TOWARD UNDERSTANDING THE IMPACT OF STALENESS IN DISTRIBUTED MACHINE LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Most distributed machine learning (ML) systems store a copy of the model parameters locally on each machine to minimize network communication. In practice, in order to reduce synchronization waiting time, these copies of the model are not necessarily updated in lock-step, and can become *stale*. Despite much development in large-scale ML, the effect of staleness on the learning efficiency is inconclusive, mainly because it is challenging to control or monitor the staleness in complex distributed environments. In this work, we study the convergence behaviors of a wide array of ML models and algorithms under delayed updates. Our extensive experiments reveal the rich diversity of the effects of staleness on the convergence of ML algorithms and offer insights into seemingly contradictory reports in the literature. The empirical findings also inspire a new convergence analysis of SGD in non-convex optimization under staleness, matching the best-known convergence rate of $\mathcal{O}(1/\sqrt{T})$.

1 INTRODUCTION

With the advent of big data and complex models, there is a growing body of works on scaling machine learning under synchronous and non-synchronous[1] distributed execution (Dean et al., 2012; Goyal et al., 2017; Li et al., 2014a). These works, however, point to seemingly contradictory conclusions on whether non-synchronous execution outperforms synchronous counterparts in terms of absolute convergence, which is measured by the wall clock time to reach the desired model quality. For deep neural networks, Chilimbi et al. (2014); Dean et al. (2012) show that fully asynchronous systems achieve high scalability and model quality, but others argue that synchronous training converges faster (Chen et al., 2016; Cui et al., 2016). The disagreement goes beyond deep learning models: Ho et al. (2013); Zhang & Kwok (2014); Langford et al. (2009); Lian et al. (2015); Recht et al. (2011) empirically and theoretically show that many algorithms scale effectively under non-synchronous settings, but McMahan & Streeter (2014); Mitliagkas et al. (2016); Hadjis et al. (2016) demonstrate significant penalties from asynchrony.

The crux of the disagreement lies in the trade-off between two factors contributing to the absolute convergence: *statistical efficiency* and *system throughput*. Statistical efficiency measures convergence per algorithmic step (e.g., a mini-batch), while system throughput captures the performance of the underlying implementation and hardware. Non-synchronous execution can improve system throughput due to lower synchronization overheads, which is well understood (Ho et al., 2013; Chen et al., 2016; Cui et al., 2014; Chilimbi et al., 2014). However, by allowing various workers to use *stale* versions of the model that do not always reflect the latest updates, non-synchronous systems can exhibit lower statistical efficiency (Chen et al., 2016; Cui et al., 2016). How statistical efficiency and system throughput trade off in distributed systems, however, is far from clear.

The difficulties in understanding the trade-off arise because statistical efficiency and system throughput are coupled during execution in distributed environments. Non-synchronous executions are in general non-deterministic, which can be difficult to profile. Furthermore, large-scale experiments are sensitive to the underlying hardware and software artifacts, which confounds the comparison between studies. Even when they are controlled, innocuous change in the system configurations such as adding more machines or sharing resources with other workloads can inadvertently alter the underlying staleness levels experienced by ML algorithms, masking the true effects of staleness.

[1]We use the term "non-synchronous" to include both fully asynchronous model (Recht et al., 2011) and bounded asynchronous models such as Stale Synchronous Parallel (Ho et al., 2013).

Understanding the impact of staleness on ML convergence independently from the underlying distributed systems is a crucial step towards decoupling statistical efficiency from the system complexity. The gleaned insights can also guide distributed ML system development, potentially using different synchronization for different problems. In particular, we are interested in the following aspects: Do ML algorithms converge under staleness? To what extent does staleness impact the convergence?

By resorting to simulation study, we side step the challenges faced in distributed execution. We study the impact of staleness on a diverse set of models: Convolutional Neural Networks (CNNs), Deep Neural Networks (DNNs), multi-class Logistic Regression (MLR), Matrix Factorization (MF), Latent Dirichlet Allocation (LDA), and Variational Autoencoders (VAEs). They are addressed by 7 algorithms, spanning across optimization, sampling, and blackbox variational inference. Our findings suggest that while some algorithms are more robust to staleness, no ML method is immune to the negative impact of staleness. We find that all investigated algorithms reach the target model quality under moderate levels of staleness, but the convergence can progress very slowly or fail under high staleness levels. The effects of staleness are also problem dependent. For CNNs and DNNs, the staleness slows down deeper models much more than shallower counterparts. For MLR, a convex objective, staleness has minimal effect. Different algorithms respond to staleness very differently. For example, high staleness levels incur drastically more statistical penalty for RMSProp (Hinton, 2012) and Adam (Kingma & Ba, 2014) optimization than stochastic gradient descent (SGD) and Adagrad (Duchi et al., 2011), which are robust to staleness. Separately, Gibbs sampling for LDA is highly resistant to staleness up to a certain level, beyond which it does not converge to a fixed point. Overall, it appears that staleness is a key governing parameter of ML convergence.

To gain deeper insights, for gradient-based methods we further introduce *gradient coherence* along the optimization path, and show that gradient coherence is a possible explanation for an algorithm's sensitivity to staleness. In particular, our theoretical result establishes the $\mathcal{O}(1/\sqrt{T})$ convergence rate of the asynchronous SGD in nonconvex optimization by exploiting gradient coherence, matching the rate of best-known results (Lian et al., 2015).

2 Related Work

Staleness is reported to help absolute convergence for distributed deep learning in Chilimbi et al. (2014); Dean et al. (2012) and has minimal impact on convergence (Mitliagkas et al., 2016; Hadjis et al., 2016; Lian et al., 2015). But Chen et al. (2016); Cui et al. (2016) show significant negative effects of staleness. LDA training is generally insensitive to staleness (Smola & Narayanamurthy, 2010; Yuan et al., 2015; Wei et al., 2015; Ho et al., 2013), and so is MF training (Yun et al., 2013; Low et al., 2012; Cui et al., 2014; Zhang & Kwok, 2014). However, none of their evaluations quantifies the level of staleness in the systems. By explicitly controlling the staleness, we decouple the distributed execution, which is hard to control, from ML convergence outcomes.

We focus on algorithms that are commonly used in large-scale optimization (Goyal et al., 2017; Chen et al., 2016; Dean et al., 2012), instead of methods specifically designed to minimize synchronization (Neiswanger et al., 2013; Scott et al., 2016; Jordan et al., 2013). Non-synchronous execution has theoretical underpinning (Li et al., 2014b; Ho et al., 2013; Zhang & Kwok, 2014; Lian et al., 2015; Recht et al., 2011). Here we study algorithms that do not necessarily satisfy assumptions in their analyses.

3 Methods

We study six ML models and focus on algorithms that lend itself to data parallelism, which a primary approach for distributed ML. Our algorithms span optimization, sampling, and black box variational inference. Table 1 summarizes the studied models and algorithms.

Simulation Model. Each update generated by worker p needs to be propagated to both worker p's model cache and other worker's model cache. We apply a uniformly random delay model to these updates that are in transit. Specifically, let u_p^t be the update generated at iteration t by worker p. For each worker p' (including p itself), our delay model applies a delay $r_{p,p'}^t \sim \text{Categorical}(0, 1, .., s-1)$, where s is the maximum delay and Categorical() is the categorical distribution placing equal weights on each integer[2]. Under this delay model, update u_p^t shall arrive at worker p' at the start of iteration

[2]We find that geometrically distributed delays have qualitatively similar impacts on convergence. We defer read-my-write consistency to future work.

$t + 1 + r^t_{p,p'}$. The average delay under this model is $\frac{1}{2}s + 1$. Notice that for one worker with $s = 0$ we reduce to the sequential setting. Since model caches on each worker are symmetric, we use the first worker's model to evaluate the model quality. Finally, we are most interested in measuring convergence against the logical time, and wall clock time is in general immaterial as the simulation on a single machine is not optimized for performance.

3.1 Models and Algorithms

Model	Algorithms	Key Parameters	Dataset
CNN	SGD	$\eta = 0.01$	CIFAR10
	Momentum SGD	$\eta = 0.01$, momentum=0.9	
	Adam	$\eta = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$	
	Adagrad	$\eta = 0.01$	
	RMSProp	$\eta = 0.01$, decay=0.9, momentum=0	
DNN/MLR	SGD	$\eta = 0.01$	MNIST
	Adam	$\eta = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$	
LDA	Gibbs Sampling	$\alpha = 0.1, \beta = 0.1$	20 NewsGroup
MF	SGD	$\eta = 0.005$, rank=5, $\lambda = 0.0001$	MovieLens1M
VAE	Blackbox VI (SGD, Adam)	Optimization parameters same as MLR/DNN	MNIST

Table 1: Overview of the models, algorithms (Qian, 1999; Duchi et al., 2011; Kingma & Ba, 2014; Hinton, 2012; Griffiths & Steyvers, 2004), and dataset (Krizhevsky & Hinton, 2009; LeCun, 1998; Harper & Konstan, 2016; Rennie) in our study. η denotes learning rate (empirically chosen), β_1, β_2 are optimization hyperparameters (using common default values). α, β in LDA are Dirichlet priors for document topic and word topic random variables, respectively.

