DEEP LEARNING WITH ELASTIC AVERAGING SGD

Sixin Zhang
Courant Institute of Mathematical Sciences
New York, NY, USA
zsx@cims.nyu.edu

Anna Choromanska
Courant Institute of Mathematical Sciences
New York, NY, USA
achoroma@cims.nyu.edu

Yann LeCun
Courant Institute of Mathematical Sciences
New York, NY, USA
yann@cims.nyu.edu

ABSTRACT

We study the problem of stochastic optimization for deep learning in the parallel computing environment under communication constraints. A new algorithm is proposed in this setting where the communication and coordination of work among concurrent processes (local workers), is based on an elastic force which links the parameter vectors they compute with a center variable stored by the parameter server (master). The algorithm enables the local workers to perform more exploration, i.e. the algorithm allows the local variables to fluctuate further from the center variable by reducing the amount of communication between local workers and the master. We empirically demonstrate that in the deep learning setting, due to the existence of many local optima, allowing more exploration can lead to the improved performance. We propose synchronous and asynchronous variants of the new algorithm. We provide the theoretical analysis of the synchronous variant in the quadratic case and prove it achieves the highest possible asymptotic rate of convergence for the center variable. We additionally propose the momentum-based version of the algorithm that can be applied in both synchronous and asynchronous settings. An asynchronous variant of the algorithm is applied to train convolutional neural networks for image classification on the *CIFAR* and *ImageNet* datasets. Experiments demonstrate that the new algorithm accelerates the training of deep architectures compared to *DOWNPOUR* and other common baseline approaches and furthermore is very communication efficient.

1 INTRODUCTION

One of the most challenging problems in large-scale machine learning is how to parallelize the training of large models that use a form of stochastic gradient descent (*SGD*) Bottou (1998). There have been attempts to parallelize SGD-based training for large-scale deep learning models on large number of CPUs, including the Google's Distbelief system Dean et al. (2012). But practical image recognition systems consist of large-scale convolutional neural networks trained on one GPU card (or a small number of GPU cards) sitting in a single computer Krizhevsky et al. (2012); Sermanet et al. (2013). The main challenge is to devise parallel *SGD* algorithms to train large-scale deep learning models (particularly convolutional networks) that yield a significant speedup when run on multiple GPU cards (modern machines can host four, or even eight, high-end GPU cards). This optimization problem remains very open in deep learning and to the best of our knowledge there exists only one successful approach in the literature Dean et al. (2012).

In this paper we introduce the *Elastic Averaging SGD* method (*EASGD*) and its variants. *EASGD* is motivated by quadratic penalty method Nocedal & Wright (2006), but is re-interpreted as a parallelized extension of the averaging *SGD* algorithm Polyak & Juditsky (1992). The basic idea is to let each worker, i.e. a GPU card, maintain its own local parameter vector, and the communication and coordination of work among the local workers is based on an elastic force which links the parameter vectors they compute with a center variable stored by the master. The center variable is updated as a moving average where the average is taken in time and also in space over the parameter vectors computed by local workers. The main contribution of this paper is a new algorithm that provides fast convergent minimization while outperforming *DOWNPOUR* method Dean et al. (2012) and other baseline approaches in practice. Simultaneously it reduces the communication overhead between the master and the local workers while at the same time it maintains high-quality performance measured by the test error. The new algorithm applies to deep learning settings such as parallelized training of convolutional neural networks.

The article is organized as follows. Section 2 explains the problem setting more in details. Section 3, 4 and 5 presentes the *EASGD* algorithm and its variants (asynchronous variant in Section 4 and momentum-based variant in Section 5). Section 6 shows the theoretical analysis of the convergence of the synchronous variant of *EASGD*. Section 7 provides experimental results and Section 8 concludes. The Supplement contains proofs and additional material.

2 PROBLEM SETTING

Consider minimizing a function $F(x)$ in a parallel computing environment Bertsekas & Tsitsiklis (1989) with $p \in \mathbb{N}$ workers (each worker uses a single GPU processor) and a master. In this paper we focus on the stochastic optimization problem of the following form

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

where x is the model parameter to be estimated and ξ is a random disturbance that follows the probability distribution $\mathbb{P}$ on the sample space Ω such that $F(x) = \int_\Omega f(x, \xi)\mathbb{P}(d\xi)$. The optimization problem in Equation 1 can be reformulated as follows

$$min_{x^i, \tilde{x}} \sum_{i=1}^{p} \mathbb{E}[f(x^i, \xi^i)] + \frac{\rho}{2}\|x^i - \tilde{x}\|^2 \qquad \text{s.t.} \;\; x^i = \tilde{x}, \;\; i = 1, 2, \ldots, p, \tag{2}$$

where each ξ^i follows the same distribution $\mathbb{P}$. In the paper we refer to x^i's as local variables and we refer to $\tilde{x}$ as a center variable. The equivalence of both problems comes from the fact that for any feasible solution the term added to the objective is zero. The problem in Equation 2 appears in the literature as a *global variable consensus optimization* problem Boyd et al. (2011); Bertsekas & Tsitsiklis (1989). The quadratic penalty term in Equation 2 is expected to ensure that local workers will not fall into different local optima that are far away from the center variable. Note that we do not use Lagrangian multipliers in the objective function[1] and we focus on how fast the center variable converges to the local optimum. As opposed to other methods used for solving the consensus problem, which split the data among the workers, we assume each worker has access to the entire dataset. This paper focuses on the problem of reducing the parameter communication overhead Shamir (2014) in the stochastic deep learning setting Dean et al. (2012); Yadan et al. (2013); Paine et al. (2013). The problem of data communication when the data is distributed among the workers Bekkerman et al. (2011); Bertsekas & Tsitsiklis (1989) is a more general problem and is not addressed in this work (we explain the details of sampling the entire dataset in parallel in Section 7). We however emphasize that to the best of our knowledge the problem of stochastic optimization in the parallel deep learning setting under the communication constraints was never addressed in the literature before (and thus as well a more general problem when the data is distributed).

3 EASGD UPDATE RULE

Note that we focus on a stochastic setting as shown in Equation 2. The standard methods for solving the consensus problem such as *ADMM* Boyd et al. (2011); Zhang & Kwok (2014); Wei & Ozdaglar

[1] It will be shown that despite not using Lagrangian multipliers the algorithm we obtain for minimizing this objective achieves asymptotically optimal convergence rate of the center variable in a quadratic case. It is furthermore unclear if this optimal rate could be achieved when using Lagrangians.

(2012) therefore no longer apply. Let us introduce the following notation

$$\Phi(x^i, \tilde{x}) := \mathbb{E}[f(x^i, \xi^i)] + \frac{\rho}{2}\|x^i - \tilde{x}\|^2.$$

To obtain the *EASGD* algorithm we *linearize* $\Phi(x^i, \tilde{x})$ as follows

$$L_t^{\rho}(x^i, \tilde{x}) = \left[\langle g_t^i(x_t^i) + \rho(x_t^i - \tilde{x}_t), x^i - x_t^i\rangle + \frac{1}{2\eta}\|x^i - x_t^i\|^2\right] + \left[\rho\langle \tilde{x}_t - x_t^i, \tilde{x} - \tilde{x}_t\rangle + \frac{1}{2\eta}\|\tilde{x} - \tilde{x}_t\|^2\right], \quad (3)$$

where the first component of the summand captured in the first squared brackets in Equation 3 comes from linearizing $\Phi(x^i, \tilde{x})$ with respect to x^i and the second one comes from linearizing $\Phi(x^i, \tilde{x})$ with respect to $\tilde{x}$. t is the iteration index, $g_t^i(x_t^i)$ denotes the stochastic gradient of f with respect to x^i evaluated at iteration t, x_t^i and $\tilde{x}_t$ denote respectively the value of variables x^i and $\tilde{x}$ at iteration t, and η is the learning rate. Linearization directly leads to the update rule used by the *EASGD* algorithm of the following form

$$x_{t+1}^i = x_t^i - \eta(g_t^i(x_t^i) + \rho(x_t^i - \tilde{x}_t)), \quad (4)$$

$$\tilde{x}_{t+1} = \tilde{x}_t + \eta\sum_{i=1}^{p}\rho(x_t^i - \tilde{x}_t). \quad (5)$$

The above update[2] can be easily implemented in the parallel computing environment with one master and p workers where the updates performed by the workers, each one having the form captured in Equation 4, can be carried out independently, in parallel, and the master performs the update captured in Equation 5. Note that as opposed to stochastic *ADMM* Azadi & Sra (2014); Ouyang et al. (2013) we also linearize the term $\frac{\rho}{2}\|x^i - \tilde{x}\|^2$. The linearization of the penalty term leads to the update rule for the center $\tilde{x}$ of the form of moving average where the average is taken over both space and time. To see this note that if $\alpha = \eta\rho$ (in the paper we refer to α as a moving rate) then

$$\tilde{x}_{t+1} = (1 - p\alpha)\tilde{x}_t + p\alpha\left(\frac{1}{p}\sum_{i=1}^{p} x_t^i\right). \quad (6)$$

The magnitude of ρ represents the amount of exploration we allow in the model. In particular, small ρ allows for more exploration as it allows x^i's to fluctuate further from the center $\tilde{x}$. In the non-convex setting, e.g. deep learning setting, due to the existence of local optima we want to allow for more exploration (small ρ). In particular we demonstrate in the experimental section that *EASGD* and its momentum-based variant allow to increase the amount of exploration by reducing the communication between local workers and the master. Note that standard approaches in the literature, like stochastic *ADMM* Azadi & Sra (2014), use the original squared penalty term (without linearizing it) and typically explore large values of ρ to heavily push local x^i's to the center $\tilde{x}$ preventing exploration. This is because these approaches care how fast the local variables converge to the local optimum. In our setting we instead focus on how fast the center variable converges. Thus our goal is the same as for the *DOWNPOUR* method Dean et al. (2012). To summarize, the distinctive feature of *EASGD* is that it allows the local workers for significantly more exploration as opposed to *ADMM*-like methods, *DOWNPOUR* and many others Borkar (1998); Nedić et al. (2001); Langford et al. (2009); Agarwal & Duchi (2011); Recht et al. (2011); Zinkevich et al. (2010).

