OPT2020: 12th Annual Workshop on Optimization for Machine Learning

Local AdaAlter: Communication-Efficient Stochastic Gradient Descent with Adaptive Learning Rates

Cong Xie [1,2] * CX2@ILLINOIS.EDU
Oluwasanmi Koyejo [1] SANMI@ILLINOIS.EDU
Indranil Gupta [1] INDY@ILLINOIS.EDU
Haibin Lin [2] HAIBIN.LIN.AWS@GMAIL.COM
[1] *Department of Computer Science, University of Illinois Urbana-Champaign* [2] *Amazon Web Services*

Abstract

When scaling distributed training, the communication overhead is often the bottleneck. In this paper, we propose a novel SGD variant with reduced communication and adaptive learning rates. We prove the convergence of the proposed algorithm for smooth but non-convex problems. Empirical results show that the proposed algorithm significantly reduces the communication overhead, which, in turn, reduces the training time by up to 30% for the 1B word dataset.

1. Introduction

Stochastic Gradient Descent (SGD) and its variants are commonly used for training deep neural networks. We can distribute the workload across multiple workers, which results in distributed SGD with data parallelism [8, 34–36]. A larger number of workers accelerates the training, but also increases the overall communication cost. In the worst case, it saturates the network interconnections. In this paper, we reduce the communication overhead by skipping communication rounds, and periodically averaging the models across the workers. Such an approach is called local SGD [17, 25, 30, 37, 38]. There are other approaches to reduce the communication overhead of distributed SGD, such as quantization [3, 4, 12, 22, 27, 32, 42] and sparsification [2, 10, 26, 33].

Adaptive learning rate methods adapt coordinate-wise dynamic learning rates by accumulating the historical gradients. Examples include AdaGrad [6, 18], RMSProp [28], AdaDelta [39], and Adam [13]. Along similar lines, recent research has shown that AdaGrad can converge without explicitly decreasing the learning rate [31, 44]. We note that these methods were not designed for local SGD. Nevertheless, in distributed SGD, it remains unclear how to use infrequent synchronization to reduce the communication overhead in SGD with adaptive learning rates. In this paper, we answer this question by introducing staleness to the updates of the adaptive learning rates. To be more specific, the update of the adaptive variables is delayed until the communication round.

We propose a novel SGD variant based on AdaGrad, and adopt the concept of local SGD to reduce the communication. To the best of our knowledge, this paper is the first to theoretically and empirically study local SGD with adaptive learning rates. The main contributions are as follows:

- We propose *Local AdaAlter*, a new technique to lazily update the adaptive variables. This enables communication reduction via periodic synchronization for SGD with adaptive learning rates.
- We prove the convergence of the proposed algorithm for non-convex problems.

* The work was done when Cong Xie was a (part-time) intern in Amazon Web Services.

- We show empirically that Local AdaAlter significantly reduces the communication overhead, thus also cutting training time by up to 30% for the 1B word.

2. Related work

In this paper, we consider a centralized server-worker architecture, also known as the Parameter Server (PS) [9, 15, 16, 21]. A common alternative is the AllReduce algorithm [23, 29]. Most of the existing deep-learning frameworks, such as Tensorflow [1], and PyTorch [24] support either of them. Similar to local SGD, there are other SGD variants that also reduce the communication overhead by skipping synchronization rounds, such as federated learning [14, 19] and EASGD [40]. In this paper, we focus on synchronous training with homogeneous workers. In contrast, asynchronous training [20, 41, 43] is faster when there are stragglers, but noisier due to asynchrony [7].