Convolutional Neural Networks (CNNs) have been a strong focus of large-scale training, both under synchronous (Goyal et al., 2017; Cui et al., 2016; Coates et al., 2013) and non-synchronous (Chilimbi et al., 2014; Dean et al., 2012; Chen et al., 2016; Hadjis et al., 2016) training. We consider residual networks with $6n + 2$ weight layers (He et al., 2016). The networks consist of 3 groups of n residual blocks, with 16, 32, and 64 feature maps in each group, respectively, followed by a global pooling layer and a softmax layer. The residual blocks have the same construction as in (He et al., 2016). We measure the model quality using test accuracy. For simplicity, we omit data augmentation in our experiments.

Deep Neural Networks (DNNs) are neural networks composed of fully connected layers. Our DNNs have 1 to 6 hidden layers, with 256 neurons in each layer, followed by a softmax layer. We use rectified linear units (ReLU) for nonlinearity after each hidden layer (Nair & Hinton, 2010). **Multi-class Logistic Regression (MLR)** is the special case of DNN with 0 hidden layers. We measure the model quality using test accuracy.

Matrix factorization (MF) is commonly used in recommender systems and have been implemented at scale (Yun et al., 2013; Low et al., 2012; Cui et al., 2014; Zhang & Kwok, 2014; Kim et al., 2016; Ho et al., 2013; Kumar et al., 2014). Let $D \in \mathbb{R}^{M \times N}$ be a partially filled matrix, MF factorizes D into two factor matrices $L \in \mathbb{R}^{M \times r}$ and $R \in \mathbb{R}^{N \times r}$ ($r \ll \min(M, N)$ is the user-defined rank). The ℓ_2-penalized optimization problem is: $\min_{L,R} \frac{1}{|D_{obs}|} \left\{ \sum_{(i,j) \in D_{obs}} ||D_{ij} - \sum_{k=1}^{K} L_{ik} R_{kj}||^2 + \lambda(||L||_F^2 + ||R||_F^2) \right\}$ where $|| \cdot ||_F$ is the Frobenius norm and λ is the regularization parameter. We partition observations D to workers while treating L, R as shared model parameters. We optimize MF via SGD, and measure model quality by training loss defined by the objective function above.

Latent Dirichlet Allocation (LDA) is an unsupervised method to uncover hidden semantics ("topics") from a group of documents, each represented as a bag of tokens. In LDA each token w_{ij} (j-th token in the i-th document) is assigned with a latent topic z_{ij} from totally K topics. We use Gibbs sampling to infer the topic assignments z_{ij}. The Gibbs sampling step involves three sets of parameters, known as sufficient statistics: (1) document-topic vector $\theta_i \in \mathbb{R}^K$ where θ_{ik} the number of topic assignments within document i to topic $k = 1...K$; (2) word-topic vector $\phi_w \in \mathbb{R}^K$ where ϕ_{wk} is the number of topic assignments to topic $k = 1, ..., K$ for word (vocabulary) w across all documents; (3) $\tilde{\phi} \in \mathbb{R}^K$ where $\tilde{\phi}_k = \sum_{w=1}^{W} \phi_{wk}$ is the number of tokens in the corpus assigned to topic k. The corpus (w_{ij}, z_{ij}) is partitioned to workers, while ϕ_w and $\tilde{\phi}$ are shared model parameters.

We measure the model quality using log likelihood. LDA has been scaled under non-synchronous execution (Ahmed et al., 2012; Low et al., 2012; Yuan et al., 2015) with great success.

Variational Autoencoder (VAE) is commonly optimized by black box variational inference, which can be considered as a hybrid of optimization and sampling methods. The inputs to VAE training include two sources of stochasticity: the data sampling $\boldsymbol{x}$ and samples of random variable ϵ. We measure the model quality by test loss. We use DNNs with 1∼3 layers as the encoders and decoders in VAE, in which each layer has 256 units furnished with rectified linear function for non-linearity. The model quality is measured by the training objective value, assuming continuous input $\boldsymbol{x}$ and isotropic Gaussian prior $p(\boldsymbol{z}) \sim \mathcal{N}(0, \boldsymbol{I})$.

4 Experiments

We use batch size 32 for CNNs, DNNs, MLR, and VAEs[3]. For MF, we use batch size of 25000 samples, which is 2.5% of the MovieLens dataset (1M samples). We study staleness up to $s = 50$ on 8 workers, which means model caches can miss updates up to 8.75 data passes. For LDA we use $\frac{D}{10P}$ as the batch size, where D is the number of documents and P is the number of workers. We study staleness up to $s = 20$, which means model caches can miss updates up to 2 data passes. We measure time in terms of the amount of work performed, such as the number of batches processed.

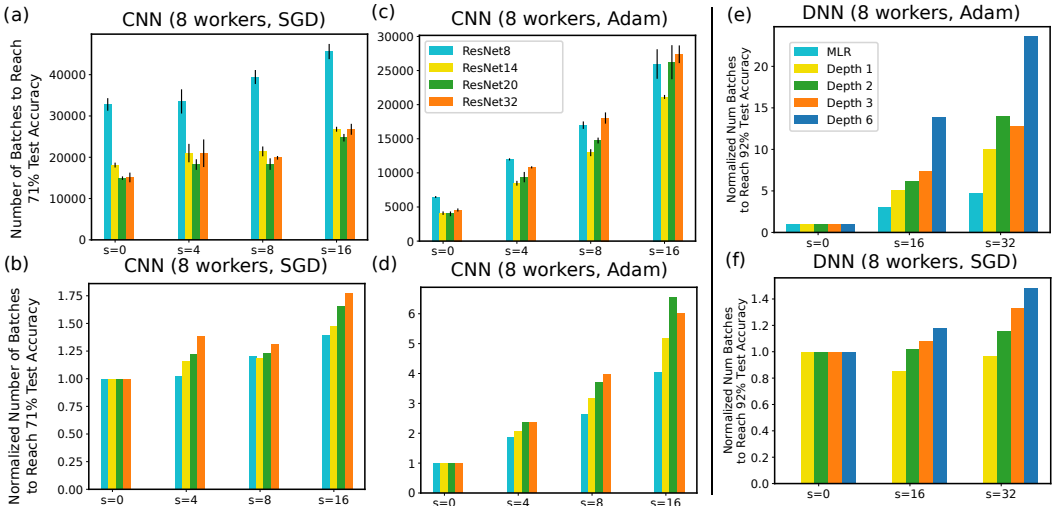

Figure 1: **(a)(c)** The number of batches to reach 71% test accuracy on CIFAR10 for 4 variants of ResNet with varying staleness, using 8 workers and SGD / Adam. The mean and standard deviation are calculated over 3 randomized runs. **(b)(d)** The same metrics as (a)(c), but each model is normalized by the value under staleness 0 ($s = 0$), respectively. **(e)(f)** The number of batches to reach 92% accuracy for MLR and DNN with varying depths, normalized by the value under staleness 0. MLR with SGD did not converge within the experiment horizon (77824 batches) and is thus not shown in (f).

Convergence Slowdown. Perhaps the most prominent effect of staleness on ML algorithms is the slowdown in convergence, evident throughout the experiments. Fig. 1 shows the number of batches needed to reach the desired model quality for CNNs and DNNs/MLR with varying network depths and different staleness ($s = 0, ..., 16$). Fig. 1(b)(d) show that convergence under higher level of staleness requires more batches to be processed in order to reach the same model quality. This additional work can potentially be quite substantial, such as in Fig. 1(d) where it takes up to 6x more batches compared with settings without staleness ($s = 0$). It is also worth pointing out that while there can be a substantial slowdown in convergence, the optimization still reaches desirable models under most cases in our experiments. When staleness is geometrically distributed (Fig. 4(c)), we observe similar patterns of convergence slowdown.

We are not aware of any prior work reporting slowdown as high as observed here. This finding has important ramifications for distributed ML. Usually, the moderate amount of workload increases due to parallelization errors can be compensated by the additional computation resources and higher

[3]Non-synchronous execution allows us to use small batch sizes, eschewing the potential generalization problem with large batch SGD (Keskar et al., 2016; Masters & Luschi, 2018).

system throughput in the distributed execution. However, it may be difficult to justify spending large amount of resources for a distributed implementation if the statistical penalty is too high, which should be avoided (e.g., by staleness minimization system designs or synchronous execution).

Model Complexity. Fig. 1 also reveals that the impact of staleness can depend on ML parameters, such as the depths of the networks. Overall we observe that staleness impacts deeper networks more than shallower ones. This holds true for SGD, Adam, Momentum, RMSProp, Adagrad (Fig. 2), and other optimization schemes, and generalizes to other numbers of workers (see Appendix).

This is perhaps not surprising, given the fact that deeper models pose more optimization challenges even under the sequential settings (Glorot & Bengio, 2010; He et al., 2016), though we point out that existing literature does not explicitly consider model complexity as a factor in distributed ML (Lian et al., 2015; Goyal et al., 2017). Our results suggest that the staleness level acceptable in distributed training can depend strongly on the complexity of the model. For sufficiently complex models it may be more advantageous to eliminate staleness altogether and use synchronous training.

