Adaptive Communication Strategies to Achieve the Best Error-Runtime Trade-off in Local-Update SGD

Jianyu Wang
Carnegie Mellon University
Pittsburgh, PA 15213
jianyuw1@andrew.cmu.edu

Gauri Joshi
Carnegie Mellon University
Pittsburgh, PA 15213
gaurij@andrew.cmu.edu

Abstract

Large-scale machine learning training, in particular distributed stochastic gradient descent, needs to be robust to inherent system variability such as node straggling and random communication delays. This work considers a distributed training framework where each worker node is allowed to perform local model updates and the resulting models are averaged periodically. We analyze the true speed of error convergence with respect to wall-clock time (instead of the number of iterations), and analyze how it is affected by the frequency of averaging. The main contribution is the design of ADACOMM, *an adaptive communication strategy* that starts with infrequent averaging to save communication delay and improve convergence speed, and then increases the communication frequency in order to achieve a low error floor. Rigorous experiments on training deep neural networks show that ADACOMM can take $3\times$ less time than fully synchronous SGD, and still reach the same final training loss.

1 Introduction

Stochastic gradient descent (SGD) is the backbone of state-of-the-art supervised learning, which is revolutionizing inference and decision-making in many diverse applications. Due to the massive training data-sets and neural network architectures used today, it has became imperative to design distributed SGD implementations, where gradient computation and aggregation is parallelized across multiple worker nodes. Although parallelism boosts the amount of data processed per iteration, it exposes SGD to unpredictable node slowdown and communication delays stemming from variability in the computing infrastructure. Thus, there is a critical need to make distributed SGD fast, yet robust to system variability.

Local-Update SGD to Reduce Communication Delays. A popular distributed SGD implementation is the parameter server framework [7, 4, 14, 10, 18] where in each iteration, worker nodes compute gradients on one mini-batch of data and a central parameter server aggregates these gradients (synchronously or asynchronously) and updates the parameter vector $\mathbf{x}$. The constant communication between the parameter server and worker nodes in each iteration can be expensive and slow in bandwidth-limited computed environments. Recently proposed distributed SGD frameworks such as Elastic-averaging [27, 2], Federated Learning [17, 23] and decentralized SGD [15, 12] save this communication cost by allowing worker nodes to perform local updates to the parameter $\mathbf{x}$ instead of just computing gradients. The resulting locally trained models (which are different due to variability in training data across nodes) are periodically averaged through a central server, or via direct inter-worker communication. Periodic averaging has been shown to offer significant speedup in deep neural network training [19, 26, 24, 28, 16].

Error-Runtime Trade-offs in Local-Update SGD. While local updates reduce the communication-delay incurred per iteration, discrepancies between the models can result in an inferior error-

Preprint. Work in progress.

convergence. For example, consider the case of periodic averaging SGD (PASGD) where each of m worker nodes makes τ local updates, and the resulting models are averaged after every τ iterations. In the error-runtime in Figure 1, we observe a trade-off between the convergence speed and the error floor when the number of local updates τ is varied. A larger τ gives a faster initial drop in the training loss but results in a higher error floor. This calls for *adaptive communication strategies* that start with a larger τ and gradually decrease it as the model reaches closer to convergence. Such an adaptive strategy will offer a win-win in the error-runtime trade-off by achieving fast convergence as well as low error floor. In this paper, based on joint runtime and error-convergence analysis, we develop an adaptive communication scheme: ADACOMM. Experiments on training VGG-16 and ResNet-50 deep neural networks and different settings (with/without momentum, fixed/decaying learning rate) show that ADACOMM can give a $3\times$ runtime speed-up and still reach the same low training loss as fully synchronous SGD.

Although we focus on periodic simple-averaging of local models, the insights on error-runtime trade-offs and adaptive communication strategies are directly extendable to other communication-efficient SGD algorithms including Federated Learning [17], Elastic-Averaging [27] and Decentralized averaging [12, 15], as well as synchronous/asynchronous distributed SGD with a central parameter server [7, 4, 9].