3. Problem formulation

We consider the optimization problem: $\min_{x\in\mathbb{R}^d} F(x)$, where $F(x) = \frac{1}{n}\sum_{i\in[n]} \mathbb{E}_{z_i\sim\mathcal{D}_i} f(x; z_i)$, for $\forall i \in [n]$, z_i is sampled from the local dataset $\mathcal{D}_i$ on the ith worker. We solve this problem in a distributed manner with n workers. In each iteration, the ith worker will sample a mini-batch of independent samples from the dataset $\mathcal{D}_i$, and compute the stochastic gradient $G_i = \nabla f(x; z_i), \forall i \in [n]$, where $z_i \sim \mathcal{D}_i$. Note that we assume non-IID workers, i.e., $\mathcal{D}_i \neq \mathcal{D}_j, \forall i \neq j$.

Table 1: Notations

Notation	Description
$x \in \mathbb{R}^d$	Model parameter
$F(x)$, $F_i(x)$, $f_i(x)$	$F(x) = \frac{1}{n}\sum_{i\in[n]} F_i(x)$; $F_i(x) = \mathbb{E}[f(x; z_i)]$, $z_i \sim \mathcal{D}_i$; $\mathbb{E}[f_i(x)] = F_i(x)$
T, t	Total number and index of iterations
G_t, $(G_t)_j$, $(\nabla F_t)_j$	Stochastic gradient $\mathbb{E}[G_t] = \nabla F(x_t)$, $(\cdot)_j$ is the jth coordinate, $j \in [d]$
$(G_{i,t})_j$	The jth coordinate of $G_{i,t}$, on the ith worker, $i \in [n]$, $j \in [d]$
$\circ$	Hadamard (coordinate-wise) product
B_t^2	$B_t^2 = b_0^2\mathbf{1} + \frac{1}{n}\sum_{i\in[n]}\sum_{s=1}^{t} G_{i,s} \circ G_{i,s}$, $B_0^2 = b_0^2\mathbf{1}$
B_t, $\frac{1}{B_t}$	$\left[\sqrt{(B_t^2)_1}, \ldots, \sqrt{(B_t^2)_d}\right]^\top$, $\left[\frac{1}{\sqrt{(B_t^2)_1}}, \ldots, \frac{1}{\sqrt{(B_t^2)_d}}\right]^\top$
$\frac{G_t}{B_t}$, $\frac{G_t}{\sqrt{B_t^2+\epsilon^2\mathbf{1}}}$	$G_t \circ \frac{1}{B_t} = \left[\frac{(G_t)_1}{\sqrt{(B_t^2)_1}}, \ldots, \frac{(G_t)_d}{\sqrt{(B_t^2)_d}}\right]^\top$, $\left[\frac{(G_t)_1}{\sqrt{(B_t^2)_1+\epsilon^2}}, \ldots, \frac{(G_t)_d}{\sqrt{(B_t^2)_d+\epsilon^2}}\right]^\top$

4. Methodology

First, we introduce two SGD variants that are highly related to our work: AdaGrad and local SGD. Then, we will propose a new algorithm: local AdaAlter.

4.1. Preliminary

To help understand our proposed algorithm, we first introduce the classic SGD variant with adaptive learning rate: AdaGrad. The detailed algorithm is shown in Algorithm 1. The general idea is to accumulate the gradients coordinate-wise as the denominator to normalize the gradients.

We adopt the concept of local SGD to reduce the communication overhead. The vanilla local SGD algorithm is shown in Algorithm 2. Local SGD skips the communication rounds, and synchronizes/averages the model parameters for every H iterations. Thus, on average, the communication overhead is reduced by the factor of H, compared to fully synchronous SGD.