Algorithms' Sensitivity to Staleness. Staleness has uneven impacts on different SGD variants. Fig. 2 shows the amount of work (measured in the number of batches) to reach the desired model quality for five SGD variants. Fig. 2(d)(e)(f) reveals that while staleness generally increases the number of batches needed to reach the target test accuracy, the increase can be drastic for certain algorithms, such as Adam, Momentum, and RMSProp. RMSProp, in particular, fails to converge to the desired test accuracy under several settings. On the other hand, SGD and Adagrad appear to be robust to the staleness, with the total workload <2x of non-stale case ($s = 0$) even under high staleness ($s = 16$)[4]. In fact, in some cases, the convergence is accelerated by staleness, such as Adagrad on 1 worker under $s = 16$. This may be attributed to the implicit momentum created by staleness (Mitliagkas et al., 2016) and the aggressive learning rate shrinking schedule in Adagrad.

Our finding is consistent with the fact that, to our knowledge, all existing successful cases applying non-synchronous training to deep neural networks use SGD (Dean et al., 2012; Chilimbi et al., 2014). In contrast, works reporting subpar performance from non-synchronous training all use variants SGD, such as RMSProp with momentum (Chen et al., 2016) and momentum (Cui et al., 2016). Our results suggest that these different outcomes may be primarily driven by the choice of optimization algorithms, leading to the seemingly contradictory reports of whether non-synchronous execution is advantageous over synchronous ones.

Effects of More Workers. The impact of staleness is amplified by the number of workers. In the case of MF, Fig. 3(b) shows that the convergence slowdown in terms of the number of batches (normalized by the convergence for $s = 0$) on 8 workers is more than twice of the slowdown on 4 workers. For example, in Fig. 3(b) the slowdown at $s = 15$ is $\sim$3.4, but the slowdown at the same staleness level on 8 workers is $\sim$8.2. Similar observations can be made for CNNs (Fig. 3). This can be explained by the fact that additional workers amplifies the effect of staleness by (1) generating updates that will be subject to delays, and (2) missing updates from other workers that are subject to delays.

LDA. Fig. 3(c)(d) show the convergence curves of LDA with different staleness levels for two settings varying on the number of workers and topics. Unlike the convergence curves for SGD-based algorithms (see Appendix), the convergence curves of Gibbs sampling are highly smooth, even under high staleness and a large number of workers. This can be attributed to the structure of log likelihood objective function (Griffiths & Steyvers, 2004). Since in each sampling step we only update the count statistics based on a portion of the corpus, the objective value will generally change smoothly.

Staleness levels under a certain threshold ($s \leq 10$) lead to convergence, following indistinguishable log likelihood trajectories, regardless of the number of topics ($K = 10, 100$) or the number of workers (2–16 workers, see Appendix). Also, there is very minimal variance in those trajectories. However, for staleness beyond a certain level ($s \geq 15$), Gibbs sampling does not converge to a fixed point. The convergence trajectories are distinct and are sensitive to the number of topics and the number of workers. There appears to be a "phase transition" at a certain staleness level that creates two distinct phases of convergence behaviors[5]. We believe this is the first report of a staleness-induced failure case for LDA Gibbs sampling.

[4]Note that $s = 0$ execution treats each worker's update as separate updates instead of one large batch in other synchronous systems.

[5]We leave the investigation into this distinct phenomenon as future work.

VAE In Fig. 3(e)(f), VAEs exhibit a much higher sensitivity to staleness compared with DNNs (Fig. 1(e)(f)). This is the case even considering that VAE with depth 3 has 6 weight layers, which has a comparable number of model parameters and network architecture to DNNs with 6 layers. We hypothesize that this is caused by the additional source of stochasticity from the sampling procedure, in addition to the data sampling process.

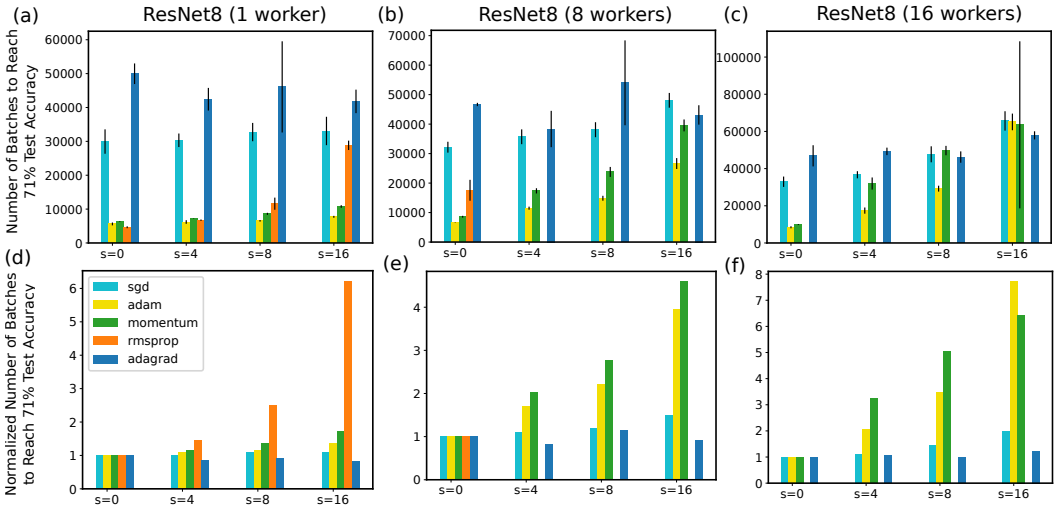

Figure 2: (a)(b)(c) The number of batches to reach 71% test accuracy on 1, 8, 16 workers with staleness $s = 0, ..., 16$ using ResNet8. We consider 5 variants of SGD: SGD, Adam, Momentum, RMSProp, and Adagrad. (d)(e)(f) show the same metric but each algorithm is normalized by the value under staleness 0 ($s = 0$), respectively. Under certain settings RMSProp does not converge to the desired model quality within the experiment horizon and is thus omitted.

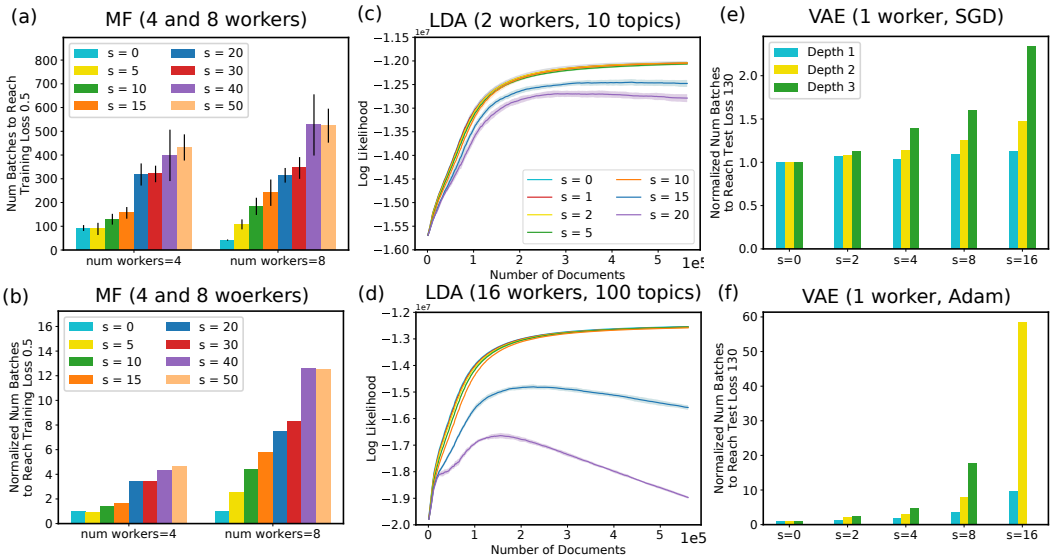

Figure 3: **(a)** The number of batches to reach training loss of 0.5 for Matrix Factorization (MF). **(b)** shows the same metric in (a) but normalized by the values of staleness 0 of each worker setting, respectively (4 and 8 workers). **(c)(d)** Convergence of LDA log likelihood using 10 and 100 topics under staleness levels $s = 0, ..., 20$, with 2 and 16 workers. The convergence is recorded against the number of documents processed by Gibbs sampling. The shaded regions are 1 standard deviation around the means (solid lines) based on 5 randomized runs. (e)(f) The number of batches to reach test loss 130 by Variational Autoencoders (VAEs) on 1 worker, under staleness $s = 0, ..., 16$. We consider VAEs with depth 1, 2, and 3 (the number of layers in the encoder and the decoder networks, separately). The numbers of batches are normalized by $s = 0$ for each VAE depth, respectively. Configurations that do not converge to the desired test loss are omitted in the graph, such as Adam optimization for VAE with depth 3 and $s = 16$.