Finally note that we use the same learning rate η when linearizing $\Phi(x^i, \tilde{x})$ with respect to x^i and $\tilde{x}$. This leads to a simple algorithm where there exists an elastic force equal to $\alpha(x_t^i - \tilde{x}_t)$ between the update of each x_t^i and $\tilde{x}$. Note that using different η's in Equation 3 may lead to more complicated update rules with preconditioning of the gradients of $\Phi(x^i, \tilde{x})$. This is a more complicated approach that we do not analyze in this paper. We emphasize however that our algorithm can still achieve the asymptotic optimal convergence rate of the center variable as will be shown in Section 6.

4 ASYNCHRONOUS EASGD

In Section 3 we showed the synchronous version of the *EASGD* algorithm, where the workers update the local variables in parallel such that the i^{th} worker reads the current value of the center variable

[2]In the literature the update for the master involves x_{t+1}^i instead of x_t^i. However we obtained better experimental performance in the asynchronous case when having x_t^i in the update rule for the master.

and use it to update local variable x^i using Equation 4. All workers share the same global clock. The master has to wait for the x^i updates from all p workers before being allowed to update the value of the center variable $\tilde{x}$ according to Equation 5.

In this section we instead show the asynchronous variant of the *EASGD* algorithm. The local workers are still responsible for updating the local variables x^i's, whereas the master is updating the center variable $\tilde{x}$. Since the algorithm is fully asynchronous, every worker has its own clock t^i, which starts from 0 and is incremented by 1 after each stochastic gradient update of x^i as shown in Algorithm 1. The master performs an update whenever the local workers finished τ steps of their gradient updates, where we refer to τ as the *communication period*. As can be seen in Algorithm 1, whenever τ divides the local clock of the i^{th} worker, the i^{th} worker communicates with the master and requests the current value of the center variable $\tilde{x}$. The worker then waits until the master sends back the requested parameter value, and computes the elastic difference $\alpha(x - \tilde{x})$ (this entire procedure is captured in step a) in Algorithm 1). The elastic difference is then sent back to the master (step b) in Algorithm 1) who then updates $\tilde{x}$.

The communication period τ controls the frequency of the communication between every local worker and the master. In this paper we are interested in reducing the communication overhead in the parallel computing environment where the parameter vector is very large. As will be seen in the experimental section, the *EASGD* algorithm and its momentum-based variant achieve good performance with large τ (less frequent communication).

Algorithm 1: Asynchronous EASGD: Processing by worker i and the master

Input: learning rate η, moving rate α, communication period $\tau \in \mathbb{N}$
Initialize: $\tilde{x}$ is initialized randomly, $x^i = \tilde{x}$, $t^i = 0$

Repeat
  $x \leftarrow x^i$
  if (τ divides t^i) **then**
    a) $x^i \leftarrow x^i - \alpha(x - \tilde{x})$
    b) $\tilde{x} \leftarrow \tilde{x} + \alpha(x - \tilde{x})$
  end
  $x^i \leftarrow x^i - \eta g^i_{t^i}(x)$
  $t^i \leftarrow t^i + 1$
Until forever

Algorithm 2: Asynchronous EAMSGD: Processing by worker i and the master

Input: learning rate η, moving rate α, communication period $\tau \in \mathbb{N}$, momentum term δ
Initialize: $\tilde{x}$ is initialized randomly, $x^i = \tilde{x}$, $v^i = 0$, $t^i = 0$

Repeat
  $x \leftarrow x^i$
  if (τ divides t^i) **then**
    a) $x^i \leftarrow x^i - \alpha(x - \tilde{x})$
    b) $\tilde{x} \leftarrow \tilde{x} + \alpha(x - \tilde{x})$
  end
  $v^i \leftarrow \delta v^i - \eta g^i_{t^i}(x + \delta v^i)$
  $x^i \leftarrow x^i + v^i$
  $t^i \leftarrow t^i + 1$
Until forever

5 MOMENTUM EASGD

The momentum EASGD (*EAMSGD*) is a variant of our Algorithm 1 and is captured in Algorithm 2. It is based on the Nesterov's momentum scheme Nesterov (2005); Lan (2012); Sutskever et al. (2013), where the update of the local worker of the form captured in Equation 4 is replaced by the following update

$$\begin{aligned} v^i_{t+1} &= \delta v^i_t - \eta g^i_t(x^i_t + \delta v^i_t) \\ x^i_{t+1} &= x^i_t + v^i_{t+1} - \eta\rho(x^i_t - \tilde{x}_t), \end{aligned} \tag{7}$$

where δ is the momentum term. Note that when $\delta = 0$ we recover the original *EASGD* algorithm.

6 CONVERGENCE ANALYSIS OF SYNCHRONOUS EASGD

In this section we provide the convergence analysis of the synchronous *EASGD* algorithm for the quadratic case, i.e. where the function f is assumed to have a quadratic form. The style of our analysis resembles the convergence analysis of ASGD Polyak & Juditsky (1992) and is focused on the convergence of the center variable to the local optimum. Since, as opposed to other popular approaches focusing on distributed optimization (e.g. Azadi & Sra (2014); Nedić et al. (2001)), we

allow the workers to fluctuate further from the center performing the exploration of the space (as we do not split the data among the workers), the form of our theoretical results is fundamentally different from the existing guarantees for these other approaches.

Assume each of the p local workers $x_t^i \in \mathbb{R}^n$ observes a noisy gradient at (discrete) time $t \geq 0$ of the linear form given in Equation 8.

$$g_t^i(x_t^i) = Ax_t^i - b - \xi_t^i, \quad i \in \{1, \dots, p\}, \tag{8}$$

where the matrix A is positive-definite (each eigenvalue is strictly positive) and $\{\xi_t^i\}$'s are i.i.d. random variables, with zero mean and positive-definite covariance Σ. Let x^* denote the optimum solution, where $x^* = A^{-1}b \in \mathbb{R}^n$. In this section we analyze the behavior of the mean squared error (MSE) of the center $\tilde{x}_t$, where this error is denoted as $\mathbb{E}[\|\tilde{x}_t - x^*\|^2]$, as a function of t, p, η, α and β, where $\beta = p\alpha$. Note that the MSE error can be decomposed as (squared) bias and variance[3]: $\mathbb{E}[\|\tilde{x}_t - x^*\|^2] = \|\mathbb{E}[\tilde{x}_t - x^*]\|^2 + \mathbb{V}[\tilde{x}_t - x^*]$. We first provide Lemma 6.1 and 6.2 for one-dimensional case ($n = 1$), where $A = h > 0$ and $\Sigma = \sigma^2 > 0$. Lemma 6.3 generalizes our results to multidimensional case. All the proofs are deferred to the Supplementary material.

Lemma 6.1. *Let $\tilde{x}_0$, and $\{x_0^i\}_{i=1,\dots,p}$ be arbitrary constants. Then*

$$\mathbb{E}[\tilde{x}_t - x^*] = \gamma^t(\tilde{x}_0 - x^*) + \frac{\gamma^t - \phi^t}{\gamma - \phi}\alpha u_0, \tag{9}$$

$$\mathbb{V}[\tilde{x}_t - x^*] = \frac{p^2\alpha^2\eta^2}{(\gamma - \phi)^2}\left(\frac{\gamma^2 - \gamma^{2t}}{1 - \gamma^2} + \frac{\phi^2 - \phi^{2t}}{1 - \phi^2} - 2\frac{\gamma\phi - (\gamma\phi)^t}{1 - \gamma\phi}\right)\frac{\sigma^2}{p}, \tag{10}$$

where $u_0 = \sum_{i=1}^p (x_0^i - x^* - \frac{\alpha}{1-p\alpha-\phi}(\tilde{x}_0 - x^*))$, $a = \eta h + (p+1)\alpha$, $c^2 = \eta h p \alpha$, $\gamma = 1 - \frac{a - \sqrt{a^2 - 4c^2}}{2}$, *and* $\phi = 1 - \frac{a + \sqrt{a^2 - 4c^2}}{2}$.

It follows from Equations 9 and 10 that for the MSE to converge the following condition has to hold

$$-1 < \phi < \gamma < 1. \tag{11}$$

Note that ϕ and γ are the two zero-roots of the polynomial in λ of the form: $\lambda^2 - (2 - a)\lambda + (1 - a + c^2)$. Recall that ϕ and λ are the functions of η and β. Thus condition in Equation 11 implies that

- $\gamma < 1$ iff $c^2 > 0$ (i.e. $\eta > 0$ and $\beta > 0$).
- $\phi > -1$ iff $(2 - \eta h)(2 - \beta) > 2\beta/p$ and $(2 - \eta h) + (2 - \beta) > \beta/p$.
- $\phi = \gamma$ iff $a^2 = 4c^2$ (i.e. $\eta h = \beta = 0$).

In particular, the necessary condition for Equation 11 to hold is that $\eta \in (0, 2/h)$ and $\beta \in (0, 2)$, i.e. $\alpha \in (0, 2/p)$. For the purpose of the following analysis recall that $\beta = p\alpha$.

Corollary 6.1. *When β is fixed then the following holds*

$$\lim_{p\to\infty}\lim_{t\to\infty} p\mathbb{E}[(\tilde{x}_t - x^*)^2] = \frac{\beta\eta h}{(2 - \beta)(2 - \eta h)} \cdot \frac{2 - \beta - \eta h + \beta\eta h}{\beta + \eta h - \beta\eta h} \cdot \frac{\sigma^2}{h^2}.$$

The crucial point of Corollary 6.1 is that the MSE in the limit $t \to \infty$ is in the order of $1/p$ which implies that as the number of processors p grows, the MSE will decrease for the *EASGD* algorithm. Also note that the smaller the β is (recall that $\beta = p\alpha = p\eta\rho$), the more exploration is allowed (small ρ) and simultaneously the smaller the MSE is. The next lemma (Lemma 6.2) shows that *EASGD* algorithm achieves the highest possible rate of convergence when we consider the double averaging sequence (similarly to Polyak & Juditsky (1992)) $\{z_1, z_2, \dots\}$ defined as below

$$z_{t+1} = \frac{1}{t+1}\sum_{k=0}^{t} \tilde{x}_k. \tag{12}$$

Lemma 6.2 (Weak convergence)**.** *If the condition in Equation 11 holds, then the normalized double averaging sequence defined in Equation 12 converges weakly to the normal distribution with zero mean and variance σ^2/ph^2,*

$$\sqrt{t}(z_t - x^*) \rightharpoonup \mathcal{N}(0, \frac{\sigma^2}{ph^2}), \quad t \to \infty. \tag{13}$$

[3] In our notation, $\mathbb{V}$ denotes the variance.

