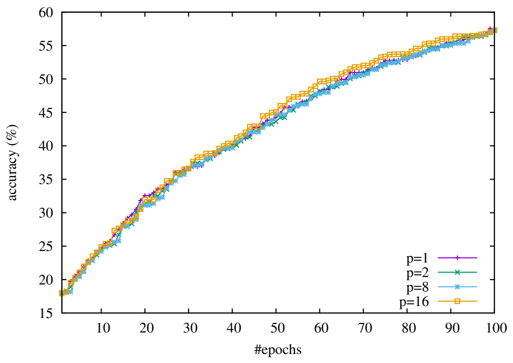


Fig. 3: ASGD convergence for CIFAR10 with $\gamma = 0.005$

Using our estimated quantities, $\sqrt{\frac{D_f}{MKL\sigma^2}}$ is approximately 0.005, much smaller than 0.1. With $\gamma = 0.005$, we experiment with $p = 1$, 2, 8, and 16 learners. The results are shown in Fig. 3. In Fig. 3, indeed linear convergence speedup is observed for $p > 1$ (implied by the curves for different $p$ overlapping almost perfectly). However, $\gamma = 0.005$ is clearly sub-optimal for *CIFAR-10* as it achieves only about 57% accuracy compared to 80% accuracy achieved with $\gamma = 0.1$.

## III. Sparse-aggregation SGD

To reduce the communication overhead of distributed training, some recent studies propose using large minibatch sizes (e.g., see [12]). As large batch size has been found to be inefficient for gradient descent with respect to training accuracy [26], additional measures such as reducing the variance among the gradients are needed [14], [27]. Some implementations tolerate even more stale gradient updates. In fact, *Downpour* itself has a version that processes multiple minibatches before sending gradients asynchronously to the parameter server. However, its practical behavior turns out to be erratic. Zhang *et al.* [29] proposes an ASGD variant called *EAMSGD* that enforces some constraints on the parameters when the gradient update interval increases. *EAMSGD* is shown to have superior performance over *Downpour*.

While providing some degree of fault tolerance, the parameter server in *Downpour* and *EAMSGD* also poses performance challenges. A sharded server is used in these implementations for fast gradient aggregation. Although more scalable, sharded server suffers increased stochasticity and inconsistency in gradient updates. Additionally, the amount of data transported with a parameter server increases linearly with the number of learners. To balance communication with computation for high performance, tremendous bandwidth is needed between the learners and the parameter server.

We propose a bulk-synchronous, distributed SGD algorithm, *SASGD*, with explicit gradient aggregation intervals that allow for sparse gradient updates to amortize communication cost. The learners themselves aggregate the learned gradients through collective operations without a parameter server.

Alg. 1 gives a formal description of *SASGD*. In Alg. 1, $T$ is the aggregation interval, $p$ is the number of learners, $id$ is the learner ID ($0 \leq id < p$), and $K$ is the total number of global gradient aggregations. Note there are two different learning rates, $\gamma$ and $\gamma_p$. Learning rate $\gamma$ determines the step size for local updates within an aggregation interval, while $\gamma_p$ is the step size for global aggregation. Each learner accumulates gradients learned within an interval into $g_s$. Global aggregation aggregates $g_s$ from all learners through *allreduce*. The parameter $x$ is initialized by learner 0, and then broadcast to all learners.

**Algorithm 1** *SASGD* ($T$, $p$, $id$, $\gamma$, $\gamma_p$, $K$)

```
$g_s \leftarrow 0$, $i \leftarrow 0$
**if** $id = 0$ **then**
    initialize parameters $x$
**end if**
$x \leftarrow$ *broadcast*$(x, p, id)$
$x' \leftarrow x$
**while** $i < K$ **do**
    $j \leftarrow 0$
    **while** $j < T$ **do**
        compute gradient $g$ from a random minibatch
        $x \leftarrow x - \gamma * g$, $g_s \leftarrow g_s + g$
        $j \leftarrow j + 1$
    **end while**
    $g_s \leftarrow$ *allreduce*$(g_s, p, id)$
    $x \leftarrow x' - \gamma_p g_s$
    $x' \leftarrow x$, $g_s \leftarrow 0$
    $i \leftarrow i + 1$
**end while**
```