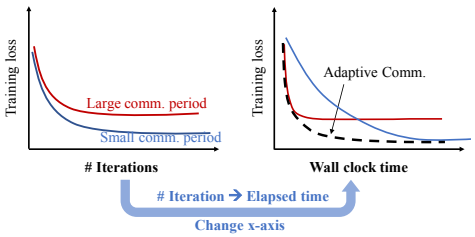


Figure 1: This work departs from the traditional view of considering error-convergence with respect to the number of iterations, and instead considers the true convergence in terms of error versus wall-clock time. Adaptive strategies that start with infrequent model-averaging and increase the communication frequency can achieve the best error-runtime trade-off.

2 Problem Framework

Empirical Risk Minimization via Mini-batch SGD. Suppose the training dataset is denoted by $\mathcal{S} = \{s_1, \ldots, s_N\}$, where s_i represents the i-th labeled data point. Our goal is to minimize the empirical risk function with respect to model parameters denoted by $\mathbf{x} \in \mathbb{R}^d$: $F(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}; s_i)$ where $f(\mathbf{x}; s_i)$ is the composite loss function at the i^{th} data point. In classic mini-batch stochastic gradient descent (SGD) [8], if $\xi_k \subset S$ represents a randomly sampled mini-batch, then updates to the parameter vector $\mathbf{x}$ are performed as: $\mathbf{x}_{k+1} = \mathbf{x}_k - \eta g(\mathbf{x}_k; \xi_k)$ where η denotes the learning rate and the stochastic gradient is defined as $g(\mathbf{x}; \xi) = \frac{1}{|\xi|}\sum_{s_i \in \xi} \nabla f(\mathbf{x}; s_i)$. For simplicity, we will use $g(\mathbf{x}_k)$ instead of $g(\mathbf{x}_k; \xi_k)$ in the rest of the paper. A complete review of convergence properties of serial SGD can be found in [1].

Periodic Averaging SGD (PASGD). In PASGD, all workers start at the same initial point $\mathbf{x}_1$ and perform τ local mini-batch SGD updates. The local models are averaged by a fusion node or by performing an all-node broadcast. The workers then update their local models with the averaged model. Thus, the overall update rule at the i^{th} worker is given by

$$\mathbf{x}_{k+1}^{(i)} = \begin{cases} \frac{1}{m}\sum_{j=1}^{m}[\mathbf{x}_k^{(j)} - \eta g(\mathbf{x}_k^{(j)})], & k \bmod \tau = 0 \\ \mathbf{x}_k^{(i)} - \eta g(\mathbf{x}_k^{(i)}), & \text{otherwise} \end{cases} \tag{1}$$

where $\mathbf{x}_k^{(i)}$ denote the model parameters in the i-th worker after k iterations and τ is defined as the communication period. Note that the iteration index k corresponds to the local iterations, and not the number of averaging steps. When $\tau = 1$, that is, the local models are synchronized after every iteration, periodic averaging SGD is equivalent to fully synchronous SGD.

Local Computation Times and Communication Delay. In order to analyze the effect of τ on the expected runtime per iteration, we consider the following delay model. The time taken by the i^{th} worker to compute a mini-batch gradient at the k^{th} local-step is modeled as a random variable $Y_{i,k} \sim F_Y$, assumed to be i.i.d. across workers and mini-batches. The communication delay is a random variable D for each all-node broadcast. The value of random variable D should depend on the number of workers invloved in the communication.

3 Jointly Analyzing Runtime and Error-Convergence

Runtime Analysis. We now present a comparison of the runtime per iteration of periodic averaging SGD ($T_{\text{P-Avg}}$) with fully synchronous SGD (T_{sync}) to illustrate how increasing τ can lead to a large runtime speed-up.

Theorem 1 (Runtime per iteration comparison)**.** *Suppose the i^{th} worker takes wall-clock time $Y_{i,k}$ to compute the gradient of the k^{th} mini-batch. Then,*

$$\frac{\mathbb{E}\left[T_{sync}\right]}{\mathbb{E}\left[T_{P\text{-}Avg}\right]} = \frac{\mathbb{E}\left[\max(Y_{1,1}, Y_{2,1}, \ldots, Y_{m,1}) + D\right]}{\mathbb{E}\left[\max(\overline{Y}_1, \overline{Y}_2, \ldots, \overline{Y}_m) + D/\tau\right]} = \frac{\mathbb{E}\left[Y_{m:m} + D\right]}{\mathbb{E}\left[\overline{Y}_{m:m} + D/\tau\right]} \tag{2}$$

where $\overline{Y}_i = (Y_{i,1} + Y_{i,2} + \cdots + Y_{i,\tau})/\tau$ denotes the average computation time of τ local updates at the i^{th} worker. The term $Y_{m:m}$ denotes the highest order statistic of m i.i.d. random variables [6].