The asymptotic variance in the Lemma 6.2 is optimal with any fixed η and β for which Equation 11 holds. The next lemma (Lemma 6.3) extends the result in Lemma 6.2 to the multi-dimensional setting.

Lemma 6.3 (Weak convergence). *Let h denotes the largest eigenvalue of A. If $(2-\eta h)(2-\beta) > 2\beta/p$, $(2-\eta h)+(2-\beta) > \beta/p$, $\eta > 0$ and $\beta > 0$, then the normalized double averaging sequence converges weakly to the normal distribution with zero mean and the covariance matrix $V = A^{-1}\Sigma(A^{-1})^T$,*

$$\sqrt{tp}(z_t - x^*) \rightharpoonup \mathcal{N}(0, V), \quad t \to \infty. \tag{14}$$

As before, the asymptotic covariance in the Lemma 6.3 is optimal, i.e. meets the Fisher information lower-bound, when the conditions of the lemma are satisfied. The fact that this asymptotic covariance matrix V does not contain any term involving ρ is quite remarkable, since the penalty term ρ does have an impact on the condition number of the Hessian in Equation 2.

7 EXPERIMENTS

In this section we compare the performance of *EASGD* and *EAMSGD* with various competitor methods listed below:

- *DOWNPOUR* Dean et al. (2012): the pseudo-code of the implementation of *DOWNPOUR* used in this paper is enclosed in the Supplementary material.
- *Momentum DOWNPOUR* (*MDOWNPOUR*)[4], where the Nesterov's momentum scheme is applied to the master's update (note it is unclear how to apply it to the local workers). The pseudo-code is in the Supplementary material.
- A method that we call *ADOWNPOUR*, where we compute the average over time of the center variable $\tilde{x}$ as follows:

  $$z_{t+1} = \alpha_{t+1} z_t + (1-\alpha_{t+1})\tilde{x}_t,$$

  and $\alpha_{t+1} = \frac{t}{t+1}$ is a moving rate, and $z_0 = \tilde{x}$. t denotes the master clock, which is initialized to 0 and incremented every time the center variable $\tilde{x}$ gets updated.
- A method that we call *MVADOWNPOUR*, where we compute the moving average of the center variable $\tilde{x}$ as follows:

  $$z_{t+1} = \alpha z_t + (1-\alpha)\tilde{x}_t,$$

  and the moving rate α was chosen to be constant, and $z_0 = \tilde{x}$. t denotes the master clock and is defined in the same way as for the *ADOWNPOUR* method.
- *SGD* Bottou (1998) with constant learning rate η.
- *Momentum SGD* (*MSGD*) Sutskever et al. (2013) with constant momentum term.
- *ASGD* Polyak & Juditsky (1992) with moving rate $\alpha_{t+1} = \frac{t}{t+1}$.
- *MVASGD* Polyak & Juditsky (1992) with moving rate α set to a constant.

SGD, *MSGD*, *ASGD* and *MVASGD* are run on a single GPU processor ($p = 1$). Remaining methods are run on multiple GPU processors with different settings of p as will be shown later. We perform experiments in a deep learning setting on two benchmark datasets: CIFAR-10 (we refer to it as *CIFAR*) [5] and ImageNet ILSVRC 2013 (we refer to it as *ImageNet*) [6]. We focus on the classification task with deep convolutional neural networks. We next explain the experimental setup and details regarding data sampling.

7.1 EXPERIMENTAL SETUP

For all our experiments we use 4 computing nodes interconnected with InfiniBand. Each node has 4 Titan GPU processors where each local worker corresponds to one GPU processor. The center

[4] It is implemented for the case when $\tau = 1$ as it is unclear how to apply it otherwise.

[5] Downloaded from http://www.cs.toronto.edu/~kriz/cifar.html.

[6] Downloaded from http://image-net.org/challenges/LSVRC/2013.

variable of the master is stored and updated on the centralized parameter server, which keeps the current state of center variable sharded across p CPU's (p is the number of local workers). This concept of a centralized sharded parameter server is identical to the common implementation of *DOWNPOUR* Dean et al. (2012). All the methods we use are implemented in Torch7[7]. The Message Passing Interface (MPI) implementation MVAPICH2[8] is used for the GPU-CPU communication.

For the *ImageNet* experiment we use the similar approach to Sermanet et al. (2013). To be more precise, we re-size each RGB image so that the smallest dimension is 256 pixels. We also re-scale each pixel value to the interval $[0, 1]$. We then extract random crops (and their horizontal flips) of size $3 \times 221 \times 221$ pixels and present these to the network in mini-batches of size 128. We will now explain the details of the network architecture. In all our experiments we use the cross-entropy error criterion with a network having a softmax output non-linearity. All layers use rectified linear units. To provide the details of the architecture of the network we will first introduce some notation. Let (c, y) denotes the sizes of the input to each layer, where c is the number of color channels and y is the horizontal and, at the same time, vertical dimension of the input (thus the input size is really $c \times y \times y$). Let $C(a, b)$ denotes the fully-connected convolutional layer with the filter size $a \times a$ and stride $b \times b$ and let $P(a, b)$ denotes the max pooling layer with the pool size $a \times a$ and stride $b \times b$. $DO(a)$ is a shortcut for the fully connected layer with dropout rate equal to a and $SMax$ is the shortcut for the fully connected layer with softmax output non-linearity. For the *ImageNet* experiment we then use the following network architecture

Input: $(\mathbf{3}, \mathbf{221}) \rightarrow C(7, 2) \rightarrow (96, 108) \rightarrow P(3, 3) \rightarrow (96, 36) \rightarrow C(5, 1) \rightarrow (256, 32) \rightarrow P(2, 2) \rightarrow (256, 16) \rightarrow C(3, 1) \rightarrow (384, 14) \rightarrow C(2, 1) \rightarrow (384, 13) \rightarrow C(2, 1) \rightarrow (256, 12) \rightarrow P(2, 2) \rightarrow (256, 6) \rightarrow DO(0.5) \rightarrow (4096, 1) \rightarrow DO(0.5) \rightarrow (4096, 1) \rightarrow SMax \rightarrow (1000, 1)$.

For the *CIFAR* experiment we use the similar approach to Wan et al. (2013). We use the original RGB image of size $3 \times 32 \times 32$. As before, we re-scale each pixel value to the interval $[0, 1]$. We then extract random crops (and their horizontal flips) of size $3 \times 28 \times 28$ pixels and present these to the network in mini-batches of size 128. The network architecture that we use for the *CIFAR* experiment can then be summarized as follows

Input: $(\mathbf{3}, \mathbf{28}) \rightarrow C(5, 1) \rightarrow (64, 24) \rightarrow P(2, 2) \rightarrow (64, 12) \rightarrow C(5, 1) \rightarrow (128, 8) \rightarrow P(2, 2) \rightarrow (128, 4) \rightarrow C(3, 1) \rightarrow (64, 2) \rightarrow DO(0.5) \rightarrow (256, 1) \rightarrow SMax \rightarrow (10, 1)$.

In our experiments all the methods we run use the same initial parameter vector chosen randomly, except that we set all the biases to zero for *CIFAR* case and to 0.1 for *ImageNet* case. This parameter is used to initialize the master and all the local workers. We also use l_2-regularization for the loss function with the regularization parameter λ. For *ImageNet* we use $\lambda = 10^{-5}$ and for *CIFAR* we use $\lambda = 10^{-4}$. We will next explain precisely how the dataset is sampled by each local worker.

7.2 Sampling the dataset by the local workers

The general parallel data loading scheme on a single machine is as follows: we use k CPUs, where $k = 8$, to load the data in parallel. Each data loader reads from the memory-mapped (mmap) file a chunk of c raw images (preprocessing was described in the previous subsection) and their labels (for *CIFAR* $c = 512$ and for *ImageNet* $c = 64$). For the *CIFAR*, the mmap file of each data loader contains the entire dataset whereas for *ImageNet*, each mmap file of each data loader contains different $1/k$ fractions of the entire dataset. A chunk of data is always sent by one of the data loaders to the first worker who requests the data. The next worker requesting the data from the same data loader will get the next chunk. Each worker requests in total k data chunks from k different data loaders and then process them before asking for new data chunks. Notice that each data loader cycles through the data in the mmap file, sending consecutive chunks to the workers in order in which it receives requests from them. When the data loader reaches the end of the mmap file, it selects the address in memory uniformly at random from the interval $[0, s]$, where $s =$ (number of images in the mmap file modulo mini-batch size), and uses this address to start cycling again through the data in the mmap file. After the local worker receives the k data chunks from the data loaders, it shuffles them and divides it into mini-batches of size 128.

[7] `http://torch.ch`

[8] `http://mvapich.cse.ohio-state.edu`

7.3 EXPERIMENTAL RESULTS

For all experiments in this section we use *EASGD* with $\beta = 0.9$[9], for all momentum-based methods we use the momentum term δ set to $\delta = 0.99$ and finally for *MVADOWNPOUR* we use the moving rate α set to $\alpha = 0.999$. We start with the experiment on *CIFAR* dataset with $p = 4$ local workers running on the one computing node. For all the methods, except *MDOWNPOUR*, we examined the communication periods from the following set $\tau = \{1, 4, 16, 64\}$ (recall that for *MDOWNPOUR* τ is equal to $\tau = 1$). For comparison we also report the performance of *MSGD* which outperformed *SGD*, *ASGD* and *MVASGD* as shown in Figure 6 in the Supplementary material (all figures should be read in color). For each method we examined a wide range of learning rates (those are summarized in Table 1 in the Supplementary material). The experiment was re-run 3 times independently and for each method we report its best performance measured by the smallest achievable test error.