Algorithm 1 Distributed AdaGrad

Initialize x_0, ϵ^2, $B_0^2 = \mathbf{0}$
for *iteration* $t \in [T]$ **do**
  for *workers* $i \in [n]$ *in parallel* **do**
    $G_{i,t} = \nabla f(x_{t-1}; z_{i,t})$, $z_{i,t} \sim \mathcal{D}_i$
    $G_t = \frac{1}{n}\sum_{i\in[n]} G_{i,t}$
    $B_t^2 \leftarrow B_{t-1}^2 + G_t \circ G_t$
    $x_t \leftarrow x_{t-1} - \eta \frac{G_t}{\sqrt{B_t^2+\epsilon^2\mathbf{1}}}$

Algorithm 2 Local SGD

Initialize $x_{1,0} = \ldots = x_{n,0} = x_0$
for *iteration* $t \in [T]$ **do**
  for *workers* $i \in [n]$ *in parallel* **do**
    $G_{i,t} = \nabla f(x_{i,t-1}; z_{i,t})$, $z_{i,t} \sim \mathcal{D}_i$
    $y_{i,t} \leftarrow x_{i,t-1} - \eta G_{i,t}$
    if $\mod(t, H) \neq 0$ **then** $x_{i,t} \leftarrow y_{i,t}$;
    else $x_{i,t} \leftarrow \frac{1}{n}\sum_{k\in[n]} y_{k,t}$;

4.2. Local AdaAlter

We propose an SGD variant based on AdaGrad, namely, local AdaAlter, which skips synchronization rounds, and periodically averages the model parameters *and* the accumulated denominators after every H iterations. The detailed algorithm is shown in Algorithm 3. Note that in the communication rounds, AdaAlter has to synchronize not only the model parameters, but also the accumulated denominators across the workers. Thus, compared to the distributed AdaGrad (Algorithm 1), local AdaAlter (Algorithm 3) reduces the communication overhead to $\frac{2}{H}$ on average.

Algorithm 3 Local AdaAlter

Initialize $x_{1,0} = \ldots = x_{n,0} = x_0$, $B_{1,0}^2 = \ldots = B_{n,0}^2 = b_0^2\mathbf{1}$, ϵ^2
for *iteration* $t \in [T]$ **do**
  for *workers* $i \in [n]$ *in parallel* **do**
    $t' = \mod(t-1, H) + 1$
    $G_{i,t} = \nabla f(x_{i,t-1}; z_{i,t})$, $z_{i,t} \sim \mathcal{D}_i$
    $y_{i,t} \leftarrow x_{i,t-1} - \eta \frac{G_{i,t}}{\sqrt{B_{i,t-t'}^2 + t'\epsilon^2\mathbf{1}}}$; $A_{i,t}^2 \leftarrow B_{i,t-1}^2 + G_{i,t} \circ G_{i,t}$
    if $\mod(t, H) \neq 0$ **then** $x_{i,t} \leftarrow y_{i,t}$; $B_{i,t}^2 \leftarrow A_{i,t}^2$;
    else Synchronize: $x_{i,t} \leftarrow \frac{1}{n}\sum_{k\in[n]} y_{k,t}$; $B_{i,t}^2 \leftarrow \frac{1}{n}\sum_{k\in[n]} A_{k,t}^2$;

Lazy update of the denominators: In AdaGrad, a small positive constant ϵ is added for the numerical stability, in case that the denominator B_t^2 is too small. However, in AdaAlter, ϵ^2 acts as a placeholder for the yet-to-be-added $G_{i,t} \circ G_{i,t}$. Thus, after t' local steps without synchronization, such placeholder becomes $t'\epsilon^2$. The denominators $B_{i,t}^2$ are updated in the synchronization rounds only, which guarantees that the denominators are the same on different workers in the local iterations. In a nutshell, in AdaAlter, the denominators $B_{i,t}^2$ are lazily updated to enable the infrequent synchronization. The key idea is to use $B_{i,t-t'}^2 + t'\epsilon^2\mathbf{1}$ as a placeholder before synchronization. Note that even if we take $H = 1$, AdaAlter is different from AdaGrad due to the lazy update.

The lazy update of the denominators keeps the adaptive learning rates synchronized across different workers, thus mitigates the noise caused by the local steps. However, it also incurs staleness in the adaptivity. In our experiments, we show that such small staleness does not affect the accuracy.