Due to space limitations, please refer to the arXiv version[1] for the proof details if interested. Observe that as we increase τ, T_{sync} remains same but $T_{\text{P-Avg}}$ decreases in two ways: 1) τ-fold reduction in the communication delay and 2) reduction in the expectation of the maximum of the local computation times (Y is replaced by $\overline{Y}$ in the denominator), that is, reduction in additional delay due to slow or straggling workers. Figure 2 shows the probability distribution of T_{sync} and $T_{\text{P-Avg}}$ for exponentially distributed Y. Note that $T_{\text{P-Avg}}$ has a much lighter tail.

Joint Analysis with Error-convergence. Then, we combine the runtime analysis with previous error-convergence analysis for PASGD [25]. From the following theorem, one can easily observe the error-runtime trade-off for different communication periods.

Theorem 2 (Error-runtime Convergence of PASGD)**.** *For PASGD, under certain assumptions (stated in the arXiv version), if the learning rate satisfies $\eta L + \eta^2 L^2 \tau(\tau - 1) \leq 1$ and all workers are initialized at the same point $\mathbf{x}_1$, then after total T wall-clock time, the minimal expected squared gradient norm within this time interval will be bounded by:*

$$\frac{2\left[F(\mathbf{x}_1) - F_{inf}\right]}{\eta T}\left(\mathbb{E}\left[\overline{Y}_{m:m}\right] + \frac{\mathbb{E}\left[D\right]}{\tau}\right) + \frac{\eta L \sigma^2}{m} + \eta^2 L^2 \sigma^2 (\tau - 1) \tag{3}$$

where L is the Lipschitz constant of the objective function and σ^2 is the variance bound of mini-batch stochastic gradients. When the upper bound is arbitrary small, the algorithm is guaranteed to converge to a stationary point.

While a larger τ reduces the runtime per iteration and let the first term in (3) become smaller, it also adds additional noise and increases the last term. In Figure 3, we plot theoretical bounds for both fully synchronous SGD ($\tau = 1$) and PASGD. It is shown that although PASGD with $\tau = 10$ starts with a rapid drop, it will eventually converge to a high error floor. This theoretical result is corroborated by experiments in Section 5. Another direct outcome of Theorem 2 is the determination of the best communication period that balances the first and last terms in (3). We will discuss the selection of communication period later in Section 4.

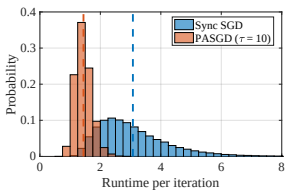


Figure 2: Probability distribution of runtime per iteration, where communication delay $D = 1$, mean computation time $\mathbb{E}[Y] = 1$, and number of workers $m = 16$. Dash lines represent the mean values.

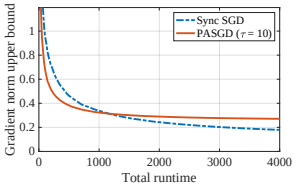


Figure 3: Illustration of theoretical error bound versus runtime in Theorem 2. The runtime per iteration is generated under the same parameters as Figure 2. Other constants in (3) are set as follows: $F(\mathbf{x}_1) = 1, F_{\text{inf}} = 0, \eta = 0.08, L = 1, \sigma^2 = 1$.

[1] `https://arxiv.org/abs/1810.08313`

4 ADACOMM: Proposed Adaptive Communication Strategy

Inspired by Theorem 2, the basic idea to adapt the communication is to choose the communication period that minimizes the optimization error at each wall-clock time. In order to achieve this goal, we divide the whole training procedure into uniform wall-clock time intervals with the same length T_0. At the beginning of each time interval, we select the best value of τ that has the fastest decay rate in the next T_0 wall-clock time as illustrated in Figure 4.