5 GRADIENT COHERENCE AND CONVERGENCE OF ASYNCHRONOUS SGD

We now provide theoretical insight into the effect of staleness on the observed convergence slowdown. We focus on the challenging asynchronous SGD (Async-SGD) case, which characterizes the neural network models, among others. Consider the following nonconvex optimization problem

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

where f_i corresponds to the loss on the i-th data sample, and the objective function is assumed to satisfy the following standard conditions:

Assumption 1. *The objective function F in the problem (P) satisfies:*

1. *Function F is continuously differentiable and bounded below, i.e., $\inf_{\mathbf{x}\in\mathbb{R}^d} F(\mathbf{x}) > -\infty$;*
2. *The gradient of F is L-Lipschitz continuous.*

Notice that we allow F to be nonconvex. We apply the Async-SGD to solve the problem (P). Let $\xi(k)$ be the mini-batch of data indices sampled from $\{1,\ldots,n\}$ uniformly at random by the algorithm at iteration k, and $|\xi(k)|$ is the mini-batch size. Denote mini-batch gradient as $\nabla f_{\xi(k)}(\mathbf{x}_k) := \sum_{i\in\xi(k)} \nabla f_i(\mathbf{x}_k)$. Then, the update rule of Async-SGD can be written as

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \tfrac{\eta_k}{|\xi(\tau_k)|}\nabla f_{\xi(\tau_k)}(\mathbf{x}_{\tau_k}), \tag{Async-SGD}$$

where η_k corresponds to the stepsize, τ_k denotes the delayed clock and the maximum staleness is assumed to be bounded by s. This implies that $k-s+1 \le \tau_k \le k$.

The optimization dynamics of Async-SGD is complex due to the nonconvexity and the uncertainty of the delayed updates. Interestingly, we find that the following notion of gradient coherence provides insights toward understanding the convergence property of Async-SGD.

Definition 1 (Gradient coherence)**.** *The gradient coherence at iteration k is defined as*

$$\mu_k := \min_{k-s+1\le t\le k} \tfrac{\langle \nabla F(\mathbf{x}_k), \nabla F(\mathbf{x}_t)\rangle}{\|\nabla F(\mathbf{x}_k)\|^2}.$$

Parameter μ_k captures the minimum coherence between the current gradient $\nabla F(\mathbf{x}_k)$ and the gradients along the past s iterations. Intuitively, if μ_k is positive, then the direction of the current gradient is well aligned to those of the past gradients. In this case, the convergence property induced by using delayed stochastic gradients is close to that induced by using synchronous stochastic gradients. Note that μ_k is easy to compute empirically during the course of asynchronous optimization[6]. Empirically we observe that $\mu_k > 0$ through most of the optimization path, especially when the staleness is minimized in practice by system optimization (Fig. 4). Our theory can be readily adapted to account for a limited amount of negative μ_k (see Appendix), but our primary interest is to provide a quantity that is (1) easy to compute empirically during the course of optimization, and (2) informative for the impact of staleness and can potentially be used to control synchronization levels. We now characterize the convergence property of Async-SGD.

Theorem 1. *Let Assumption 1 hold. Suppose for some $\mu > 0$, the gradient coherence satisfies $\mu_k \ge \mu$ for all k and the variance of the stochastic gradients is bounded by $\sigma^2 > 0$. Choose stepsize $\eta_k = \frac{\mu}{sL\sqrt{k}}$. Then, the iterates generated by the Async-SGD satisfy*

$$\min_{0\le k\le T} \mathbb{E}\|\nabla F(\mathbf{x}_k)\|^2 \le \left(\tfrac{sL(F(\mathbf{x}_0)-\inf_{\mathbf{x}} F(\mathbf{x}))}{\mu^2} + \tfrac{\sigma^2 \log T}{s}\right)\tfrac{1}{\sqrt{T}}. \tag{1}$$

We refer readers to Appendix for the the proof. Theorem 1 characterizes several theoretical aspects of Async-SGD. First, the choice of the stepsize $\eta_k = \frac{\mu}{sL\sqrt{k}}$ is adapted to both the maximum staleness and the gradient coherence. Intuitively, if the system encounters a larger staleness, then a smaller stepsize should be used to compensate the negative effect. On the other hand, the stepsize can be accordingly enlarged if the gradient coherence along the iterates turns out to be high. In this case, the direction of the gradient barely changes along the past several iterations, and a more aggressive stepsize can be adopted. In summary, the choice of stepsize should trade-off between the effects caused by both the staleness and the gradient coherence.

[6] It can be approximated by storing a pre-selected batch of data on a worker. The worker just needs to compute gradient every T minibatches to obtain approximate $\nabla F(\mathbf{x}_k)$, $\nabla F(\mathbf{x}_t)$ in Definition 1.

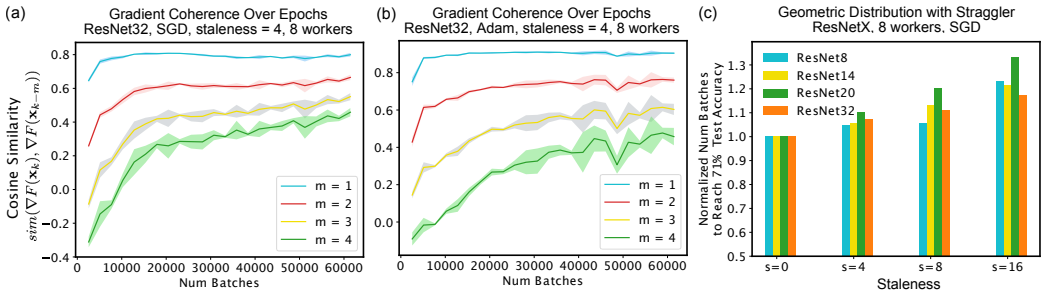

Figure 4: (a)(b) Cosine similarity between the gradient at the k-th iteration $\nabla F(\mathbf{x}_k)$, and the gradient m steps prior $\nabla F(\mathbf{x}_{k-m})$, over the course of convergence for ResNet32 on CIFAR10 optimized by SGD (a) and Adam (b) under staleness $s = 4$ on 8 workers with parameters in Table 1. Shaded region is 1 standard deviation over 3 runs. For computational efficiency, we approximate the full gradient $\nabla F(\mathbf{x}_k)$ by gradients on a fixed set of 1000 training samples D_{fixed} and use $\nabla_{D_{fixed}} F(\mathbf{x}_k)$. (c) The number of batches to reach 71% test accuracy on CIFAR10 for ResNet8-32 using 8 workers and SGD under geometric delay distribution (details in Appendix).

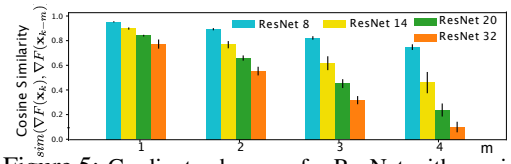

Figure 5: Gradient coherence for ResNet with varying depths optimized by SGD using 8 workers. The x-axis m is defined in Fig. 4

Furthermore, Theorem 1 shows that the minimum gradient norm decays at the rate $\mathcal{O}(\frac{\log T}{\sqrt{T}})$, implying that the Async-SGD converges to a stationary point provided a positive gradient coherence, which we observe empirically in the sequel. On the other hand, the bound in Eq. (1) captures the trade-off between the maximum staleness s and the gradient coherence μ. Specifically, minimizing the right hand side of Eq. (1) with regard to the maximum staleness s yields the optimal choice $s^* = \sigma\mu\sqrt{\frac{\log T}{L(F(\mathbf{x}_0) - \inf_{\mathbf{x}} F(\mathbf{x}))}}$, i.e., a larger staleness is allowed if the gradients remain to be highly coherent along the past iterates.

Empirical Observations. Theorem 1 suggests that more coherent gradients along the optimization paths can be advantageous under non-synchronous execution. Fig. 4 shows the cosine similarity $sim(\boldsymbol{a}, \boldsymbol{b}) := \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{\|\boldsymbol{a}\| \|\boldsymbol{b}\|}$ between gradients along the convergence path for CNNs and DNNs[7]. We observe the followings: (1) Cosine similarity improves over the course of convergence (Fig. 4(a)(b)). Except the highest staleness during the early phase of convergence, cosine similarity remains positive. In practice the staleness experienced during run time can be limited to small staleness (Dai et al., 2015), which minimizes the likelihood of negative gradient coherence during the early phase. (2) Fig. 5 shows that Cosine similarity decreases with increasing model complexity for CNN models. This is consistent with the convergence difficulty encountered in deeper models (Fig. 1).

6 Discussion and Conclusion

In this work, we study the convergence behaviors under delayed updates for a wide array of models and algorithms. Our extensive experiments reveal that staleness appears to be a key governing parameter in learning. Overall staleness slows down the convergence, and under high staleness levels the convergence can progress very slowly or fail. The effects of staleness are highly problem dependent, influenced by model complexity, choice of the algorithms, the number of workers, and the model itself, among others. Our empirical findings inspire new analyses of non-convex optimization under asynchrony based on gradient coherence, matching the existing rate of $\mathcal{O}(1/\sqrt{T})$.

Our findings have clear implications for distributed ML. To achieve actual speed-up in absolute convergence, any distributed ML system needs to overcome the slowdown from staleness, and carefully trade off between system throughput gains and statistical penalties. Many ML methods indeed demonstrate certain robustness against low staleness, which should offer opportunities for system optimization. Our results support the broader observation that existing successful non-synchronous systems generally keep staleness low and use algorithms efficient under staleness (Li et al., 2014a; Ho et al., 2013).

[7]Cosine similarity is closely related to the coherence measure in Definition 1.