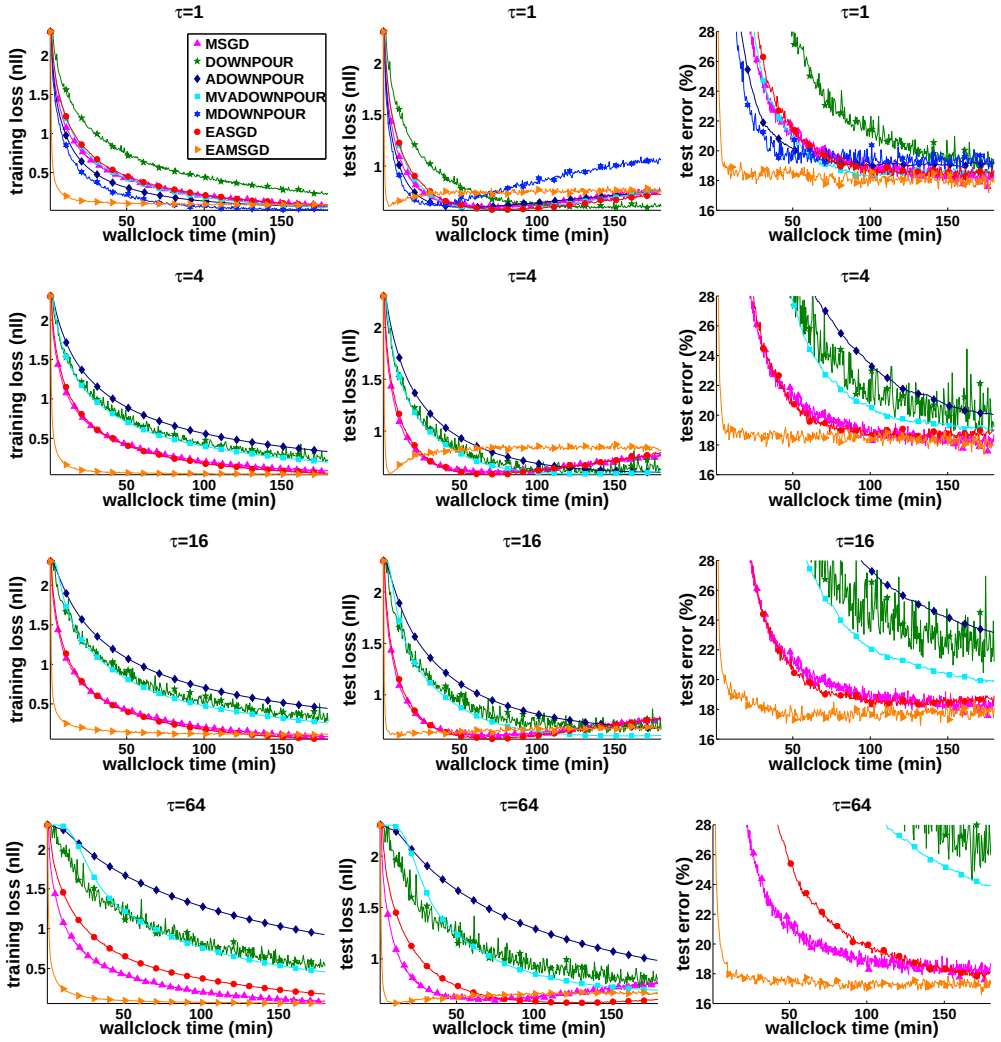

Figure 1: Convergence of the training and test loss (negative log-likelihood) and the test error computed for the center variable as a function of wallclock time for different values of τ on the *CIFAR* experiment. $p = 4$.

Figure 1 captures the convergence of the training and test loss (negative log-likelihood) and the test error[10] computed for the center variable as a function of wallclock time for different values

[9]Intuitively one can think of the 'effective β' as being proportional to β/τ in the asynchronous setting. Also recall that $\beta = p\alpha$.

[10]The training and test loss and the test error are computed from the center patch 28×28 for the *CIFAR* experiment and the center patch 221×221 for the *ImageNet* experiment. For the *ImageNet* experiment, the training loss is measured on a fixed subset of the training data of size 50,000.

of τ. We conclude that all $DOWNPOUR$-based methods, i.e. *DOWNPOUR*, *ADOWNPOUR*, *MVADOWNPOUR* and *MDOWNPOUR*, achieve their best performance (test error) for small τ ($\tau \in \{1, 4\}$). Also *DOWNPOUR* and *ADOWNPOUR* become highly instable for $\tau \in \{16, 64\}$ whereas in this case *MVADOWNPOUR* has stable but very slow convergence. Simultaneously, *EAMSGD* significantly outperforms comparator methods for all values of τ by having faster convergence. It also finds better-quality solution measured by the test error and this advantage becomes more significant for $\tau \in \{16, 64\}$. For $\tau = 1$ and $\tau = 4$ we observe a pronounced overfitting effect for the *EAMSGD* method. We conjecture that these values of τ are too small to allow local workers to perform exploration which hurts the test performance of the algorithm. Note that the tendency to achieve worst test performance with smaller τ is also characteristic for the *EASGD* algorithm which simultaneously outperforms all $DOWNPOUR$-based methods for higher values of τ ($\tau \in \{16, 64\}$).

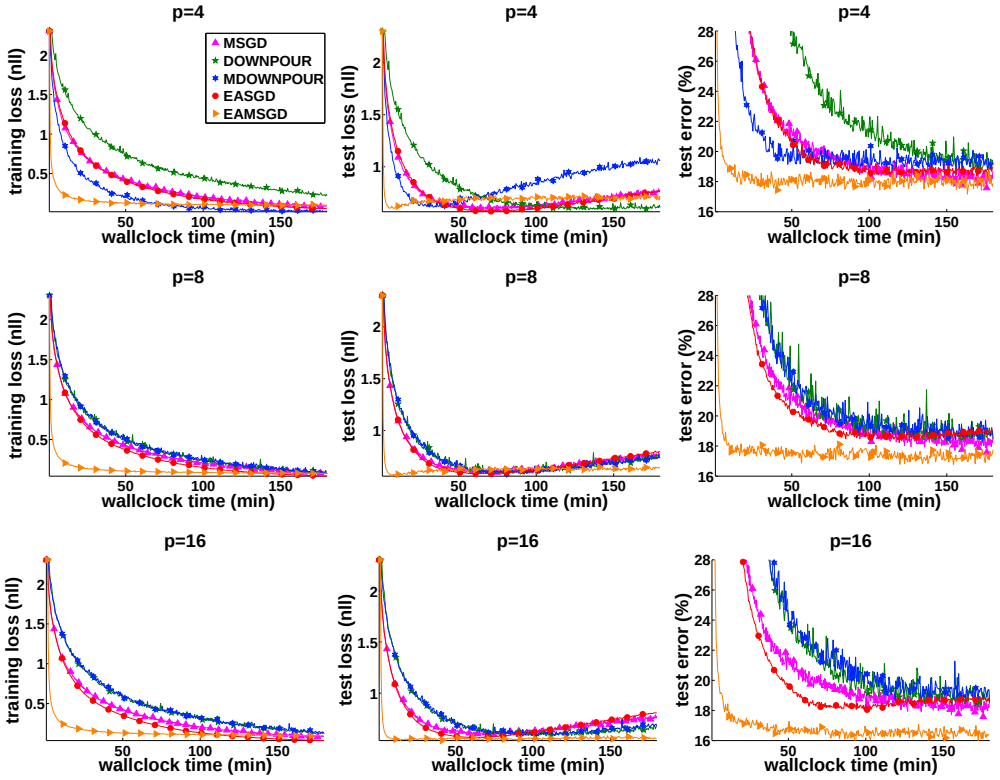

Figure 2: Convergence of the training and test loss (negative log-likelihood) and the test error computed for the center variable as a function of wallclock time for different values of p on the *CIFAR*.

We next explore different number of local workers p from the set $p = \{4, 8, 16\}$ for the *CIFAR* experiment, and $p = \{4, 8\}$ for the *ImageNet* experiment. The *CIFAR* experiment was again re-run 3 times independently whereas for the *ImageNet* experiment we report the results of one run. We compared *EASGD*, *EAMSGD*, *DOWNPOUR* and *MDOWNPOUR* methods for the *CIFAR* experiment, and we compared *EASGD*, *EAMSGD* and *DOWNPOUR* methods for the *ImageNet* experiment. We also show the best performer *MSGD* from among the following methods: *SGD*, *MSGD*, *ASGD* and *MVASGD* (see Figure 6 and Figure 7 in the Supplementary material). *EASGD* and *EAMSGD* were run with $\tau = 10$ whereas *DOWNPOUR* and *MDOWNPOUR* were run with $\tau = 1$. Table 2 and 3 in the Supplementary material summarizes the learning rates explored for respectively the *CIFAR* and *ImageNet* experiments. For the *ImageNet* experiment we used the rule of the thumb to decrease the initial learning rate twice, first time we divided it by 5 and the second time by 2, when we observed that the decrease of the training loss saturates.

Figure 2 captures the convergence of the training and test loss and the test error computed for the center variable as a function of wallclock time for different values of p on the *CIFAR* experiment. Analogous results are captured in Figure 3 for the *ImageNet* experiment. From the *CIFAR* experiment we conclude that *EASGD* converges faster than *DOWNPOUR* and *MDOWNPOUR* while *EAMSGD* significantly outperforms competitor methods for $p = 8$ and $p = 16$. Also the lowest

achievable test error by either *EASGD* or *EAMSGD* decreases with larger p which again can potentially be explained by the fact that larger p allows for more exploration of the parameter space. The results obtained for the *ImageNet* experiment again shows the advantage of *EAMSGD* over the competitor methods. To make sense of the wallclock time, we report in the Table 4 of Supplementary material its breakdown into computation time, data loading time and parameter communication time. We also summarize in the Supplementary material (Figure 8 and 9) the wall clock time needed to achieve the same level of the test error for all the methods.

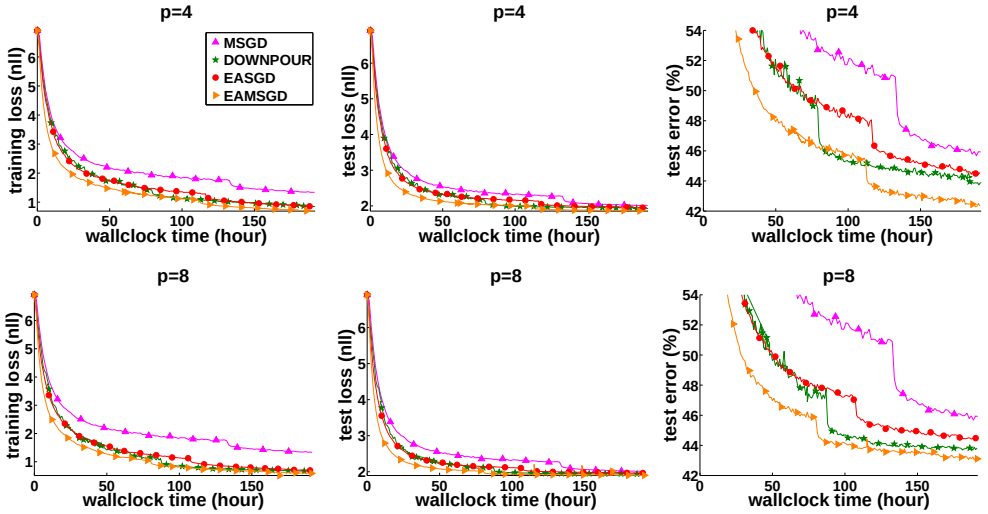

Figure 3: Convergence of the training and test loss (negative log-likelihood) and the test error computed for the center variable as a function of wallclock time for different values of p on the *ImageNet*.