Alg. 1 is quite simple, and bears resemblance to existing ASGD and synchronous SGD implementations. One major difference between *SASGD* and ASGD is the explicit constraint on the staleness of the gradients. In *SASGD*, the staleness of the gradients is bound explicitly by $T$, while in most ASGD implementations, in addition to $T$, the staleness is also impacted by the relative processing speed of the learners and the position of the learners in the communication network. The amount of data transported per gradient aggregation is O($m \log p$) in *SASGD* (with tree reduction *allreduce*), where $m$ is the model size. In comparison, the amount of data transported in ASGD is O($mp$). Moreover, on current and emerging systems with many GPUs, the communication in *SASGD* can benefit from the large bandwidth among the GPUs, while the communication in ASGD with parameter servers needs to cross a narrower channel to the host.

Alg. 1 simulates model averaging with $\gamma_p = 1/p$. Model averaging is a heuristic used in some synchronous SGD implementations that computes an average of the parameters from all learners. Some implementations average the parameters at the end of learning once (e.g., [30]), and others average the parameters after each minibatch is processed (e.g., [12]). Neither approaches work in our study. The former results in very poor training and test accuracies, and the latter incurs high communication overhead. Moreover, convergence and convergence rate of model averaging relative to $T$ have not yet been shown.

### A. Convergence

Convergence guarantees can be proven for *SASGD* by bounding the gradient norm average after $S = MTKp$ samples are processed (recall that $M$ is the minibatch size). The following theorem gives the convergence guarantee for *SASGD*

**Theorem 2.** *After $K$ global* allreduce *updates the average gradient norm satisfies the following upper bound*

$$\frac{\sum_{k=1}^{K} \mathbb{E}(\|\nabla f(x_k)\|^2)}{K} \leq \frac{2D_f}{S\gamma_p} + 2L^2\sigma^2\gamma_p\gamma MT + L\sigma^2\gamma_p$$

*when $\gamma$ and $\gamma_p$ satisfies $\gamma_p LMTp + 2L^2M^2T^2\gamma_p\gamma \leq 1$*

Due to limited space, we give a brief outline for the proof of Theorem 2. The key to the proof is to bound the difference in the objective function values between the global updates in terms of accumulated gradients. This bound is used in several prior ASGD convergence proofs (e.g., proof of Theorem 1 in [13]). Let $x_{k+1}$ denote the parameter vector after $k^{th}$ global update. With the Lipschitzian gradient property, we have (see Table III for notations)

$$\begin{aligned} f(x_{k+1}) - f(x_k) &\leq \langle \nabla f(x_k), x_{k+1} - x_k \rangle \\ &\quad + \frac{L}{2} \cdot \|x_{k+1} - x_k\|^2 \end{aligned}$$

Note that $x_{k+1} - x_k$ is same as $-\gamma_p g_s$ where we use $g_s$ that is used to compute $k^{th}$ *allreduce* from Algorithm 1. Thus we have

$$f(x_{k+1}) - f(x_k) \leq -\gamma_p \cdot \langle \nabla f(x_k), g_s \rangle + \frac{L\gamma_p^2}{2} \cdot \|g_s\|^2 \quad (10)$$

Since $g_s$ used for $k^{th}$ *allreduce* in Algorithm 1 is the sum of of all the gradients computed by individual learners between $(k-1)^{th}$ and $k^{th}$ *allreduce*, the above equation expresses the difference in the objective function values between the global *allreduce* updates in terms of accumulated gradients.

When we choose $g_s$ to be comparable to $MTp\nabla f(x_k)$, for finite $\sigma^2$, $\|g_s\|^2$ cannot be much larger than $\|\nabla f(x_k)\|^2$. In expectation $\langle \nabla f(x_k), g_s \rangle$ can be expressed in terms of $\|\nabla f(x_k)\|^2$ and an additive bound in terms of $\sigma^2$. Summing $\mathbb{E}(f(x_{k+1}) - f(x_k))$ after $K$ *allreduce* updates, we have

$$\begin{aligned} -D_f &\leq \mathbb{E}(f(x^*) - f(x_1)) \\ &\leq \sum_{k\in[K]} \mathbb{E}(f(x_{k+1}) - f(x_k)) \\ &\leq -\frac{\gamma_p MTp}{2} \sum_{k=1}^{K} \|\nabla f(x_k)\|^2 \\ &\quad + KL^2M^2T^2p\gamma_p\gamma^2\sigma^2 + \frac{KLMTp\gamma_p^2\sigma^2}{2} \end{aligned}$$

Rearranging the terms in the above inequality, we get the desired bound in Theorem 2.