REFERENCES

Amr Ahmed, Mohamed Aly, Joseph Gonzalez, Shravan Narayanamurthy, and Alexander J. Smola. Scalable inference in latent variable models. In *WSDM*, pp. 123–132, 2012.

Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous SGD. *ArXiv: 1604.00981*, 2016.

Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system. In *Proc. USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 571–582, Broomfield, CO, October 2014.

Adam Coates, Brody Huval, Tao Wang, David Wu, Bryan Catanzaro, and Ng Andrew. Deep learning with cots hpc systems. In *Proc. International Conference on Machine Learning (ICML)*, pp. 1337–1345, 2013.

Henggang Cui, James Cipar, Qirong Ho, Jin Kyu Kim, Seunghak Lee, Abhimanu Kumar, Jinliang Wei, Wei Dai, Gregory R. Ganger, Phillip B. Gibbons, Garth A. Gibson, and Eric P. Xing. Exploiting bounded staleness to speed up big data analytics. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pp. 37–48, Philadelphia, PA, June 2014. USENIX Association.

Henggang Cui, Hao Zhang, Gregory R Ganger, Phillip B Gibbons, and Eric P Xing. Geeps: Scalable deep learning on distributed gpus with a gpu-specialized parameter server. In *Proceedings of the Eleventh European Conference on Computer Systems*, pp. 4. ACM, 2016.

Wei Dai, Abhimanu Kumar, Jinliang Wei, Qirong Ho, Garth Gibson, and Eric P. Xing. Analysis of high-performance distributed ml at scale through parameter server consistency models. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, 2015.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pp. 1223–1231, 2012.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proc. International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 249–256, 2010.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training imagenet in 1 hour. *ArXiv: 1706.02677*, 2017.

Thomas L. Griffiths and Mark Steyvers. Finding scientific topics. *PNAS*, 101(suppl. 1):5228–5235, 2004.

Stefan Hadjis, Ce Zhang, Ioannis Mitliagkas, Dan Iter, and Christopher Ré. Omnivore: An optimizer for multi-device deep learning on cpus and gpus. *arXiv preprint arXiv:1606.04487*, 2016.

F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Geoffrey Hinton. Neural networks for machine learning. `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`, 2012.

Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B. Gibbons, Garth A. Gibson, Greg Ganger, and Eric Xing. More effective distributed ml via a stale synchronous parallel parameter server. In *Advances in Neural Information Processing Systems (NIPS) 26*, pp. 1223–1231. 2013.

Michael I Jordan et al. On statistics, computation and scalability. *Bernoulli*, 19(4):1378–1390, 2013.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *ArXiv: 1609.04836*, 2016.

Jin Kyu Kim, Qirong Ho, Seunghak Lee, Xun Zheng, Wei Dai, Garth A Gibson, and Eric P Xing. Strads: a distributed framework for scheduled model parallel machine learning. In *Proceedings of the Eleventh European Conference on Computer Systems*, pp. 5. ACM, 2016.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

Abhimanu Kumar, Alex Beutel, Qirong Ho, and Eric P Xing. Fugue: Slow-worker-agnostic distributed learning for big models on big data. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, pp. 531–539, 2014.

John Langford, Alexander J Smola, and Martin Zinkevich. Slow learners are fast. In *Advances in Neural Information Processing Systems*, pp. 2331–2339, 2009.

Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *OSDI*, volume 14, pp. 583–598, 2014a.

Mu Li, David G Andersen, Alex J Smola, and Kai Yu. Communication efficient distributed machine learning with the parameter server. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, pp. 19–27, 2014b.

Xiangru Lian, Yijun Huang, Yuncheng Li, and Ji Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pp. 2737–2745, 2015.

Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. Distributed graphlab: A framework for machine learning and data mining in the cloud. *PVLDB*, 2012.

Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. *ArXiv: 1804.07612*, 2018.

Brendan McMahan and Matthew Streeter. Delay-tolerant algorithms for asynchronous distributed online learning. In *Advances in Neural Information Processing Systems*, pp. 2915–2923, 2014.

Ioannis Mitliagkas, Ce Zhang, Stefan Hadjis, and Christopher Ré. Asynchrony begets momentum, with an application to deep learning. In *Communication, Control, and Computing (Allerton), 2016 54th Annual Allerton Conference on*, pp. 997–1004. IEEE, 2016.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, pp. 807–814, 2010.

Willie Neiswanger, Chong Wang, and Eric Xing. Asymptotically exact, embarrassingly parallel mcmc. *arXiv preprint arXiv:1311.4780*, 2013.

Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1): 145–151, 1999.

Benjamin Recht, Christopher Re, Stephen J. Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, pp. 693–701, 2011.

Jason Rennie. 20 newsgroups. http://qwone.com/ jason/20Newsgroups/.

Steven L Scott, Alexander W Blocker, Fernando V Bonassi, Hugh A Chipman, Edward I George, and Robert E McCulloch. Bayes and big data: The consensus monte carlo algorithm. *International Journal of Management Science and Engineering Management*, 11(2):78–88, 2016.

Alexander Smola and Shravan Narayanamurthy. An architecture for parallel topic models. *Proc. VLDB Endow.*, 3(1-2):703–710, September 2010. ISSN 2150-8097.

Jinliang Wei, Wei Dai, Aurick Qiao, Qirong Ho, Henggang Cui, Gregory R Ganger, Phillip B Gibbons, Garth A Gibson, and Eric P Xing. Managed communication and consistency for fast data-parallel iterative analytics. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, pp. 381–394. ACM, 2015.

Jinhui Yuan, Fei Gao, Qirong Ho, Wei Dai, Jinliang Wei, Xun Zheng, Eric Po Xing, Tie-Yan Liu, and Wei-Ying Ma. Lightlda: Big topic models on modest computer clusters. In *Proceedings of the 24th International Conference on World Wide Web*, pp. 1351–1361. ACM, 2015.

Hyokun Yun, Hsiang-Fu Yu, Cho-Jui Hsieh, SVN Vishwanathan, and Inderjit Dhillon. NOMAD: Non-locking, stochastic multi-machine algorithm for asynchronous and decentralized matrix completion. *ArXiv: 1312.0193*, 2013.

Ruiliang Zhang and James T. Kwok. Asynchronous distributed admm for consensus optimization. In *ICML*, 2014.

APPENDIX: TOWARD UNDERSTANDING THE IMPACT OF STALENESS IN DISTRIBUTED MACHINE LEARNING

Anonymous authors
Paper under double-blind review

1 PROOF OF THEOREM 1

Theorem 1. *Let Assumption 1 hold. Suppose the gradient coherence μ_k is lower bounded by some $\mu > 0$ for all k and the variance of the stochastic gradients is upper bounded by some $\sigma^2 > 0$. Choose stepsize $\eta_k = \frac{\mu}{sL\sqrt{k}}$. Then, the iterates generated by the Async-SGD satisfy*

$$\min_{0\le k\le T} \mathbb{E}\|\nabla F(\mathbf{x}_k)\|^2 \le \big(\tfrac{sL(F(\mathbf{x}_0)-\inf_{\mathbf{x}} F(\mathbf{x}))}{\mu^2} + \tfrac{\sigma^2 \log T}{s}\big)\tfrac{1}{\sqrt{T}}. \tag{1}$$

Proof. By the L-Lipschitz property of ∇F, we obtain that for all k

$$F(\mathbf{x}_{k+1}) \le F(\mathbf{x}_k) + \langle \mathbf{x}_{k+1}-\mathbf{x}_k, \nabla F(\mathbf{x}_k)\rangle + \frac{L}{2}\|\mathbf{x}_{k+1}-\mathbf{x}_k\|^2 \tag{2}$$

$$= F(\mathbf{x}_k) - \eta_k\langle \nabla f_{\xi(\tau_k)}(\mathbf{x}_{\tau_k}), \nabla F(\mathbf{x}_k)\rangle + \frac{L\eta_k^2}{2}\|\nabla f_{\xi(\tau_k)}(\mathbf{x}_{\tau_k})\|^2. \tag{3}$$

Taking expectation on both sides of the above inequality and note that the variance of the stochastic gradient is bounded by σ^2, we further obtain that

$$\mathbb{E}[F(\mathbf{x}_{k+1})] \le \mathbb{E}[F(\mathbf{x}_k)] - \eta_k\mathbb{E}[\langle \nabla F(\mathbf{x}_{\tau_k}), \nabla F(\mathbf{x}_k)\rangle] + \frac{L\eta_k^2}{2}\mathbb{E}[\|\nabla F(\mathbf{x}_{\tau_k})\|^2 + \sigma^2] \tag{4}$$

$$\le \mathbb{E}[F(\mathbf{x}_k)] - \eta_k\mu_k\mathbb{E}\|\nabla F(\mathbf{x}_k)\|^2 + \frac{L\eta_k^2}{2}\mathbb{E}\|\nabla F(\mathbf{x}_{\tau_k})\|^2 + \frac{\sigma^2 L\eta_k^2}{2} \tag{5}$$

$$\le \mathbb{E}[F(\mathbf{x}_k)] - \eta_k\mu\mathbb{E}\|\nabla F(\mathbf{x}_k)\|^2 + \frac{L\eta_k^2}{2}\sum_{t=k-s+1}^{k}\mathbb{E}\|\nabla F(\mathbf{x}_t)\|^2 + \frac{\sigma^2 L\eta_k^2}{2}. \tag{6}$$