The results showed so far suggest that more exploration lead to better test performance. It should be remembered however that there exists a trade-off between exploration and exploitation. Figure 4 shows this trade-off. We compare the performance of respectively *EAMSGD* and *EASGD* for different learning rates η when $p = 16$ and $\tau = 10$. We observe from Figure 4 that higher learning rates η lead to better test performance for the *EAMSGD* algorithm which potentially can be justified by the fact that they sustain higher fluctuations of the local workers. We conjecture that higher fluctuations lead to more exploration and simultaneously they also impose higher regularization. This picture however seems to be opposite for the *EASGD* algorithm for which larger learning rates hurt the performance of the method and lead to overfitting. Interestingly in this experiment for both *EASGD* and *EAMSGD* algorithm, the learning rate for which the worst training performance was achieved simultaneously led to the best test performance.

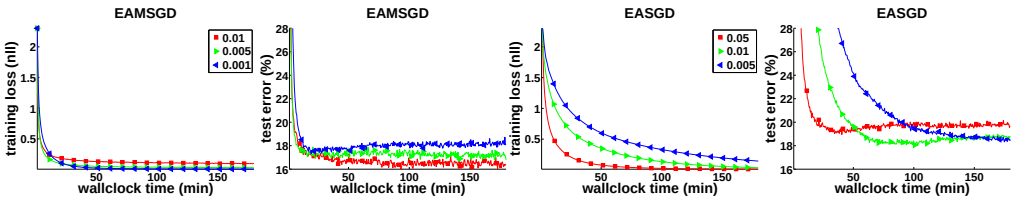

Figure 4: Convergence of the training loss (negative log-likelihood) and the test error computed for the center variable as a function of wallclock time for *EAMSGD* and *EASGD* run with different values of η on the *CIFAR* experiment. $p = 16$, $\tau = 10$.

8 Conclusion

In this paper we describe a new algorithm called *EASGD* and its variants for training deep neural networks in the stochastic setting when the computations are parallelized over multiple GPUs. Experiments demonstrate that this new algorithm quickly achieves improvement in test error compared to more common baseline approaches such as *DOWNPOUR* and its variants. We show that our approach is plausible under communication constraints. The different behavior of the *EASGD* algo-

rithm from its momentum-based variant *EAMSGD* is intriguing and will be a subject of our future studies.

ACKNOWLEDGMENTS

The authors thank R. Power, J. Bruna and O. Henaff and the referees for valuable feedback.

REFERENCES

Agarwal, A. and Duchi, J. Distributed delayed stochastic optimization. In *NIPS*. 2011.

Azadi, S. and Sra, S. Towards an optimal stochastic alternating direction method of multipliers. In *ICML*, 2014.

Bekkerman, R., Bilenko, M., and Langford, J. Scaling up machine learning: Parallel and distributed approaches. Camridge Universityy Press, 2011.

Bertsekas, D. P. and Tsitsiklis, J. N. *Parallel and Distributed Computation*. Prentice Hall, 1989.

Borkar, V. Asynchronous stochastic approximations. *SIAM Journal on Control and Optimization*, 36(3):840–851, 1998.

Bottou, L. Online algorithms and stochastic approximations. In *Online Learning and Neural Networks*. Cambridge University Press, 1998.

Boyd, S., Parikh, N., Chu, E., Peleato, B., and Eckstein, J. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn.*, 3(1): 1–122, 2011.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Le, Q., Mao, M., Ranzato, M., Senior, A., Tucker, P., Yang, K., and Ng, A. Large scale distributed deep networks. In *NIPS*. 2012.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pp. 1106–1114, 2012.

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

Langford, J., Smola, A., and Zinkevich, M. Slow learners are fast. In *NIPS*, 2009.

Nedić, A., Bertsekas, D.P., and Borkar, V.S. Distributed asynchronous incremental subgradient methods. In *Inherently Parallel Algorithms in Feasibility and Optimization and their Applications*, volume 8 of *Studies in Computational Mathematics*, pp. 381 – 407. 2001.

Nesterov, Y. Smooth minimization of non-smooth functions. *Math. Program.*, 103(1):127–152, 2005.

Nocedal, J. and Wright, S.J. Numerical optimization, second edition. *Numerical optimization*, pp. 497–528, 2006.

Ouyang, Hua, He, Niao, Tran, Long, and Gray, Alexander. Stochastic alternating direction method of multipliers. In *Proceedings of the 30th International Conference on Machine Learning*, pp. 80–88, 2013.

Paine, T., Jin, H., Yang, J., Lin, Z., and Huang, T. Gpu asynchronous stochastic gradient descent to speed up neural network training. In *Arxiv*. 2013.

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

Recht, B., Re, C., Wright, S. J., and Niu, F. Hogwild: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. In *NIPS*, 2011.

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and LeCun, Y. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. *ArXiv*, 2013.

Shamir, O. Fundamental limits of online and distributed algorithms for statistical learning and estimation. In *NIPS*. 2014.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. On the importance of initialization and momentum in deep learning. In *ICML*, 2013.

Wan, L., Zeiler, M. D., Zhang, S., LeCun, Y., and Fergus, R. Regularization of neural networks using dropconnect. In *ICML*, 2013.

Wei, Ermin and Ozdaglar, Asuman. Distributed alternating direction method of multipliers. In *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, pp. 5445–5450. IEEE, 2012.

Yadan, O., Adams, K., Taigman, Y., and Ranzato, MA. Multi-gpu training of convnets. In *Arxiv*. 2013.

Zhang, R. and Kwok, J. Asynchronous distributed admm for consensus optimization. In *ICML*, 2014.

Zinkevich, M., Weimer, M., Smola, A., and Li, L. Parallelized stochastic gradient descent. In *NIPS*, 2010.

Deep learning with Elastic Averaging SGD (Supplementary Material)

9 THEORETICAL PROOFS

9.1 PROOF OF LEMMA 6.1

Proof. Substituting the gradient from Equation 8 into the update rule used by each local worker in the synchronous *EASGD* algorithm (Equation 4) we obtain

$$x_{t+1}^i = x_t^i - \eta(Ax_t^i - b - \xi_t^i) - \alpha(x_t^i - \tilde{x}_t), \tag{15}$$

$$\tilde{x}_{t+1} = \tilde{x}_t + \sum_{i=1}^{p} \alpha(x_t^i - \tilde{x}_t), \tag{16}$$

where η is the learning rate, and α is the moving rate. Recall that $\alpha = \eta\rho$ and $A = h$.

For the ease of notation we redefine $\tilde{x}_t$ and x_t^i as follows:

$$\tilde{x}_t \triangleq \tilde{x}_t - x^* \quad \text{and} \quad x_t^i \triangleq x_t^i - x^*.$$

We prove the lemma by explicitly solving the linear equations 15 and 16. Let $x_t = (x_t^1, \ldots, x_t^p, \tilde{x}_t)^T$. We rewrite the recursive relation captured in Equation 15 and 16 as simply

$$x_{t+1} = Mx_t + b_t,$$

where the drift matrix M is defined as

$$M = \begin{bmatrix} 1-\alpha-\eta h & 0 & ... & 0 & \alpha \\ 0 & 1-\alpha-\eta h & 0 & ... & \alpha \\ ... & 0 & ... & 0 & ... \\ 0 & ... & 0 & 1-\alpha-\eta h & \alpha \\ \alpha & \alpha & ... & \alpha & 1-p\alpha \end{bmatrix},$$

and the (diffusion) vector $b_t = (\eta\xi_t^1, \ldots, \eta\xi_t^p, 0)^T$.

Note that one of the eigenvalues of matrix M, that we call ϕ, satisfies $(1-\alpha-\eta h-\phi)(1-p\alpha-\phi) = p\alpha^2$. The corresponding eigenvector is $(1, 1, \ldots, 1, -\frac{p\alpha}{1-p\alpha-\phi})^T$. Let u_t be the projection of x_t onto this eigenvector. Thus $u_t = \sum_{i=1}^p (x_t^i - \frac{\alpha}{1-p\alpha-\phi}\tilde{x}_t)$. Let furthermore $\xi_t = \sum_{i=1}^p \xi_t^i$. Therefore we have

$$u_{t+1} = \phi u_t + \eta\xi_t. \tag{17}$$

By combining Equation 16 and 17 as follows

$$\tilde{x}_{t+1} = \tilde{x}_t + \sum_{i=1}^{p} \alpha(x_t^i - \tilde{x}_t) = (1-p\alpha)\tilde{x}_t + \alpha(u_t + \frac{p\alpha}{1-p\alpha-\phi}\tilde{x}_t)$$

$$= (1 - p\alpha + \frac{p\alpha^2}{1-p\alpha-\phi})\tilde{x}_t + \alpha u_t = \gamma\tilde{x}_t + \alpha u_t,$$

where the last step results from the following relations: $\frac{p\alpha^2}{1-p\alpha-\phi} = 1-\alpha-\eta h-\phi$ and $\phi + \gamma = 1-\alpha-\eta h+1-p\alpha$. Thus we obtained

$$\tilde{x}_{t+1} = \gamma\tilde{x}_t + \alpha u_t. \tag{18}$$

Based on Equation 17 and 18, we can then expand u_t and $\tilde{x}_t$ recursively,

$$u_{t+1} = \phi^{t+1}u_0 + \phi^t(\eta\xi_0) + \ldots + \phi^0(\eta\xi_t), \tag{19}$$

$$\tilde{x}_{t+1} = \gamma^{t+1}\tilde{x}_0 + \gamma^t(\alpha u_0) + \ldots + \gamma^0(\alpha u_t). \tag{20}$$

Substituting $u_0, u_1, \ldots, u_t$, each given through Equation 19, into Equation 20 we obtain

$$\tilde{x}_t = \gamma^t \tilde{x}_0 + \frac{\gamma^t - \phi^t}{\gamma - \phi}\alpha u_0 + \alpha\eta \sum_{l=1}^{t-1} \frac{\gamma^{t-l} - \phi^{t-l}}{\gamma - \phi}\xi_{l-1}. \tag{21}$$