Determining the Best Communication Period for Each Time Interval. From Theorem 2, it can be observed that there is an optimal value τ^* that minimizes the optimization error bound at given wall-clock time. In particular, consider the simplest setting where Y and D are constants. Then, by minimizing the upper bound (3) over τ, we obtain the following.

Theorem 3. *For PASGD, under the same assumptions as Theorem 2, the optimization error upper bound in* (3) *at time T is minimized when the communication period is*

$$\tau^* = \sqrt{\frac{2(F(\mathbf{x}_1) - F_{inf})D}{\eta^3 L^2 \sigma^2 T}}. \tag{4}$$

The proof is straightforward by setting the derivative of (3) to zero. Directly applying Theorem 3 to the $l^{th}(l \geq 0)$ time interval, since workers can be viewed as restarting training at a new initial point $\mathbf{x}_1 = \mathbf{x}_{t=lT_0}$, the best choice of communication period is:

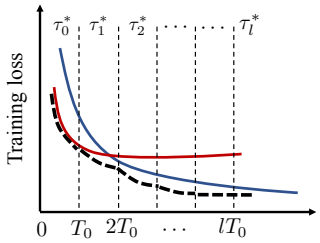


Figure 4: Illustration of communication period adaptation strategy: choosing the best τ for each time interval. Dash line denotes the learning curve using adaptive communication.

$$\tau_l = \sqrt{\frac{2(F(\mathbf{x}_{t=lT_0}) - F_{\text{inf}})D}{\eta_l^3 L_l^2 \sigma^2 T_0}} \tag{5}$$

where η_l, L_l denote the learning rate and local Lipschitz constant in the l^{th} time interval respectively. When the learning rate is fixed, it is easy to see the generated communication period sequence decreases along with the objective value $F(\mathbf{x}_t)$. *Decaying communication period* is similar in spirit to decaying learning rate. *The key difference is that here we are optimizing the true error convergence with respect to wall-clock time rather than the number iterations.*

Practical Considerations. Note that, for deep neural networks, estimating the unknown constants (such as Lipschitz constant L and variance bound σ^2) in (5) can be difficult and unreliable due to the highly non-convex and high-dimensional loss surface. As an alternative, we propose a simpler rule where we approximate F_{inf} by 0, $\eta_l L_l \approx 1$ (which is common in SGD literature), and divide (5) by the expression of τ_0 to obtain the basic communication period update rule:

$$\textbf{Basic update rule} \quad \tau_l = \left\lceil \sqrt{\frac{\eta_0}{\eta_l} \frac{F(\mathbf{x}_{t=lT_0})}{F(\mathbf{x}_{t=0})}} \tau_0 \right\rceil \tag{6}$$

where $\lceil a \rceil$ is the ceil function to round a to the nearest integer $\geq a$. Since the objective function values (i.e., training loss) $F(\mathbf{x}_{t=lT_0})$ and $F(\mathbf{x}_{t=0})$ can be easily obtained in the training, the only remaining thing now is to determine the initial communication period τ_0. We obtain a heuristic estimate of τ_0 by a simple grid search over different τ run for one or two epochs each. Further, in order to eliminate the noise introduced by local updates, we choose to first gradually decay the communication period to 1 and then decay the learning rate as usual.

Refinement: Faster Decay When Training Saturates. The communication period update rule (6) tends to give a decreasing sequence $\{\tau_l\}$. Nonetheless, when the training loss gets stuck on plateaus and decreases very slowly, (6) will result in τ_l saturating at the same value for a long time. To address this issue, the communication period will be multiplied by $\gamma < 1$ when the τ_l given by (6) is not strictly less than τ_{l-1}. Namely, $\tau_l = \gamma \tau_{l-1}$ if $\lceil \sqrt{\frac{\eta_0}{\eta_l} \frac{F(\mathbf{x}_{t=lT_0})}{F(\mathbf{x}_{t=0})}} \tau_0 \rceil \geq \tau_{l-1}$ and $\eta_l = \eta_{l-1}$. In the experiments, $\gamma = 1/2$ turns out to be a good choice. One can obtain a more aggressive decay in τ_l by either reducing the value of γ or introducing a slack variable s in the condition, such as $\lceil \sqrt{\frac{\eta_0}{\eta_l} \frac{F(\mathbf{x}_{t=lT_0})}{F(\mathbf{x}_{t=0})}} \tau_0 \rceil + s \geq \tau_{l-1}$.