Telescoping the above inequality over k from 0 to T yields that

$$\mathbb{E}[F(\mathbf{x}_{k+1})] - \mathbb{E}[F(\mathbf{x}_0)]$$

$$\le -\sum_{k=0}^{T}\eta_k\mu\mathbb{E}\|\nabla F(\mathbf{x}_k)\|^2 + \frac{L}{2}\sum_{k=0}^{T}\sum_{t=k-s+1}^{k}\eta_k^2\mathbb{E}\|\nabla F(\mathbf{x}_t)\|^2 + \frac{\sigma^2 L}{2}\sum_{k=0}^{T}\eta_k^2 \tag{7}$$

$$\le -\sum_{k=0}^{T}\eta_k\mu\mathbb{E}\|\nabla F(\mathbf{x}_k)\|^2 + \frac{L}{2}\sum_{k=0}^{T}\sum_{t=k-s+1}^{k}\eta_t^2\mathbb{E}\|\nabla F(\mathbf{x}_t)\|^2 + \frac{\sigma^2 L}{2}\sum_{k=0}^{T}\eta_k^2 \tag{8}$$

$$\le -\sum_{k=0}^{T}\eta_k\mu\mathbb{E}\|\nabla F(\mathbf{x}_k)\|^2 + \frac{Ls}{2}\sum_{k=0}^{T}\eta_k^2\mathbb{E}\|\nabla F(\mathbf{x}_k)\|^2 + \frac{\sigma^2 L}{2}\sum_{k=0}^{T}\eta_k^2 \tag{9}$$

$$= \sum_{k=0}^{T}\big(\frac{Ls\eta_k^2}{2} - \eta_k\mu\big)\mathbb{E}\|\nabla F(\mathbf{x}_k)\|^2 + \frac{\sigma^2 L}{2}\sum_{k=0}^{T}\eta_k^2. \tag{10}$$

Rearranging the above inequality and note that $F(\mathbf{x}_{k+1}) > \inf_{\mathbf{x}} F(\mathbf{x}) > -\infty$, we further obtain that

$$\sum_{k=0}^{T}\big(\eta_k\mu - \frac{Ls\eta_k^2}{2}\big)\mathbb{E}\|\nabla F(\mathbf{x}_k)\|^2 \le \big(F(\mathbf{x}_0) - \inf_{\mathbf{x}} F(\mathbf{x})\big) + \frac{\sigma^2 L}{2}\sum_{k=0}^{T}\eta_k^2. \tag{11}$$

Note that the choice of stepsize guarantees that $\eta_k \mu - \frac{Ls\eta_k^2}{2} > 0$ for all k. Thus, we conclude that

$$\min_{0 \le k \le T} \mathbb{E}\|\nabla F(\mathbf{x}_k)\|^2 \le \frac{\big(F(\mathbf{x}_0) - \inf_{\mathbf{x}} F(\mathbf{x})\big) + \frac{\sigma^2 L}{2} \sum_{k=0}^{T} \eta_k^2}{\sum_{k=0}^{T} \big(\eta_k \mu - \frac{Ls\eta_k^2}{2}\big)} \tag{12}$$

$$\le \frac{2\big(F(\mathbf{x}_0) - \inf_{\mathbf{x}} F(\mathbf{x})\big) + \sigma^2 L \sum_{k=0}^{T} \eta_k^2}{\sum_{k=0}^{T} \eta_k \mu}, \tag{13}$$

where the last inequality uses the fact that $\eta_k \mu - \frac{Ls\eta_k^2}{2} > \frac{\eta_k \mu}{2}$. Substituting the stepsize $\eta_k = \frac{\mu}{sL\sqrt{k}}$ into the above inequality and simplifying, we finally obtain that

$$\min_{0 \le k \le T} \mathbb{E}\|\nabla F(\mathbf{x}_k)\|^2 \le \left(\frac{sL\big(F(\mathbf{x}_0) - \inf_{\mathbf{x}} F(\mathbf{x})\big)}{\mu^2} + \frac{\sigma^2 \log T}{s} \right) \frac{1}{\sqrt{T}}. \tag{14}$$

□

2 Handling Negative Gradient Coherence in Theorem 1

Our assumption of positive gradient coherence (GC) is motivated by strong empirical evidence that GC is positive (Fig. 4(a)(b) in the main text). Contrary to conventional wisdom, GC generally *improves* when reaching convergence for both SGD and Adam. Furthermore, in practice, the effective staleness for any given iteration generally concentrates in low staleness for the non-stragglers (Dai et al., 2015).

When some μ_k are negative at some iterations, in eq. 11 in supplementary we can move the negative terms in $\sum_k \eta_k \mu_k$ to the right hand side and yield a higher upper bound (i.e., slower convergence). This is also consistent with empirical observations that higher staleness lowers GC and slows convergence.

3 Exponential delay distribution.

We consider delays drawn from geometric distribution (GD), which is the discrete version of exponential distribution. For each iterate we randomly select a worker to be the straggler with large mean delay ($p = 0.1$), while all other non-straggler workers have small delays. The non-straggler delay is drawn from GD with p chosen to achieve the same mean delay as in the uniform case (after factoring in straggler) in the main text. The delay is drawn per worker for each iteration, and thus a straggler's outgoing updates to all workers suffer the same delay. Fig. 4(c) in the main text shows the convergence speed under the corresponding staleness s with the same mean delay (though s is not a parameter in GD). It exhibits trends analogous to Fig. 1(b) in main text: staleness slows convergence substantially and overall impacts deeper networks more.

4 Additional Results for DNNs

We present additional results for DNNs. Fig. 1 shows number of batches to reach convergence using 1 hidden layer and 1 worker under varying staleness level. Overall, the effect of batch size is relatively small except in high staleness regime ($s = 32$).

Fig. 2 shows the number of batches to reach convergence, normalized by $s = 0$ case, for 5 variants of SGD using 1 worker. The results are in line with the analyses in the main text. In particular, RMSProp exhibit high variance in batches to convergence (not shown) and thus does not exhibit consistent trend.

Fig. 3 shows the number of batches to convergence under Adam and SGD on 1, 8, 16 simulated workers, respectively normalized by staleness 0's values. The results are consistent with the observations and analyses in the main text.

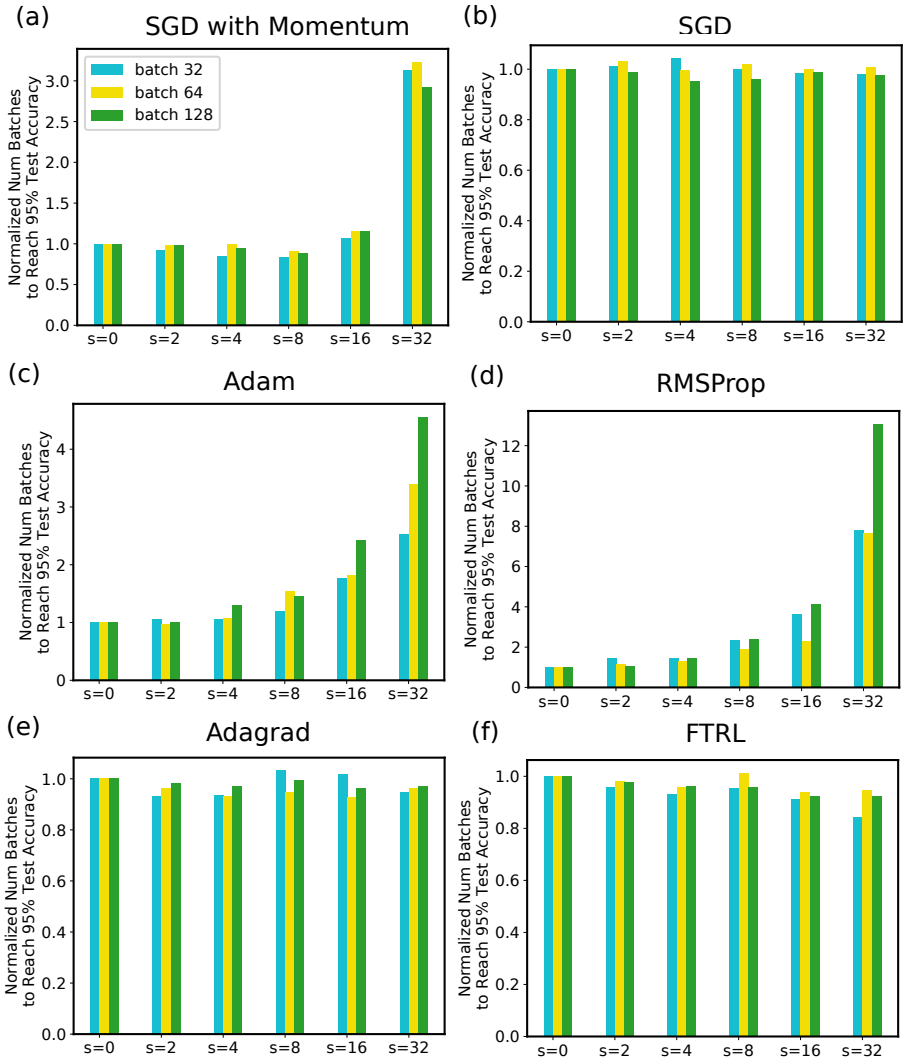

Figure 1: The number of batches to reach 95% test accuracy using 1 hidden layer and 1 worker, respectively normalized by $s = 0$.