To be more specific, the Equation 21 is obtained by integrating by parts,

$$\begin{aligned}
\tilde{x}_{t+1} &= \gamma^{t+1}\tilde{x}_0 + \sum_{i=0}^{t} \gamma^{t-i}(\alpha u_i) \\
&= \gamma^{t+1}\tilde{x}_0 + \sum_{i=0}^{t} \gamma^{t-i}(\alpha(\phi^i u_0 + \sum_{l=0}^{i-1} \phi^{i-1-l}\eta\xi_l)) \\
&= \gamma^{t+1}\tilde{x}_0 + \sum_{i=0}^{t} \gamma^{t-i}\phi^i(\alpha u_0) + \sum_{l=0}^{t-1}\sum_{i=l+1}^{t} \gamma^{t-i}\phi^{i-1-l}(\alpha\eta\xi_l) \\
&= \gamma^{t+1}\tilde{x}_0 + \frac{\gamma^{t+1} - \phi^{t+1}}{\gamma - \phi}(\alpha u_0) + \sum_{l=0}^{t-1} \frac{\gamma^{t-l} - \phi^{t-l}}{\gamma - \phi}(\alpha\eta\xi_l).
\end{aligned}$$

Since the random variables ξ_l are i.i.d, we may sum the variance term by term as follows

$$\begin{aligned}
\sum_{l=0}^{t-1}\left(\frac{\gamma^{t-l} - \phi^{t-l}}{\gamma - \phi}\right)^2 &= \sum_{l=0}^{t-1} \frac{\gamma^{2(t-l)} - 2\gamma^{t-l}\phi^{t-l} + \phi^{2(t-l)}}{(\gamma - \phi)^2} \\
&= \frac{1}{(\gamma - \phi)^2}\left(\frac{\gamma^2 - \gamma^{2(t+1)}}{1 - \gamma^2} - 2\frac{\gamma\phi - (\gamma\phi)^{t+1}}{1 - \gamma\phi} + \frac{\phi^2 - \phi^{2(t+1)}}{1 - \phi^2}\right).
\end{aligned} \tag{22}$$

Note that $\mathbb{E}[\xi_t] = \sum_{i=1}^{p} \mathbb{E}[\xi_t^i] = 0$ and $\mathbb{V}[\xi_t] = \sum_{i=1}^{p} \mathbb{V}[\xi_t^i] = p\sigma^2$. These two facts, the equality in Equation 21 and Equation 22 can then be used to compute $\mathbb{E}[\tilde{x}_t]$ and $\mathbb{V}[\tilde{x}_t]$ as given in Equation 9 and 10 in Lemma 6.1.

□

9.2 Proof of Corollary 6.1

Proof. Note that when β is fixed, $\lim_{p\to\infty} a = \eta h + \beta$ and $c^2 = \eta h \beta$. Then $\lim_{p\to\infty} \phi = \min(1 - \beta, 1 - \eta h)$ and $\lim_{p\to\infty} \gamma = \max(1 - \beta, 1 - \eta h)$. Also note that using Lemma 6.1 we obtain

$$\begin{aligned}
\lim_{t\to\infty} \mathbb{E}\tilde{x}_t^2 &= \frac{\beta^2\eta^2}{(\gamma - \phi)^2}\left(\frac{\gamma^2}{1 - \gamma^2} + \frac{\phi^2}{1 - \phi^2} - \frac{2\gamma\phi}{1 - \gamma\phi}\right)\frac{\sigma^2}{p} \\
&= \frac{\beta^2\eta^2}{(\gamma - \phi)^2}\left(\frac{\gamma^2(1 - \phi^2)(1 - \phi\gamma) + \phi^2(1 - \gamma^2)(1 - \phi\gamma) - 2\gamma\phi(1 - \gamma^2)(1 - \phi^2)}{(1 - \gamma^2)(1 - \phi^2)(1 - \gamma\phi)}\right)\frac{\sigma^2}{p} \\
&= \frac{\beta^2\eta^2}{(\gamma - \phi)^2}\left(\frac{(\gamma - \phi)^2(1 + \gamma\phi)}{(1 - \gamma^2)(1 - \phi^2)(1 - \gamma\phi)}\right)\frac{\sigma^2}{p} \\
&= \frac{\beta^2\eta^2}{(1 - \gamma^2)(1 - \phi^2)} \cdot \frac{1 + \gamma\phi}{1 - \gamma\phi} \cdot \frac{\sigma^2}{p}.
\end{aligned}$$

Corollary 6.1 is obtained by plugining in the limiting values of ϕ and γ. □

9.3 Visualizing the Lemma 6.1

In Figure 5, we illustrate the dependence of MSE on β, η and the number of processors p over time t. We consider the large-noise setting where $\tilde{x}_0 = x_0^i = 1$, $h = 1$ and $\sigma = 10$. The MSE error is color-coded such that the deep blue color corresponds to the MSE equal to 10^{-3}, the green color corresponds to the MSE equal to 1, the red color corresponds to MSE equal to 10^3 and the dark red color corresponds to the divergence of algorithm *EASGD* (condition in Equation 11 is then violated). The plot shows that we can achieve more significant variance reduction effect by increasing the number of local workers p.

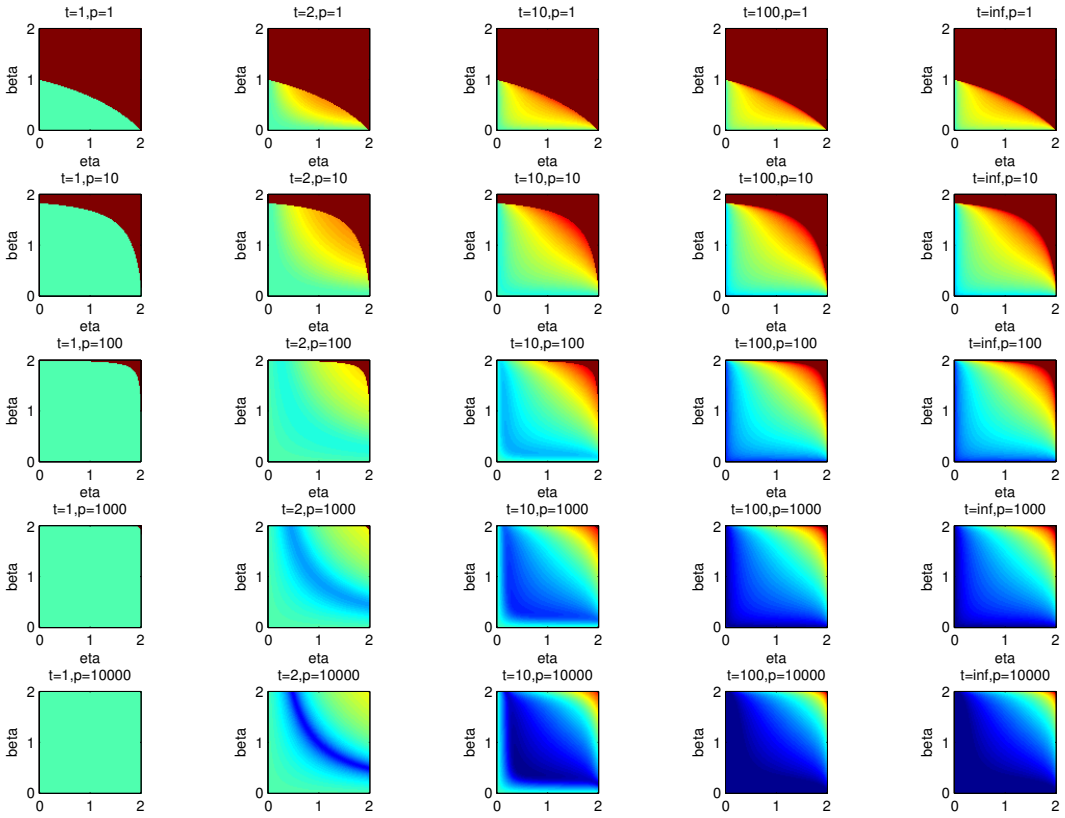

Figure 5: Theoretical mean squared error (MSE) of the center $\tilde{x}$ in the quadratic case, with various choices of the learning rate η (horizontal within each block), and the moving rate $\beta = p\alpha$ (vertical within each block), the number of processors $p = \{1, 10, 100, 1000, 10000\}$ (vertical across blocks), and the time steps $t = \{1, 2, 10, 100, \infty\}$ (horizontal across blocks). The MSE is plotted in log scale, ranging from 10^{-3} to 10^{3} (from deep blue to red). The dark red (i.e. on the upper-right corners) indicates divergence.

9.4 Proof of Lemma 6.2

Proof. As in the proof of Lemma 6.1, for the ease of notation we redefine $\tilde{x}_t$ and x_t^i as follows:

$$\tilde{x}_t \triangleq \tilde{x}_t - x^* \quad \text{and} \quad x_t^i \triangleq x_t^i - x^*.$$

Also recall that $\{\xi_t^i\}$'s are *i.i.d.* random variables (noise) with zero mean and the same covariance $\Sigma \succ 0$. We are interested in the asymptotic behavior of the double averaging sequence $\{z_1, z_2, \dots\}$ defined as

$$z_{t+1} = \frac{1}{t+1} \sum_{k=0}^{t} \tilde{x}_k. \tag{23}$$

Recall the Equation 21 from the proof of Lemma 6.1 (for the convenience it is provided below):

$$\tilde{x}_k = \gamma^k \tilde{x}_0 + \alpha u_0 \frac{\gamma^k - \phi^k}{\gamma - \phi} + \alpha\eta \sum_{l=1}^{k-1} \frac{\gamma^{k-l} - \phi^{k-l}}{\gamma - \phi} \xi_{l-1},$$

where $\xi_t = \sum_{i=1}^{p} \xi_t^i$. Therefore

$$\begin{aligned}
\sum_{k=0}^{t} \tilde{x}_k &= \frac{1-\gamma^{t+1}}{1-\gamma} \tilde{x}_0 + \alpha u_0 \frac{1}{\gamma - \mu} \left(\frac{1-\gamma^{t+1}}{1-\gamma} - \frac{1-\phi^{t+1}}{1-\phi} \right) + \alpha\eta \sum_{l=1}^{t-1} \sum_{k=l+1}^{t} \frac{\gamma^{k-l} - \phi^{k-l}}{\gamma - \phi} \xi_{l-1} \\
&= O(1) + \alpha\eta \sum_{l=1}^{t-1} \frac{1}{\gamma - \phi} \left(\gamma \frac{1-\gamma^{t-l}}{1-\gamma} - \phi \frac{1-\phi^{t-l}}{1-\phi} \right) \xi_{l-1}
\end{aligned}$$

Note that the only non-vanishing term (in weak convergence) of $1/\sqrt{t}\sum_{k=0}^{t}\tilde{x}_k$ as $t \to \infty$ is

$$\frac{1}{\sqrt{t}}\alpha\eta\sum_{l=1}^{t-1}\frac{1}{\gamma-\phi}\left(\frac{\gamma}{1-\gamma}-\frac{\phi}{1-\phi}\right)\xi_{l-1}. \tag{24}$$

Also recall that $\mathbb{V}[\xi_{l-1}] = p\sigma^2$ and

$$\frac{1}{\gamma-\phi}\left(\frac{\gamma}{1-\gamma}-\frac{\phi}{1-\phi}\right) = \frac{1}{(1-\gamma)(1-\phi)} = \frac{1}{\eta hp\alpha}.$$

Therefore the expression in Equation 24 is asymptotically normal with zero mean and variance σ^2/ph^2.