5 Experimental Results

The proposed adaptive communication scheme was implemented in `PyTorch` [20] with `MPI4Py` [5]. All experiments were conducted on a local cluster where each worker node has an NVIDIA TitanX GPU. We evaluate our method for image classification tasks on CIFAR10 and CIFAR100 dataset [13]. Each worker machine is assigned with a partition which will be randomly shuffled after every epoch.

We choose to train deep neural networks VGG-16 [22] and ResNet-50 [11] from scratch. These two neural networks have different architectures and parameter sizes, thus resulting in different performance of periodic averaging. Moreover, unless otherwise stated, we used 4 worker nodes and mini-batch size on each worker is 128. The initial learning rates for VGG-16 and ResNet-50 are 0.2 and 0.4 respectively. The weight decay for both networks is 0.0005. In the variable learning rate setting, we decay the learning rate by 10 after $80^{\text{th}}/120^{\text{th}}/160^{\text{th}}/200^{\text{th}}$ epochs. The time interval length T_0 is set as 60 seconds (about 10 epochs for the initial communication period). Instead of training for a fixed number of epochs, we train all methods for sufficiently long time to convergence and compare the training loss and test accuracy, both of which are recorded after every 100 iterations.

Adaptive Communication in PASGD. Figure 5 presents the results for VGG-16 for both fixed and variable learning rates. A large communication period τ initially results in a rapid drop in the training loss, but the error finally converges to higher floor. By adapting τ, the proposed ADACOMM scheme strikes the best error-runtime trade-off in all settings. In Figure 5a, while fully synchronous SGD takes 33.5 minutes to reach 3×10^{-3} training loss, ADACOMM costs 15.5 minutes achieving more than $2\times$ speedup.

However, for ResNet-50, the communication overhead is no longer the bottleneck. For fixed communication period, the negative effect of performing local updates becomes more obvious and cancels the benefit of low communication delay (see Figures 6b and 6c). It is not surprising to see fully synchronous SGD is nearly the best one in the error-runtime plot among all fixed-τ methods. Even in this extreme case, adaptive communication can still have a competitive performance.

Adaptive Communication in Momentum SGD. The adaptive communication scheme is proposed based on the joint error-runtime analysis for PASGD without momentum. However, it can also be extended to other SGD variants, here we show that the proposed method works well for SGD with momentum. To combine momentum and periodic averaging, we use the *block momentum scheme* that was proposed in [3, 21] and applied to speech recognition tasks. The basic idea is treating the accumulated local updates in one period as one big gradient step between two synchronized models and introducing a global momentum for this big accumulated step. The update rule can be written as follows in terms of the momentum $\mathbf{u}_j$:

$$\mathbf{u}_j = \beta_{\text{glob}}\mathbf{u}_{j-1} + \mathcal{G}_j \tag{7}$$

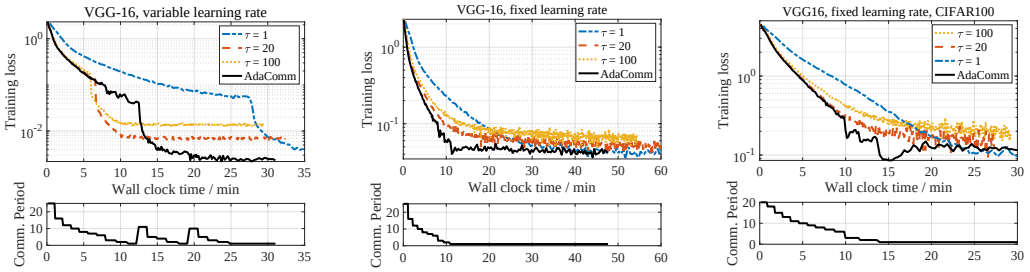
$$\mathbf{x}_{(j+1)\tau+1} = \mathbf{x}_{j\tau+1} - \eta_j \mathbf{u}_j \tag{8}$$

where $\mathcal{G}_j = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{\tau} g(\mathbf{x}_{j\tau+k}^{(i)})$ represents the accumulated gradients in the j^{th} local update period and β_{glob} denotes the global momentum factor. Moreover, workers can also conduct momentum SGD on local models, but their local momentum buffer will be cleared at the beginning of each local update period. Namely, we restart momentum SGD on local models after each averaging step. We set the global momentum factor as 0.3 and local momentum factor as 0.9 following [16].