5 Additional Results for LDA

We present additional results of LDA under different number of workers and topics in Fig. 4 and Fig. 5. These panels extends Fig. 4(c)(d) in the main manuscript. See the main text for experimental setup and analyses.

6 Additional Results for MF

We show the convergence curves for MF under different number of workers and staleness in Fig. 6. It is evident that higher staleness leads to a higher variance in convergence. Furthermore, the number of workers also affects variance, given the same staleness level. For example, MF with 4 workers incurs very low standard deviation up to staleness 20. In contrast, MF with 8 workers already exhibits a large variance at staleness 15. The amplification of staleness from increasing number of workers is consistent with the discussion in the main text. See the main text for experimental setup and analyses.

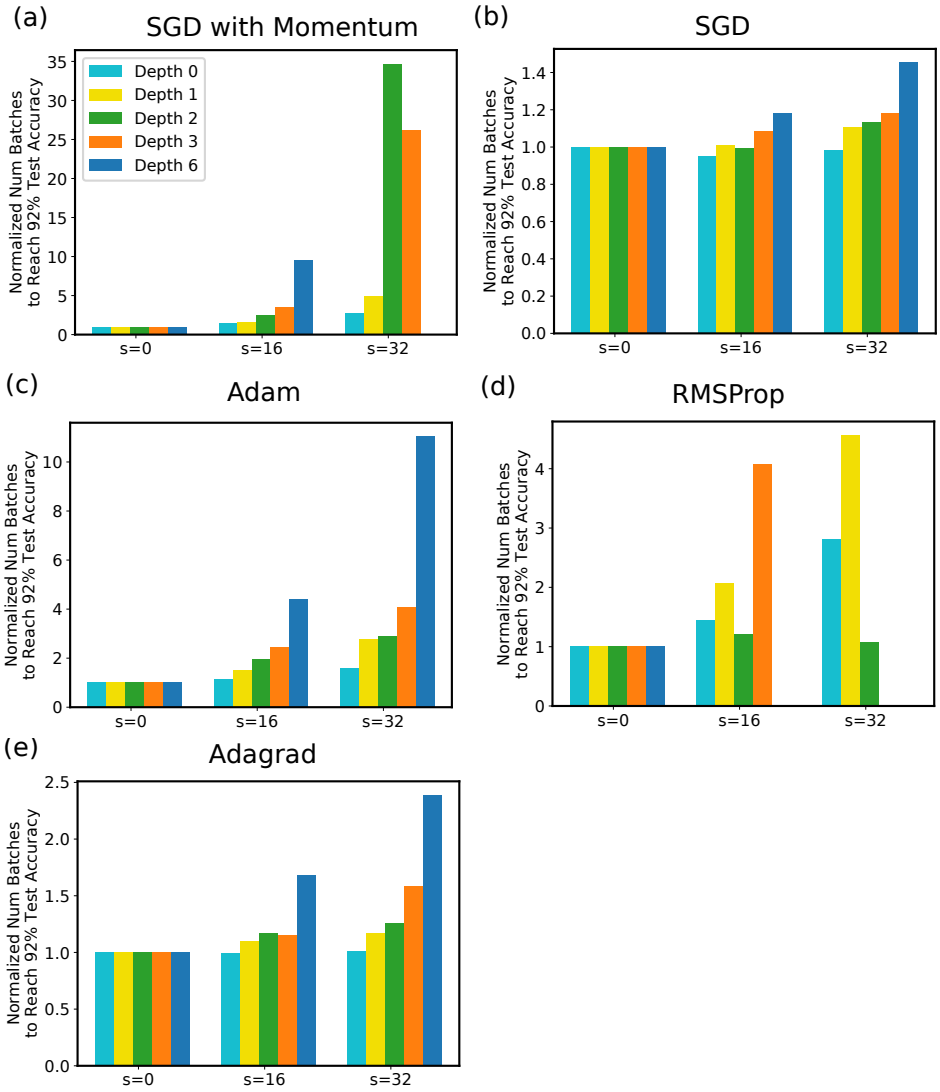

Figure 2: The number of batches to reach 92% test accuracy using DNNs with varying numbers of hidden layers under 1 worker. We consider several variants of SGD algorithms (a)-(e). Note that with depth 0 the model reduces to MLR, which is convex. The numbers are averaged over 5 randomized runs. We omit the result whenever convergence is not achieved within the experiment horizon (77824 batches), such as SGD with momentum at depth 6 and $s = 32$.

7 Additional Results for VAEs

Fig. 7 shows the number of batches to reach test loss 130 by Variational Autoencoders (VAEs) on 1 worker, under staleness 0 to 16 and 4 SGD variants. We consider VAEs with depth 1, 2, and 3 (the number of layers in the encoder and decoder networks). The number of batches are normalized by $s = 0$ for each VAE depth, respectively. See the main text for analyses.

References

Wei Dai, Abhimanu Kumar, Jinliang Wei, Qirong Ho, Garth Gibson, and Eric P. Xing. Analysis of high-performance distributed ml at scale through parameter server consistency models. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, 2015.

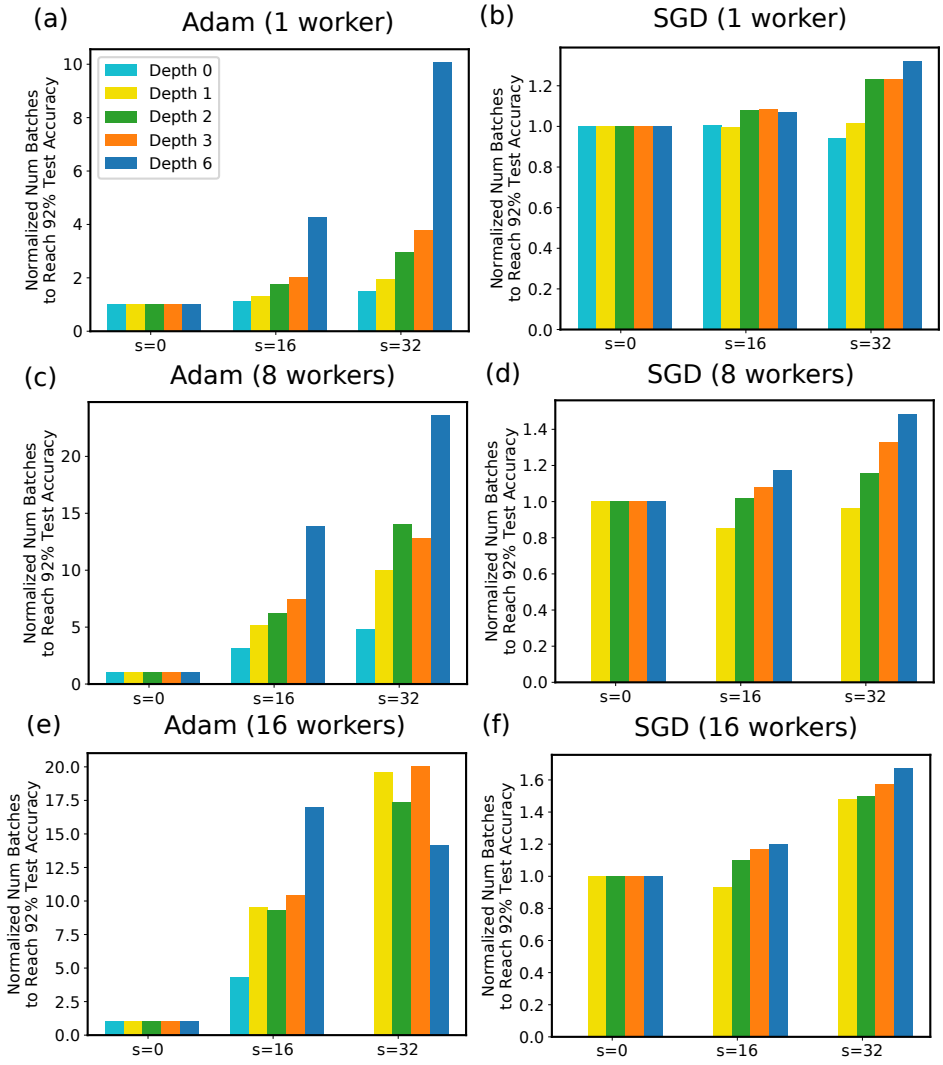

Figure 3: The number of batches to reach 92% test accuracy with Adam and SGD on 1, 8, 16 workers with varying staleness. Each model depth is normalized by the staleness 0's values, respectively. The numbers are average over 5 randomized runs. Depth 0 under SGD with 8 and 16 workers did not converge to target test accuracy within the experiment horizon (77824 batches) for all staleness values, and is thus not shown.