The following corollary establishes the asymptotic convergence guarantee of *SASGD*

**Corollary 3.** *If $\gamma = \gamma_p = \sqrt{\frac{2D_f}{S\sigma^2}}$ and $K \geq \frac{4MLD_f}{\sigma^2} \cdot \frac{(max\{p,T\}+1)^2}{pT}$, then after $K$ global updates, we have*

$$\frac{\sum_{k=1}^{K} \mathbb{E}(\|\nabla f(x_k)\|^2)}{K} \leq 4 \cdot \sqrt{\frac{D_f L\sigma^2}{S}}$$

In other words, the asymptotic convergence guarantee of $O\left(1/\sqrt{S}\right)$ can also be extended to *SASGD*. Although asymptotic convergence does not change regardless of the global update frequency determined by $T$, we must note that the number of global updates $K$ needed in order to achieve the asymptotic convergence rate can substantially increase with the increase in $T$ (see the bound on $K$ in Corollary 3).

### B. Sample complexity relative to $T$

In practice, the number of samples processed in an application oftentimes does not reach the asymptotic convergence regime, that is, $K < \frac{4MLD_f}{\sigma^2} \cdot \frac{(max\{p,T\}+1)^2}{pT}$. In this case, the following theorem says that increasing $T$ always leads to slower convergence in terms of epochs (or the number of processed samples).

**Theorem 4.** *For* SASGD *with a constant number of learners $p$ and constant minibatch size $M$, given $\gamma_p = \gamma$, the number of samples needs to be processed in order to achieve the same convergence guarantee increases as the interval $T$ between global updates increases.*

*Proof outline.* We show that keeping the same number of samples processed (i.e., the same $S$) the upper bound on convergence guarantee from Theorem 2 becomes worse as $T$ increases. Thus, it implies that in order to reach the same convergence guarantee, higher number of samples need to be processed with larger $T$. $S$ is kept constant by adjusting $K$.

Let $\gamma_p = \gamma$. The range of $\gamma$ permissible by the constraint in Theorem 2 becomes smaller as $T$ increases. The rest of the proof combines the observations that the minimum attained by the convergence guarantee in Theorem 2 must become worse if the range of $\gamma$ decreases and $T$ increases. □

For a given $p$, increasing $T$ reduces the epoch time but increases the number of epochs needed to reach the target convergence guarantee. Thus there is an optimal $T$ for a specific application in terms of the wall-clock time needed to reach convergence. For a given $T$, increasing the number of learners will also increase the sample complexity to reach convergence. The analysis for *SASGD* convergence relative to SGD is similar to the analysis for ASGD in section II-B. The advantage of *SASGD* over ASGD implementations from the convergence perspective is that the staleness in gradient updates is explicitly bounded. The impact on performance is discussed in section IV-C.

## IV. Performance

We evaluate the performance of *SASGD* with the *NLC-F* and *CIFAR-10* data sets. We study the impact of $T$ on epoch time and convergence. We also present a detailed comparison of the convergence behavior between *SASGD* and *Downpour* and *EAMSGD*.

### A. *Impact of $T$ on epoch time*

After every $T$ minibatches, a global reduction of the gradients occurs in *SASGD* among the learners. Communication time is amortized among $T$ minibatches. When $T = 1$, *SASGD* becomes the traditional synchronous SGD.

We experiment with $T = 1$ and $T = 50$ for *NLC-F* and *CIFAR-10*. The results with 1, 2, 4, and 8 learners for *CIFAR-10* and *NLC-F* are shown in Fig. 4 and Fig. 5, respectively. Increasing $T$ from 1 to 50 reduces the epoch time for both data sets. The reduction is more significant for *NLC-F* than *CIFAR-10*. With 8 learners, *SASGD* with $T = 50$ is 1.3 times faster than with $T = 1$ for *CIFAR-10*, and is 9.7 times faster for *NLC-F*. In both figures, the horizontal line shows the sequential time. The speedups with 8 learners are 4.45 and 5.35 for *CIFAR-10* and *NLC-F*, respectively.