□

9.5 Proof of Lemma 6.3

Proof. Since A is symmetric, one can use the proof technique of Lemma 6.2 to prove Lemma 6.3 by diagonalizing the matrix A. We will not go into the details of this proof as we will provide a more general way to look at the system. As in the proof of Lemma 6.1 and Lemma 6.2, for the ease of notation we redefine $\tilde{x}_t$ and x_t^i as follows:

$$\tilde{x}_t \triangleq \tilde{x}_t - x^* \quad \text{and} \quad x_t^i \triangleq x_t^i - x^*.$$

Let the spatial average of the local parameters at time t be denoted as y_t where $y_t = \frac{1}{p}\sum_{i=1}^{p} x_t^i$, and let the average noise be denoted as ξ_t, where $\xi_t = \frac{1}{p}\sum_{i=1}^{p}\xi_t^i$. Equations 15 and 16 can then be reduced to the following

$$y_{t+1} = y_t - \eta(Ay_t - \xi_t) + \alpha(\tilde{x}_t - y_t), \tag{25}$$
$$\tilde{x}_{t+1} = \tilde{x}_t + \beta(y_t - \tilde{x}_t). \tag{26}$$

We focus on the case where the learning rate η and the moving rate α are kept constant over time. Recall $\beta = p\alpha$ and $\alpha = \eta\rho$[11]

Let's introduce the block notation $U_t = (y_t, \tilde{x}_t)$, $\Xi_t = (\eta\xi_t, 0)$, $M = I - \eta L$ and

$$L = \begin{pmatrix} A + \frac{\alpha}{\eta}I & -\frac{\alpha}{\eta}I \\ -\frac{\beta}{\eta}I & \frac{\beta}{\eta}I \end{pmatrix}.$$

From Equations 25 and 26 it follows that $U_{t+1} = MU_t + \Xi_t$. Note that this linear system has a degenerate noise Ξ_t which prevents us from directly applying results of Polyak & Juditsky (1992). Expanding this recursive relation and summing by parts, we have

$$\begin{aligned}\sum_{k=0}^{t} U_k = {} & M^0U_0 + \\ & M^1U_0 + M^0\Xi_0 + \\ & M^2U_0 + M^1\Xi_0 + M^0\Xi_1 + \\ & \dots \\ & M^tU_0 + M^{t-1}\Xi_0 + \cdots + M^0\Xi_{t-1}.\end{aligned}$$

By Lemma 9.1, $\|M\|_2 < 1$ and thus

$$M^0 + M^1 + \cdots + M^t + \cdots = (I - M)^{-1} = \eta^{-1}L^{-1}.$$

[11] As a side note, notice that the center parameter $\tilde{x}_t$ is tracking the spatial average y_t of the local parameters with a non-symmetric spring. To be more precise note that the update on y_{t+1} contains $(\tilde{x}_t - y_t)$ scaled by α, whereas the update on $\tilde{x}_{t+1}$ contains $-(\tilde{x}_t - y_t)$ scaled by β. Since $\alpha = \beta/p$ the impact of the center $\tilde{x}_{t+1}$ on the spatial local average y_{t+1} becomes more negligible as p grows.

Since A is invertible, we get

$$L^{-1} = \begin{pmatrix} A^{-1} & \frac{\alpha}{\beta}A^{-1} \\ A^{-1} & \frac{\eta}{\beta} + \frac{\alpha}{\beta}A^{-1} \end{pmatrix},$$

thus

$$\frac{1}{\sqrt{t}}\sum_{k=0}^{t} U_k = \frac{1}{\sqrt{t}}U_0 + \frac{1}{\sqrt{t}}\eta L^{-1}\sum_{k=1}^{t}\Xi_{k-1} - \frac{1}{\sqrt{t}}\sum_{k=1}^{t} M^{k+1}\Xi_{k-1}.$$

Note that the only non-vanishing term (in weak convergence) of $\frac{1}{\sqrt{t}}\sum_{k=0}^{t} U_k$ is $\frac{1}{\sqrt{t}}(\eta L)^{-1}\sum_{k=1}^{t}\Xi_{k-1}$ thus we have

$$\frac{1}{\sqrt{t}}(\eta L)^{-1}\sum_{k=1}^{t}\Xi_{k-1} \rightharpoonup \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} V & V \\ V & V \end{pmatrix}\right), \tag{27}$$

where $V = A^{-1}\Sigma(A^{-1})^T$. □

Lemma 9.1. *If the following conditions hold:*

$$\begin{aligned} (2-\eta h)(2-p\alpha) &> 2\alpha \\ (2-\eta h)+(2-p\alpha) &> \alpha \\ \eta &> 0 \\ \alpha &> 0 \end{aligned}$$

then $\|M\|_2 < 1$.

Proof. The eigenvalue λ of M and the (non-zero) eigenvector (y, z) of M satisfy

$$M\begin{pmatrix} y \\ z \end{pmatrix} = \lambda\begin{pmatrix} y \\ z \end{pmatrix}. \tag{28}$$

Recall that

$$M = I - \eta L = \begin{pmatrix} I - \eta A - \alpha I & \alpha I \\ \beta I & I - \beta I \end{pmatrix}. \tag{29}$$

From the Equations 28 and 29 we obtain

$$\begin{cases} y - \eta A y - \alpha y + \alpha z = \lambda y \\ \beta y + (1-\beta)z = \lambda z \end{cases}. \tag{30}$$

Since y and z are assumed to be non-zero, we can write $z = \beta y/(\lambda + \beta - 1)$. Then the Equation 30 can be reduced to

$$\eta A y = (1 - \alpha - \lambda)y + \frac{\alpha\beta}{\lambda + \beta - 1}y. \tag{31}$$

Thus note that y is the eigenvector of A as well. Let λ_A be the eigenvalue of matrix A such that $Ay = \lambda_A y$. Thus based on Equation 31 it follows that

$$\eta\lambda_A = (1 - \alpha - \lambda) + \frac{\alpha\beta}{\lambda + \beta - 1}. \tag{32}$$

Equation 32 is equivalent to

$$\lambda^2 - (2-a)\lambda + (1 - a + c^2) = 0, \tag{33}$$

where $a = \eta\lambda_A + (p+1)\alpha$, $c^2 = \eta\lambda_A p\alpha$. It follows from the condition in Equation 11 that $-1 < \lambda < 1$ iff $\eta > 0$, $\beta > 0$, $(2 - \eta\lambda_A)(2-\beta) > 2\beta/p$ and $(2-\eta\lambda_A) + (2-\beta) > \beta/p$. Let h denote the maximum eigenvalue of A and note that $2 - \eta\lambda_A \geq 2 - \eta h$. This implies that the condition of our lemma is sufficient. □

9.6 CONDITION IN EQUATION 11

We are going to show that

- $\gamma < 1$ iff $c^2 > 0$ (i.e. $\eta > 0$ and $\beta > 0$).
- $\phi > -1$ iff $(2 - \eta h)(2 - \beta) > 2\beta/p$ and $(2 - \eta h) + (2 - \beta) > \beta/p$.
- $\phi = \gamma$ iff $a^2 = 4c^2$ (i.e. $\eta h = \beta = 0$).

Recall that $a = \eta h + (p+1)\alpha$, $c^2 = \eta h p \alpha$, $\gamma = 1 - \frac{a - \sqrt{a^2 - 4c^2}}{2}$, $\phi = 1 - \frac{a + \sqrt{a^2 - 4c^2}}{2}$, and $\beta = p\alpha$. We have

- $\gamma < 1 \Leftrightarrow \frac{a - \sqrt{a^2 - 4c^2}}{2} > 0 \Leftrightarrow a > \sqrt{a^2 - 4c^2} \Leftrightarrow a^2 > a^2 - 4c^2 \Leftrightarrow c^2 > 0$.
- $\phi > -1 \Leftrightarrow 2 > \frac{a + \sqrt{a^2 - 4c^2}}{2} \Leftrightarrow 4 - a > \sqrt{a^2 - 4c^2} \Leftrightarrow 4 - a > 0, (4 - a)^2 > a^2 - 4c^2 \Leftrightarrow 4 - a > 0, 4 - 2a + c^2 > 0 \Leftrightarrow 4 > \eta h + \beta + \alpha, 4 - 2(\eta h + \beta + \alpha) + \eta h \beta > 0$.
- $\phi = \gamma \Leftrightarrow \sqrt{a^2 - 4c^2} = 0 \Leftrightarrow a^2 = 4c^2$.

10 ADDITIONAL PSEUDO-CODES OF THE ALGORITHMS

10.1 DOWNPOUR PSEUDO-CODE

Algorithm 3 captures the pseudo-code of the implementation of the DOWNPOUR used in this paper.

Algorithm 3: DOWNPOUR: Processing by worker i and the master

Input: learning rate η, communication period $\tau \in \mathbb{N}$
Initialize: $\tilde{x}$ is initialized randomly, $x^i = \tilde{x}$, $v^i = 0$, $t^i = 0$

Repeat
  if (τ divides t^i) **then**
    $\tilde{x} \leftarrow \tilde{x} + v^i$
    $x^i \leftarrow \tilde{x}$
    $v^i \leftarrow 0$
  end
  $x^i \leftarrow x^i - \eta g^i_{t^i}(x^i)$
  $v^i \leftarrow v^i - \eta g^i_{t^i}(x^i)$
  $t^i \leftarrow t^i + 1$
Until forever

10.2 MDOWNPOUR PSEUDO-CODE

Algorithms 4 and 5 capture the pseudo-codes of the implementation of momentum DOWNPOUR (MDOWNPOUR) used in this paper. Algorithm 4 shows the behavior of each local worker and Algorithm 5 shows the behavior of the master.