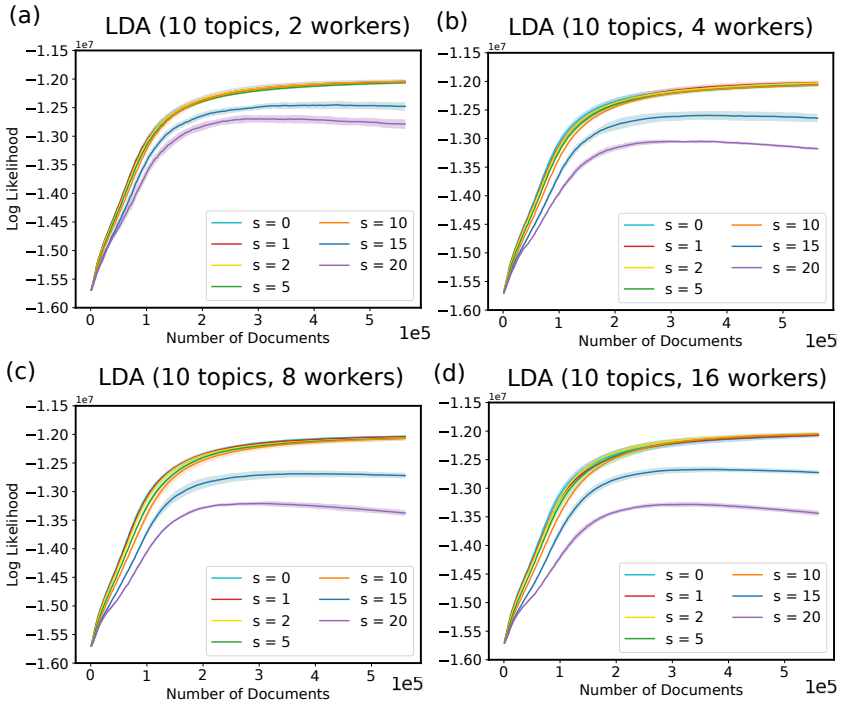

Figure 4: Convergence of LDA log likelihood using 10 topics with respect to the number of documents processed by Gibbs sampling, with varying staleness and number of workers. The shaded regions are 1 standard deviation around the means (solid lines) based on 5 randomized runs.

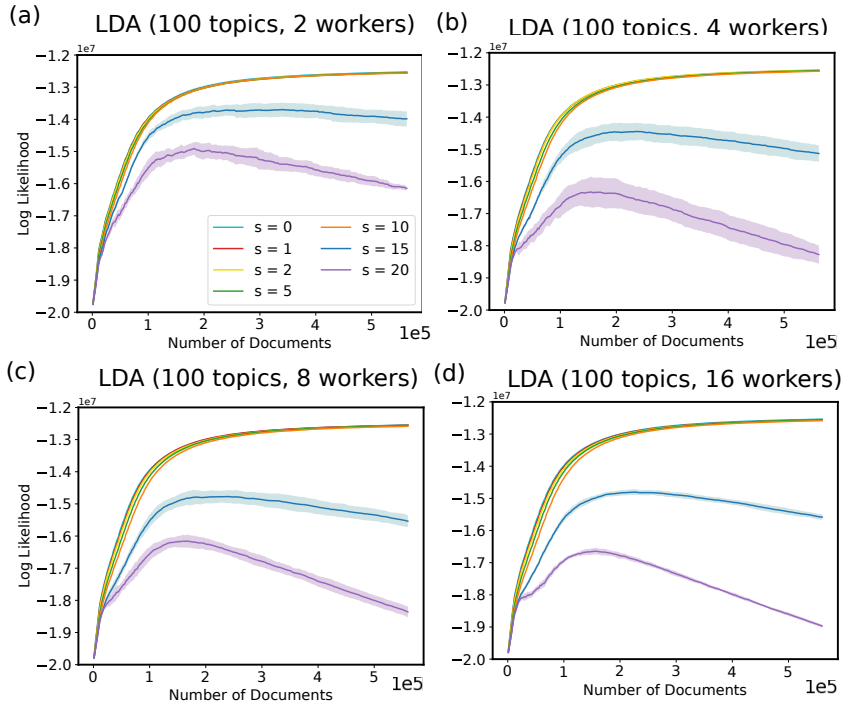

Figure 5: Convergence of LDA log likelihood using 100 topics with respect to the number of documents processed by Gibbs sampling, with varying staleness and the number of workers. The shaded regions are 1 standard deviation around the means (solid lines) based on 5 randomized runs.

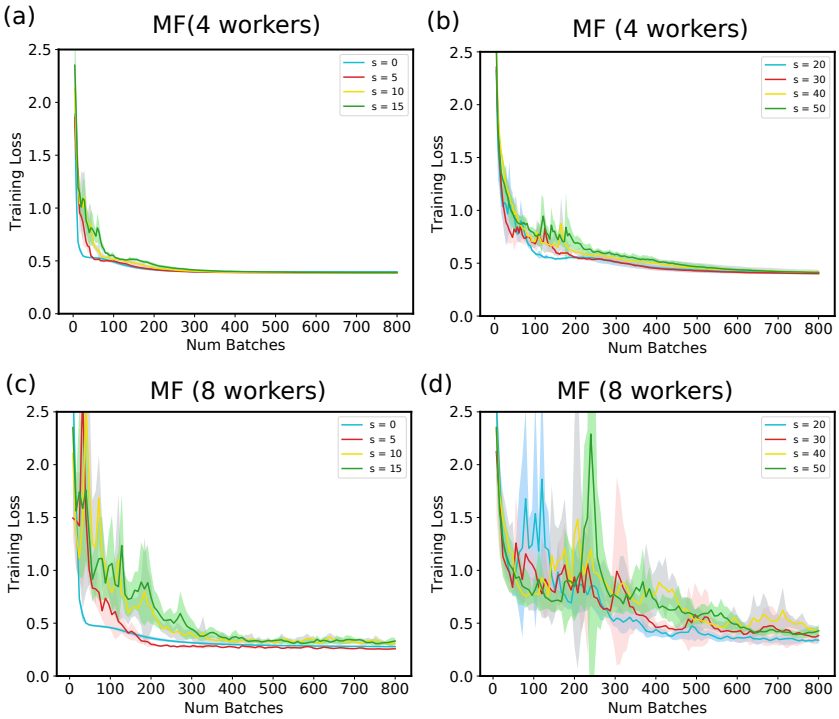

Figure 6: Convergence of Matrix Factorization (MF) using 4 and 8 workers, with staleness ranging from 0 to 50. The x-axis shows the number of batches processed across all workers. Shaded area represents 1 standard deviation around the means (solid curves) computed on 5 randomized runs.

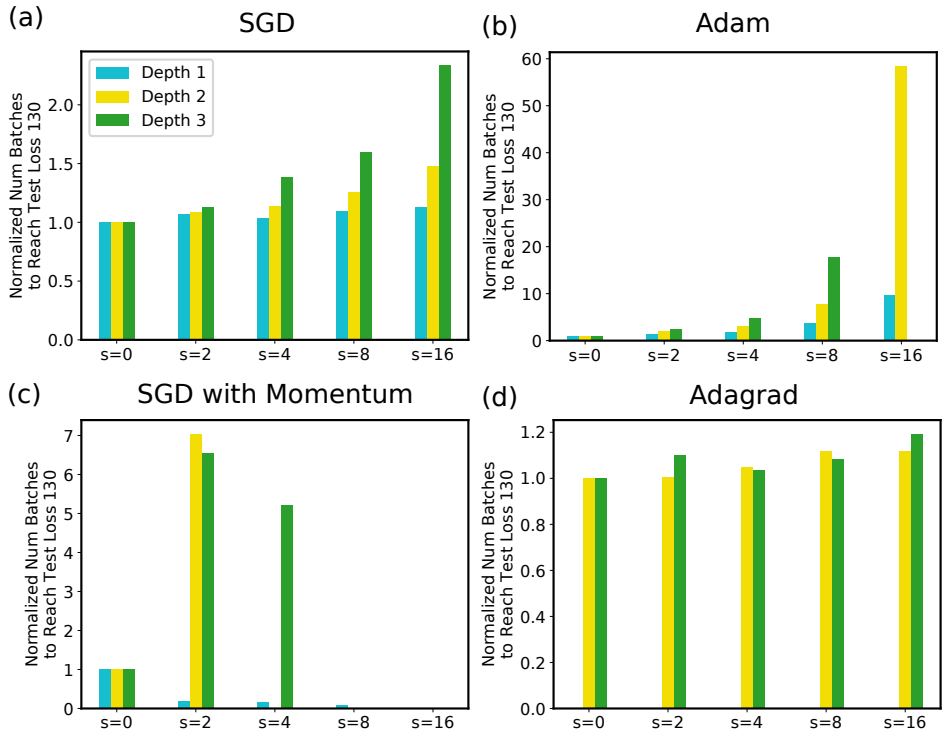

Figure 7: The number of batches to reach test loss 130 by Variational Autoencoders (VAEs) on 1 worker, under staleness 0 to 16. We consider VAEs with depth 1, 2, and 3 (the number of layers in the encoder and the decoder networks). The numbers of batches are normalized by $s = 0$ for each VAE depth, respectively. Configurations that do not converge to the desired test loss are omitted, such as Adam optimization for VAE with depth 3 and $s = 16$.