5. Theoretical analysis

In this section, we prove the convergence of Algorithm 3 for smooth but non-convex problems, with constant learning rate η. First, we introduce some assumptions for our convergence analysis.

Assumption 1 *(Smoothness) We assume that $F(x)$ and $F_i(x), \forall i \in [n]$ are L-smooth: $F_i(y) - F_i(x) \leq \langle \nabla F_i(x), y - x \rangle + \frac{L}{2}\|y - x\|^2, \forall x, y$.*

Assumption 2 *(Bounded gradients) For any stochastic gradient $G_{i,t} = \nabla f_i(x_t)$, we assume bounded coordinates $(G_{i,t})_j^2 \leq \rho^2, \forall j \in [d]$, or simply $\|G_{i,t}\|_\infty \leq \rho$.*

To analyze Algorithm 3, we introduce the following auxiliary variable: $\bar{x}_t = \frac{1}{n}\sum_{i\in[n]} x_{i,t}$. We show that the sequence $\bar{x}_0, \ldots, \bar{x}_T$ converges to a critical point. The detailed proof is in Appendix A.

Theorem 1 *(Convergence of local AdaAlter (Algorithm 3)) Taking arbitrary $\epsilon > 0$, $\eta \leq \frac{1}{L}$ in Algorithm 3, and $b_0 \geq 1$, under Assumption 1 and 2, Algorithm 3 converges to a critical point:* $\frac{\mathbb{E}\left[\sum_{t=1}^{T} \|\nabla F(\bar{x}_{t-1})\|^2\right]}{T} \leq \mathcal{O}\left(\frac{1}{\eta\sqrt{T}}\right) + \mathcal{O}\left(\frac{\eta^2 H^2 \log(T)}{\sqrt{T}}\right) + \mathcal{O}\left(\frac{\eta \log(T)}{n\sqrt{T}}\right)$.

With the constant learning rate η, local AdaAlter converges to a critical point when $T \to +\infty$. Increasing the number of workers n reduces the variance. Compared to the fully synchronous AdaAlter, local AdaAlter has the extra noise proportional to H^2 due to the reduced communication.

6. Experiments

In this section, we empirically evaluate the proposed algorithm.

6.1. Multi-GPU experiment on 1B Word

AdaGrad is mostly successful on language models. Thus, we conduct experiments on the 1B Word benchmark dataset [5]. We train Big LSTM model with 10% dropout (LSTM-2048-512 [11]).

6.1.1. EVALUATION SETUP

Our experiments are conducted on a single machine with 8 GPUs (an AWS P3.16 instance with 8 NVIDIA V100 GPUs, with 16GB memory per GPU). The batch size is 256 per GPU. We tune the learning rates in the range of $[0.2, 0.8]$ on the training data, and report the best results. Each experiment is composed of 50 epochs. Each epoch processes $20k \times 8 \times 256$ data samples. We repeat each experiment 5 times and take the average. In all the experiments, we take $\epsilon = 1$, $b_0 = 1$.

The typical measure used for language models is perplexity (PPL). We evaluate the following performance metrics to test the reduction of communication overhead and the convergence: i) the time consumed by one epoch versus different number of GPU workers; ii) the perplexity on the test dataset versus time; iii) the perplexity on the test dataset versus the number of epochs.

6.2. Practical remarks for AdaAlter on 1B Word

There are some additional remarks for using local AdaAlter in practice.

Warm-up Learning Rates: When using AdaAlter, we observe that it behaves almost the same as AdaGrad, except at the beginning, the denominator B_t^2 is too small for AdaAlter. Thus, we add a

warm-up mechanism for AdaAlter: $\eta_t \leftarrow \eta \times \min\left(1, \frac{t}{warm\_up\_steps}\right)$, where $warm\_up\_steps$ is a hyperparameter. In the first $warm\_up\_steps$ iterations, the learning rate will gradually increase from $\frac{\eta}{warm\_up\_steps}$ to η. In our default setting where we use 8 GPU workers with batch size 256, we take $\eta = 0.5$ and $warm\_up\_steps = 600$.