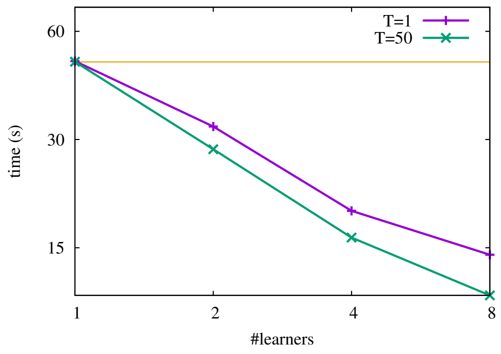


Fig. 4: Impact of $T$ on epoch time for *CIFAR-10*. In log-log plot. The horizontal line shows the sequential time

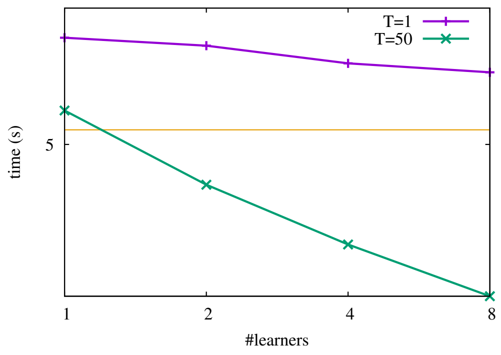


Fig. 5: Impact of $T$ on epoch time for *NLC-F*. In log-log plot. The horizontal line shows the sequential time

Both *Downpour* and *EAMSGD* also have a gradient update interval $T$. As the amount of data transported grows linearly with the number of learners, performance can take a hit when the number of learners increases. Fig. 6 shows the epoch time for *Downpour*, *EAMSGD*, and *SASGD* with *CIFAR-10* and *NLC-F*. With $T = 1$, gradient update is frequent, and communication takes a significant portion of the epoch time. *SASGD* is much faster than *Downpour* and *EAMSGD* due to its lower communication complexity. With $T = 50$, communication time in all three approaches is amortized for multiple minibatches, and computation time dominates. All three approaches have similar epoch times.

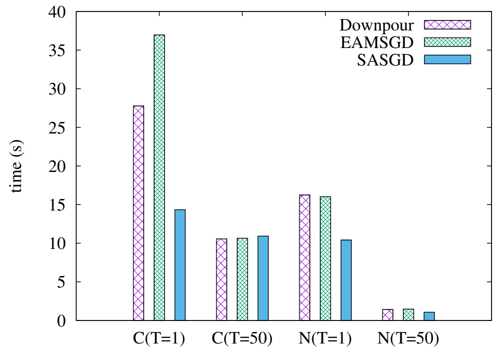


Fig. 6: Epoch time with $T = 50$ for *Downpour*, *EAMSGD*, and *SASGD* with 8 learners. "C" is for *CIFAR-10*, and "N" is for *NLC-F*

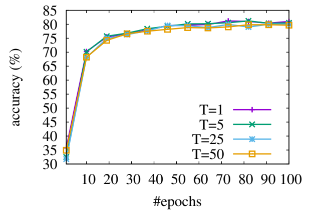
### B. *Impact of $T$ on convergence*

As analyzed in Section III-B, increasing $T$ is likely to increase the number of samples needed to reach convergence. For the same amount of samples processed, larger $T$ will likely result in lower accuracy. We experiment with $T = 1$, 5, 25, and 50 for $p = 2$, 4, 8, 16 learners. For $p = 16$, we run two learners per GPU. Since we are concerned with sample complexity and not wall-clock time, we can run multiple learners per GPU. In the rest of the paper, whenever 16 learners are used, we run two of them per GPU. The test accuracies with *CIFAR-10* are shown in Fig. 7(a), 7(b), 7(c), and 7(d).

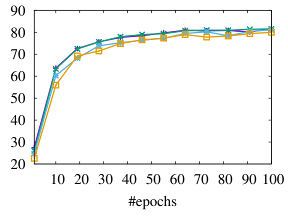
In each figure, as $T$ increases, the test accuracy achieved at the end of 100 epochs degrades slightly. Thus for a given number of learners, more epochs are needed to reach a target accuracy with larger $T$. This observation agrees with our analysis in Section III-B.