Algorithm 4: MDOWNPOUR: Processing by worker i

Initialize: $x^i = \tilde{x}$

Repeat
  Receive $\tilde{x}$ from the master: $x^i \leftarrow \tilde{x}$
  Compute gradient $g^i = g^i(x^i)$
  Send g^i to the master
Until forever

Algorithm 5: MDOWNPOUR: Processing by the master

Input: learning rate η, momentum term δ
Initialize: $\tilde{x}$ is initialized randomly, $v^i = 0$,

Repeat
  Receive g^i
  $v \leftarrow \delta v - \eta g^i$
  $\tilde{x} \leftarrow \tilde{x} + \delta v$
Until forever

11 EXPERIMENTS - ADDITIONAL MATERIAL

11.1 LEARNING RATES

In Table 1 we summarize the learning rates η explored for each method shown in Figure 1. For all values of τ the same set of learning rates was explored for each method.

Table 1: Learning rates explored for each method shown in Figure 1 (*CIFAR* experiment).

	η
EASGD	$\{0.05, 0.01, 0.005\}$
EAMSGD	$\{0.01, 0.005, 0.001\}$
DOWNPOUR ADOWNPOUR MVADOWNPOUR	$\{0.005, 0.001, 0.0005\}$
MDOWNPOUR	$\{0.00005, 0.00001, 0.000005\}$
SGD, ASGD, MVASGD	$\{0.05, 0.01, 0.005\}$
MSGD	$\{0.001, 0.0005, 0.0001\}$

In Table 2 we summarize the learning rates η explored for each method shown in Figure 2. For all values of p the same set of learning rates was explored for each method.

Table 2: Learning rates explored for each method shown in Figure 2 (*CIFAR* experiment).

	η
EASGD	$\{0.05, 0.01, 0.005\}$
EAMSGD	$\{0.01, 0.005, 0.001\}$
DOWNPOUR	$\{0.005, 0.001, 0.0005\}$
MDOWNPOUR	$\{0.00005, 0.00001, 0.000005\}$
SGD, ASGD, MVASGD	$\{0.05, 0.01, 0.005\}$
MSGD	$\{0.001, 0.0005, 0.0001\}$

In Table 3 we summarize the initial learning rates η we use for each method shown in Figure 3. For all values of p the same set of learning rates was explored for each method.

Table 3: Learning rates explored for each method shown in Figure 3 (*ImageNet* experiment).

	η
EASGD	0.1
EAMSGD	0.001
DOWNPOUR	for $p = 4$: 0.02 for $p = 8$: 0.01
SGD, ASGD, MVASGD	0.05
MSGD	0.0005

11.2 COMPARISON OF *SGD*, *ASGD*, *MVASGD* AND *MSGD*

For all *CIFAR* experiments we always start the averaging for the $ADOWNPOUR$ and $ASGD$ methods from the very beginning of each experiment. For all *ImageNet* experiments we start the averaging for the $ASGD$ at the same time when we first reduce the initial learning rate.

Figure 6 shows the convergence of the training and test loss (negative log-likelihood) and the test error computed for the center variable as a function of wallclock time for *SGD*, *ASGD*, *MVASGD* and *MSGD* ($p = 1$) on the *CIFAR* experiment.

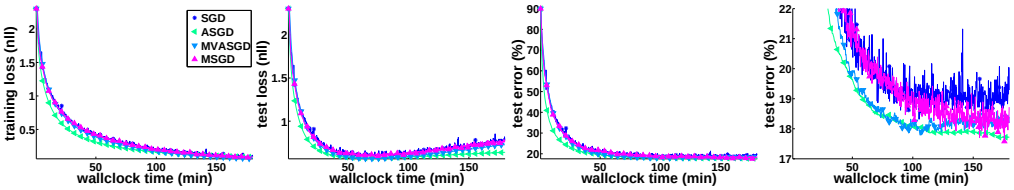

Figure 6: Convergence of the training and test loss (negative log-likelihood) and the test error (original and zoomed) computed for the center variable as a function of wallclock time for *SGD*, *ASGD*, *MVASGD* and *MSGD* ($p = 1$) on the *CIFAR* experiment.

Figure 7 shows the convergence of the training and test loss (negative log-likelihood) and the test error computed for the center variable as a function of wallclock time for *SGD*, *ASGD*, *MVASGD* and *MSGD* ($p = 1$) on the *ImageNet* experiment.

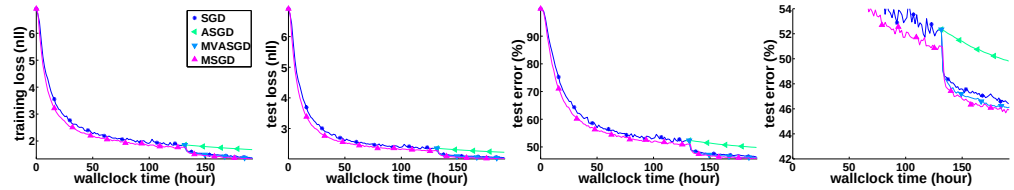

Figure 7: Convergence of the training and test loss (negative log-likelihood) and the test error (original and zoomed) computed for the center variable as a function of wallclock time for *SGD*, *ASGD*, *MVASGD* and *MSGD* ($p = 1$) on the *ImageNet* experiment.

11.3 BREAKDOWN OF THE WALLCLOCK TIME

In addition, we report in the Table 4 the breakdown of the total running time for *EASGD* when $\tau = 10$ (the time breakdown for *EAMSGD* is almost identical) and *DOWNPOUR* when $\tau = 1$ into computation time, data loading time and parameter communication time. For the *CIFAR* experiment the reported time corresponds to processing 400×128 data samples whereas for the *ImageNet* experiment it corresponds to processing 1024×128 data samples. For $\tau = 1$ and $p \in \{8, 16\}$ we observe that the communication time accounts for significant portion of the total running time whereas for $\tau = 10$ the communication time becomes negligible compared to the total running time (recall that based on previous results *EASGD* and *EAMSGD* achieve best performance with larger tau which is ideal in the setting when communication is time-consuming).

	$p=1$	$p=4$	$p=8$	$p=16$
$\tau=1$	12/1/0	11/2/3	11/2/5	11/2/9
$\tau=10$	NA	11/2/1	11/2/1	12/2/1

	$p=1$	$p=4$	$p=8$
$\tau=1$	1248/20/0	1323/24/173	1239/61/284
$\tau=10$	NA	1254/58/7	1266/84/11

Table 4: Approximate computation time, data loading time and parameter communication time [sec] for *DOWNPOUR* (top line for $\tau = 1$) and *EASGD* (the time breakdown for *EAMSGD* is almost identical) (bottom line for $\tau = 10$). Left time corresponds to *CIFAR* experiment and right table corresponds to *ImageNet* experiment.

11.4 TIME SPEED-UP

In the next two figures (Figure 8 and 9) the wall clock time needed to achieve the same level of the test error for all the methods (*EASGD*, *EAMSGD*, *DOWNPOUR* and respectively *MSGD* and *MDOWNPOUR* for the *CIFAR* experiment and *MSGD* for the *ImageNet* experiment as a function of the number of local workers p. For the *CIFAR* (Figure 8) we examined the following levels: $\{21\%, 20\%, 19\%, 18\%\}$ and for the *ImageNet* (Figure 9) we examined: $\{49\%, 47\%, 45\%, 43\%\}$. If some method does not appear on the figure for a given test error level, it indicates that this method never achieved this level. For the *CIFAR* experiment we observe that from among *EASGD*, *DOWNPOUR* and *MDOWNPOUR* methods, the *EASGD* method needs less time to achieve a particular level of test error. We observe that with higher p each of these methods does not necessarily need less time to achieve the same level of test error. This seems counter intuitive though recall that the learning rate for the methods is selected based on the smallest achievable test error. For larger p smaller learning rates were selected than for smaller p which explains our results. Meanwhile, the *EAMSGD* method achieves significant speed-up over other methods for all the test error levels. For the *ImageNet* experiment we observe that all methods outperform *MSGD* and furthermore with $p = 4$ or $p = 8$ each of these methods requires less time to achieve the same level of test error. The *EAMSGD* consistently needs less time than any other method, in particular *DOWNPOUR*, to achieve any of the test error levels.

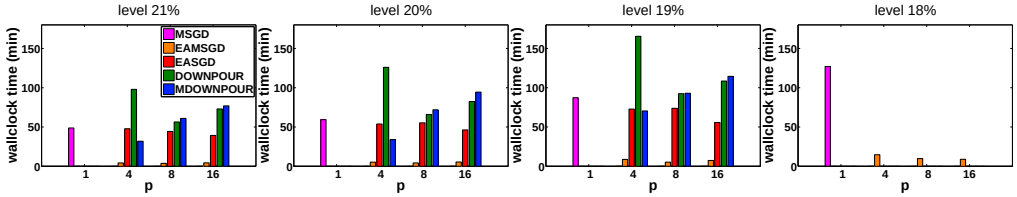

Figure 8: The wall clock time needed to achieve the same level of the test error thr as a function of the number of local workers p on the *CIFAR* experiment. From left to right: $thr = \{21\%, 20\%, 19\%, 18\%\}$.

.

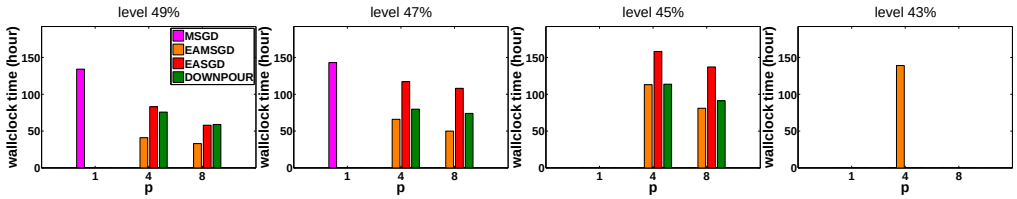

Figure 9: The wall clock time needed to achieve the same level of the test error thr as a function of the number of local workers p on the *ImageNet* experiment. From left to right: $thr = \{49\%, 47\%, 45\%, 43\%\}$.

.