In Figure 7, we observe significant performance gain when combing adaptive communication and block momentum. In particular, the adaptive communication scheme has the fastest convergence rate with respect to wall-clock time in the whole training process. While fully synchronous SGD gets stuck with a plateau before the first learning rate decay, the training loss of adaptive method continuously decreases until converging. For VGG-16 in Figure 7b, ADACOMM is $3.5\times$ faster (in terms of wall-clock time) than fully synchronous SGD in reaching a 3×10^{-3} training loss.

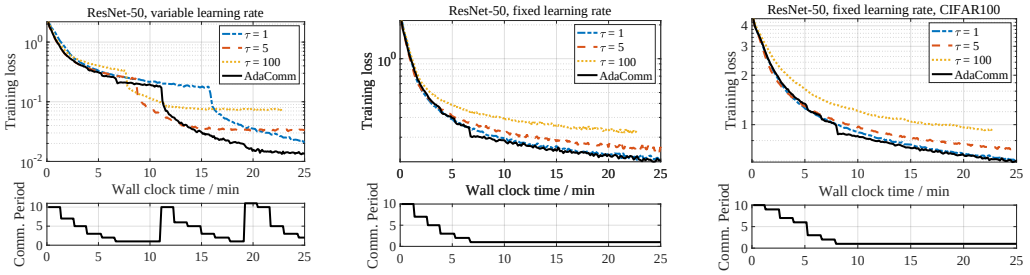
6 Concluding Remarks

The design of fast communication-efficient distributed SGD algorithms that are robust to system variability is vital to enable machine learning training to scale to resource-limited computing nodes. This paper is one of the first to analyze the convergence of error with respect to wall-clock time instead of number of iterations by accounting for the dependence of runtime per iteration on systems



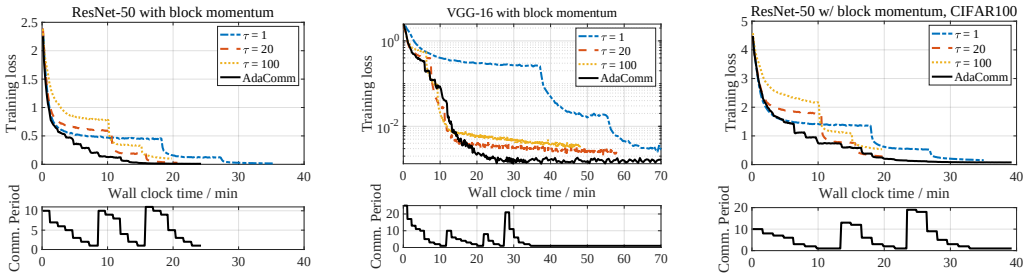
(a) Variable lr. on CIFAR10. (b) Fixed lr. on CIFAR10. (c) Fixed lr. on CIFAR100.

Figure 5: ADACOMM on VGG-16: Achieves $3.3\times$ speedup over fully synchronous SGD (in (b), 11.5 versus 38.0 minutes to achieve 4.5×10^{-2} training loss).



(a) Variable lr. on CIFAR10. (b) Fixed lr. on CIFAR10. (c) Fixed lr. on CIFAR100.

Figure 6: ADACOMM on ResNet-50: Achieves $1.4\times$ speedup over Sync SGD (in (a), 18.8 versus 26.6 minutes to achieve 2×10^{-2} training loss).



(a) ResNet-50 on CIFAR10. (b) VGG-16 on CIFAR10. (c) ResNet-50 on CIFAR100.