The degradation in accuracy is negligible when $p$ is small. For example, with $p = 2$, after 100 epochs, the gap in test accuracy for $T = 50$ versus $T = 1$ is 1.32%. As $p$ increases, the gap becomes larger. With $p = 16$, after 100 epochs, the gap is 3.21%.

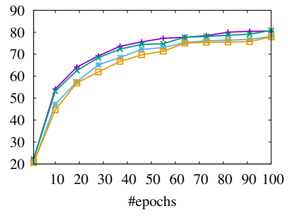
The test accuracies with *NLC-F* are shown in Fig. 8(a), 8(b), 8(c), and 8(d). In comparison to *CIFAR-10*, for a given $p$, the degradation in accuracy after 200 epochs when $T$ increases is not as pronounced. The degradation is



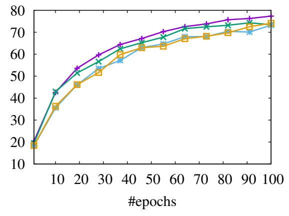
(a) $p = 2$



(b) $p = 4$

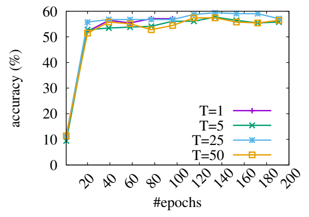


(c) $p = 8$

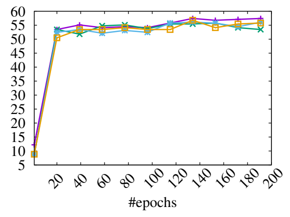


(d) $p = 16$

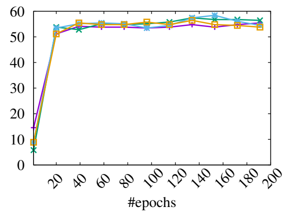
Fig. 7: Test accuracy with various $T$ values for *CIFAR-10*. For readability, the accuracies for every 10 epochs are shown



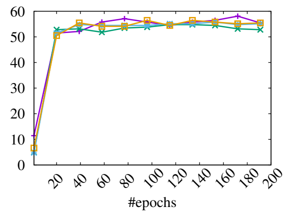
(a) $p = 2$



(b) $p = 4$



(c) $p = 8$



(d) $p = 16$

Fig. 8: Test accuracy with various $T$ values for *NLC-F*. For readability, the accuracies for every 20 epochs are shown

the most obvious for $p = 8$ in Fig. 8(c). For $p = 16$, the best accuracy is actually achieved with $T = 50$.

Similar impact of $T$ is observed for training accuracies.

## C. *Comparison with ASGD*

*SASGD* is much faster than *Downpour* and *EAMSGD* in terms of epoch time when $T$ is small. $T$ needs to be large for all three approaches to amortize the communication overhead for a large number of learners. We show *SASGD* has better convergence behavior for large $T$. That is, for the same amount of data samples processed, *SASGD* achieves better accuracy than *Downpour* and *EAMSGD*.

In our experiment with *CIFAR-10*, we run each training algorithm for 100 epochs. That is, all learners collectively make 100 passes of all input data. We use $T = 50$. Recall in section IV-A, we have shown that with $T = 50$, all three approaches have similar epoch time.

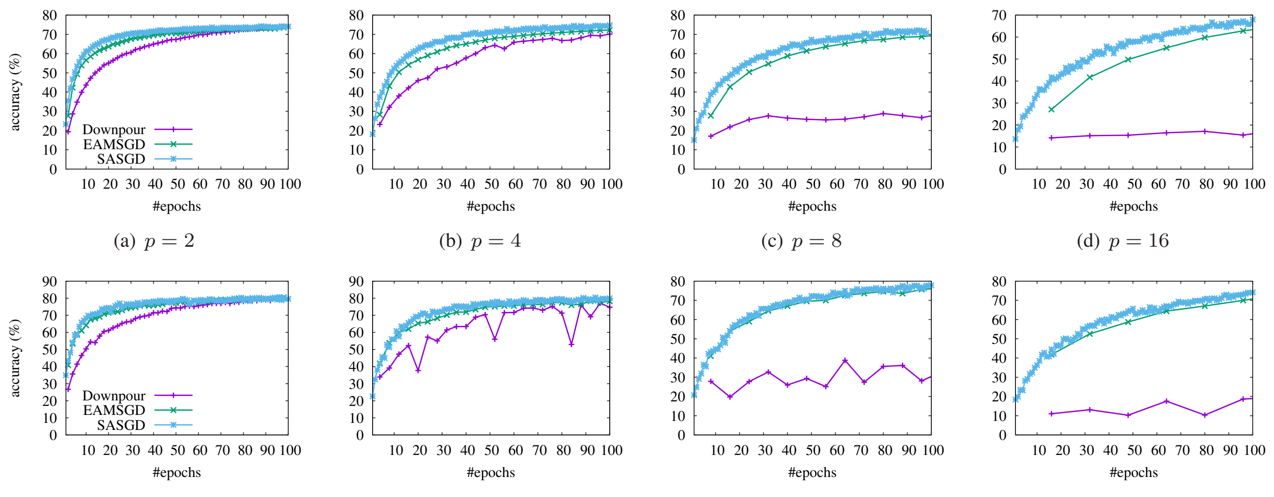
The training accuracies for *Downpour*, *EAMSGD*, and *SASGD* are shown in Fig. 9(a), 9(b), 9(c), and 9(d) for $p = 2$, 4, 8, and 16, respectively. Due to the synchronous nature of *SASGD*, the accuracy numbers for all learners after each epoch are similar. Both *EAMSGD* and *Downpour* are asynchronous, and before they terminate, the accuracy numbers for different learners see a wider range of fluctuation. For consistency, we collect accuracy numbers from one learner after it has made a complete pass of the input data. Thus from the perspective of total number of samples processed by all learners, *Downpour* and *EAMSGD* report accuracy numbers after every $p$ epochs, and *SASGD* report accuracy numbers after each epoch. In the plots *Downpour* and *EAMSGD* have $1/p$ as many data points as *SASGD*.

Comparing the plots in these figures, it is clear that *Downpour* performs poorly in terms of achieved accuracy with $p = 8, 16$. This behavior is also reported in the *EAMSGD* study [29]. *EAMSGD* performs much better than *Downpour*, and *SASGD* in turn performs consistently better than *EAMSGD*. As $p$ increases, the gap in accuracy between *SASGD* and *EAMSGD* increases, suggesting that *SASGD* tolerates stale updates better than *EAMSGD*. With $p = 16$, the accuracy gap between *SASGD* and *EAMSGD* after 100 epochs is 2.95%. This gap significant for *CIFAR-10*.

In Fig. 9(a), all three approaches converge after 100 epochs with 2 learners. Before convergence (i.e., before 70 epochs), both *SASGD* and *EAMSGD* improve training accuracy much faster than *Downpour*, and *SASGD* performs slightly better than *EAMSGD*. In Fig. 9(b), 9(c), and 9(d), none of the three approaches fully converge after 100 epochs. This demonstrates the parallelization overhead for convergence. *SASGD* is still the best performing algorithm, while *Downpour* starts to show some erratic behavior with $p = 4$ and degenerates to almost random guess for $p = 8$, 16.

Fig. 9(e), 9(f), 9(g), and 9(h) show test accuracies for *Downpour*, *EAMSGD*, and *SASGD* for $p = 2$, 4, 8, and 16, respectively. The test accuracy curves track the training accuracy curves in Fig. 9(a), 9(b), 9(c), and 9(d) for $p = 2$, 4, 8, and 16, respectively. *Downpour* shows the worst performance. With 4 learners *Downpour* starts to behave erratically, and with 16 learners, the test accuracy degrades to random guess. *SASGD* consistently ranks as the top performer among the three approaches. The gap between *SASGD* and *EAMSGD* increases as the number of learners increases, again suggesting *SASGD* a better choice for more learners.

We also compare the performance of the three approaches for *NLC-F* with $T = 50$. With each approach the learners collectively make 200 passes of the input data. Fig-



(a) $p = 2$ (b) $p = 4$ (c) $p = 8$ (d) $p = 16$

(e) $p = 2$ (f) $p = 4$ (g) $p = 8$ (h) $p = 16$

Fig. 9: Training and test accuracies for *CIFAR-10*. The top row is for training, and the bottom row is for test

ures 10(a), 10(b),10(c), and 10(d) show training accuracies for *Downpour*, *EAMSGD*, and *SASGD* for $p = 2$, 4, 8, and 16, respectively.