Figure 7: ADACOMM with block momentum: Achieves $3.5\times$ speedup over Sync SGD (in (b), 19.0 versus 66.7 minutes to achieve 3×10^{-3} training loss).

aspects such as computation and communication delays. We present a theoretical analysis of the error-runtime trade-off for periodic averaging SGD (PASGD), where each worker node performs local updates and their models are averaged after every τ iterations. Based on the joint error-runtime analysis, we design the first (to the best of our knowledge) adaptive communication strategy called ADACOMM for distributed deep learning. Experimental results using VGGNet and ResNet show that the proposed method can achieve up to a $3\times$ improvement in runtime, while achieving the same error floor as fully synchronous SGD. Going beyond periodic-averaging SGD, our idea of adapting frequency of averaging distributed SGD updates can be easily extended to other SGD frameworks including elastic-averaging [27], decentralized SGD [15] and parameter server-based training [7].

References

[1] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

[2] Pratik Chaudhari, Carlo Baldassi, Riccardo Zecchina, Stefano Soatto, Ameet Talwalkar, and Adam Oberman. Parle: parallelizing stochastic gradient descent. *arXiv preprint arXiv:1707.00424*, 2017.

[3] Kai Chen and Qiang Huo. Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 5880–5884. IEEE, 2016.

[4] Henggang Cui, James Cipar, Qirong Ho, Jin Kyu Kim, Seunghak Lee, Abhimanu Kumar, Jinliang Wei, Wei Dai, Gregory R Ganger, Phillip B Gibbons, et al. Exploiting bounded staleness to speed up big data analytics. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 37–48, 2014.

[5] Lisandro Dalcín, Rodrigo Paz, and Mario Storti. MPI for python. *Journal of Parallel and Distributed Computing*, 65(9):1108–1115, 2005.

[6] H. A. David and H. N. Nagaraja. *Order statistics*. John Wiley, Hoboken, N.J., 2003.

[7] 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*, pages 1223–1231, 2012.

[8] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, 13(Jan):165–202, 2012.

[9] Sanghamitra Dutta, Gauri Joshi, Soumyadip Ghosh, Parijat Dube, and Priya Nagpurkar. Slow and stale gradients can win the race: Error-runtime trade-offs in distributed SGD. *arXiv preprint arXiv:1803.01113*, 2018.

[10] Suyog Gupta, Wei Zhang, and Fei Wang. Model accuracy and runtime tradeoff in distributed deep learning: A systematic study. In *IEEE 16th International Conference on Data Mining (ICDM)*, pages 171–180. IEEE, 2016.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] Zhanhong Jiang, Aditya Balu, Chinmay Hegde, and Soumik Sarkar. Collaborative deep learning in fixed topology networks. In *Advances in Neural Information Processing Systems*, pages 5906–5916, 2017.

[13] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[14] 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, pages 583–598, 2014.

[15] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 5336–5346, 2017.

[16] Tao Lin, Sebastian U Stich, and Martin Jaggi. Don't use large mini-batches, use local SGD. *arXiv preprint arXiv:1808.07217*, 2018.

[17] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016.

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

[19] Philipp Moritz, Robert Nishihara, Ion Stoica, and Michael I Jordan. SparkNet: Training deep networks in spark. *arXiv preprint arXiv:1511.06051*, 2015.

[20] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[21] Frank Seide and Amit Agarwal. CNTK: Microsoft's open-source deep-learning toolkit. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2135–2135. ACM, 2016.

[22] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[23] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. Federated multi-task learning. In *Advances in Neural Information Processing Systems*, pages 4424–4434. 2017.

[24] Hang Su and Haoyu Chen. Experiments on parallel training of deep neural network using model averaging. *arXiv preprint arXiv:1507.01239*, 2015.

[25] Jianyu Wang and Gauri Joshi. Cooperative SGD: A unified framework for the design and analysis of communication-efficient SGD algorithms. *arXiv preprint arXiv:1808.07576*, 2018.

[26] Jian Zhang, Christopher De Sa, Ioannis Mitliagkas, and Christopher Ré. Parallel SGD: When does averaging help? *arXiv preprint arXiv:1606.07365*, 2016.

[27] Sixin Zhang, Anna E Choromanska, and Yann LeCun. Deep learning with elastic averaging SGD. In *NIPS'15 Proceedings of the 28th International Conference on Neural Information Processing Systems*, pages 685–693, 2015.

[28] Fan Zhou and Guojing Cong. On the convergence properties of a k-step averaging stochastic gradient descent algorithm for nonconvex optimization. *arXiv preprint arXiv:1708.01012*, 2017.