*SASGD* consistently reaches to close to 100% training accuracy at the end of 200 epochs. In comparison, *Downpour* and *EAMSGD* also reach high training accuracies with 2 and 4 learners, but with 8 learners, the training accuracy for *Downpour* and *EAMSGD* starts to degrade. With 16 learners, neither *Downpour* nor *EAMSGD* achieves accuracy better than the random guess.

Figures 10(e),10(f),10(g), and 10(h) show test accuracies for *Downpour*, *EAMSGD*, and *SASGD*. The test accuracy curves largely track the training accuracy curves except for $p = 8$. The maximum test accuracies achieved by the three approaches are around 60%. This is also the accuracy achieved by the sequential implementation after 200 epochs. For $p = 2$, and 4, all three approaches have similar accuracy curves. For $p = 8$, and 16, *SASGD* consistently achieves much better accuracies than *EAMSGD* and *Downpour*. With 8 learners, the accuracy drops to between 30% and 40% for *Downpour* and *EAMSGD*, while the accuracy for *SASGD* remains close to 60%. With 16 learners, both *Downpour* and *EAMSGD* exhibit erratic behavior and achieve accuracies not much better than random guess, while *SASGD* still achieves close to 60% accuracy.

## V. Conclusion

Parallelization impacts not only the epoch time but also the convergence rate of stochastic gradient descent. Efficient parallelization is critical in reducing the execution time while maintaining the accuracy for the trained model. Although in theory ASGD converges with asymptotic linear speedup over SGD, we show in practice ASGD faces challenges of significant communication overhead and high sample complexity to reach convergence. Stale gradients resulted from asynchronous updates increase the stochasticity in optimization, and can substantially degrade training and test accuracies with even a moderate number of learners. On current and emerging computer platforms, communication bandwidth between the (sharded) parameter server on CPUs and learners on GPUs is a limiting factor for scaling when the neural network models are large and/or gradient aggregation is frequent.

We propose a bulk-synchronous, distributed SGD algorithm, *SASGD*, that allows sparse gradient aggregation. Instead of reducing communication overhead through asynchronous updates, we adopt an explicit gradient aggregation interval $T$ that amortizes the communication cost. The aggregation is done through collective *allreduce* that does not rely on a parameter server. In our experiments, from the perspective of epoch time, *SASGD* achieves significant speedups over SGD with $T = 50$ while maintaining good convergence; *SASGD* is much faster than the ASGD implementations when $T$ is small due to its low communication complexity.

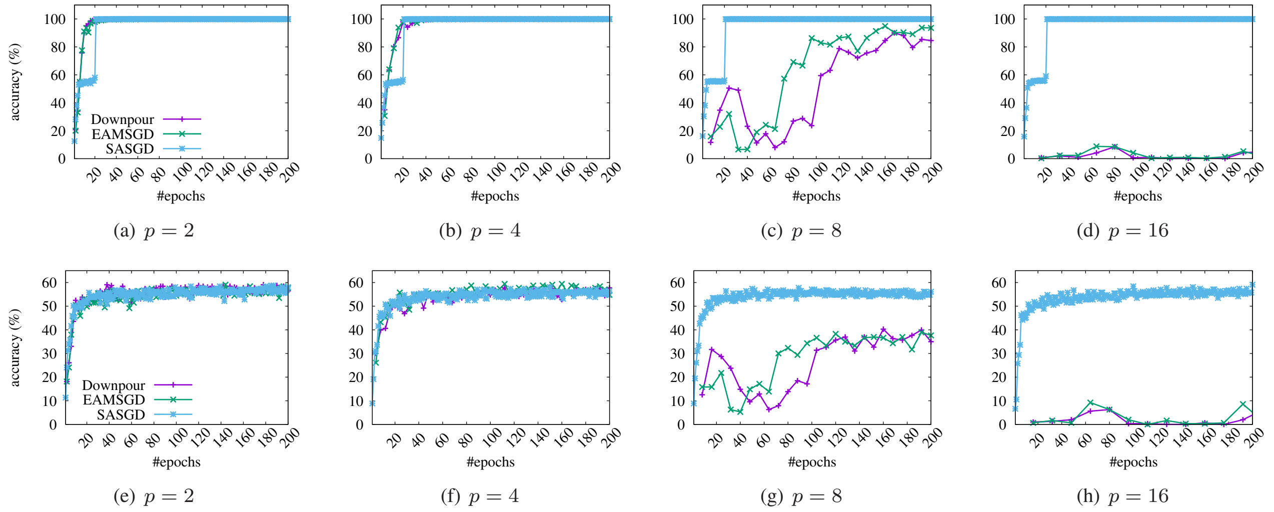
We also compare the convergence performance of *SASGD* with *Downpour* and *EAMSGD*. With a small number of learners (e.g., 1 or 2 learners), all three approaches have similar convergence behavior for different $T$. When the number of learners reach 8, 16, and beyond, the stochasticity from asynchronous updates in *Downpour* and *EAMSGD* substantially degrades accuracy, while *SASGD* still shows stable convergence behavior. As the number of GPUs in future systems is likely to increase, we expect *SASGD* perform better than ASGD implementations for machine learning applications.

## References

[1] CIFAR10 model. https://github.com/eladhoffer/ConvNet-torch/blob/master/Models/Model.lua, accessed January 11, 2016.

[2] O. Abdel-Hamid, A-R Mohamed, H. Jiang, and *et al.* Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing*, 22(10):1533–1545, 2014.

[3] L. Bottou. Online learning and stochastic approximations, 1998.



(a) $p = 2$ (b) $p = 4$ (c) $p = 8$ (d) $p = 16$

(e) $p = 2$ (f) $p = 4$ (g) $p = 8$ (h) $p = 16$

Fig. 10: Training and test accuracies for *NLC-F*. The top row is for training, and the bottom row is for test

[4] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 571–582, 2014.

[5] J. Dean, G. Corrado, R. Monga, and et al. Large scale distributed deep networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1223–1231. Curran Associates, Inc., 2012.

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

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

[8] *NVIDIA GPUDirect*, https://developer.nvidia.com/gpudirect.

[9] A. Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, 2009.

[10] A. Krizhevsky, I. Sutskever, and G.E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[11] M. Li, D. G. Andersen, J. W. Park, and et al. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 583–598, Broomfield, CO, October 2014. USENIX Association.

[12] M. Li, T. Zhang, Y. Chen, and A. J. Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 661–670, New York, NY, USA, 2014. ACM.

[13] X. Lian, Y. Huang, Y. Li, and J. Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pages 2737–2745, 2015.

[14] Q. Lin, Z. Lu, and L. Xiao. An accelerated proximal coordinate gradient method. In *Advances in Neural Information Processing Systems*, pages 3059–3067, 2014.

[15] *mpiT – MPI for Torch*, https://github.com/sixin-zh/mpiT.

[16] *Multi-process service*, https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf.

[17] 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.

[18] *One Stop System High Density Compute Accelerator*, http://www.onestopsystems.com/blog-post/one-stop-systems-shows-its-16-gpu-monster-machine-gtc-2015.

[19] T. Paine, H. Jin, J. Yang, and et al. Gpu asynchronous stochastic gradient descent to speed up neural network training. *arXiv preprint arXiv:1312.6186*, 2013.

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

[21] H. Robbins and D. Siegmund. A convergence theorem for non negative almost supermartingales and some applications. In *Herbert Robbins Selected Papers*, pages 111–135. Springer, 1985.

[22] O. Shamir and T. Zhang. Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes. In *ICML (1)*, pages 71–79, 2013.

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

[24] C. Szegedy, W. Liu, Y. Jia, and *et al.* Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[25] *Torch – A scientific computing framework for Luajit*, http://torch.ch.

[26] D.R. Wilson and T.R. Martinez. The general inefficiency of batch training for gradient descent learning. *Neural Network*, 16(10):1429–1451, December 2003.

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

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

[29] S. Zhang, A. Choromanska, and Y. LeCun. Deep learning with elastic averaging SGD. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 685–693, 2015.

[30] M. Zinkevich, M. Weimer, L. Li, and A.J. Smola. Parallelized stochastic gradient descent. In *Advances in neural information processing systems*, pages 2595–2603, 2010.