Scaling Learning Rates: The original baseline is conducted on 4 GPU with batch size 128 per GPU, and learning rate 0.2. When the batch size increases by k, it is a common strategy to re-scale the learning rate by k or $\sqrt{k}$ [8, 34–36]. In our experiments, we use 8 GPU with batch size 256 per GPU. Thus, we tuned η in the range of $[0.4, 0.8]$, and found that $\eta = 0.5$ results in the best performance.

6.2.1. EVALUATION RSULTS

Figure 1 illustrates the time consumed by one epoch and the throughput with different numbers of workers and different algorithms. We vary the synchronization periods H for local AdaAlter. It is shown that local AdaAlter efficiently reduces the communication overhead.

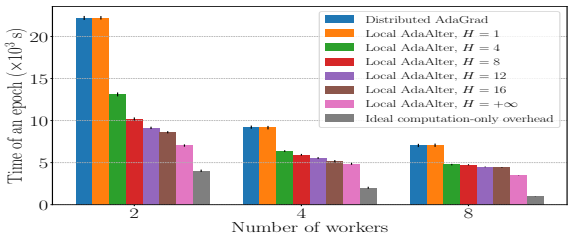


Figure 1: Time consumed by one epoch versus different numbers of workers.

Table 2: Test PPL and time at the end of training, for LSTM-2048-512 on 1B word dataset.

Method	**Test PPL**	**Time (hours)**
AdaGrad	44.58 ± 0.02	98.05
Local AdaAlter		
$H = 1$	44.36 ± 0.01	98.47
$H = 4$	44.08 ± 0.05	69.17
$H = 8$	44.26 ± 0.10	67.41
$H = 12$	44.30 ± 0.11	65.49
$H = 16$	44.51 ± 0.08	64.22

The overall overhead can be decomposed into three parts: computation, communication, and data loading. The baseline "Local AdaAlter, $H = +\infty$" is evaluated by manually removing the communication, and measures the ideal overhead without communication. The baseline "Ideal computation-only overhead" is evaluated by manually removing both the communication and the data-loading. These two baselines illustrate the ideal lower bounds of the training time.

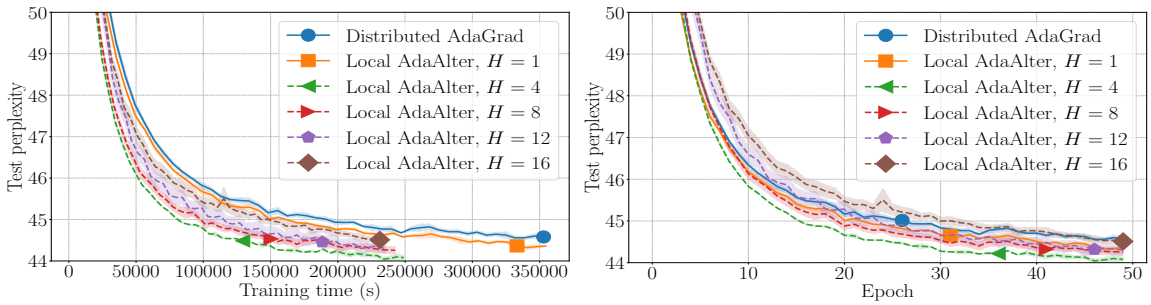
Figure 2 illustrates the perplexity on the test dataset and the loss on the training dataset with different algorithms. Compared to vanilla distributed AdaGrad, local AdaAlter enjoys equivalent convergence rate, but takes much less time. To reach the same perplexity, local AdaAlter can reduce almost 30% of the training time.

In Table 2, we report the perplexity and consumed time at the end of training for different algorithms. We can see that local AdaAlter produces comparable performance to the fully synchronous AdaGrad and AdaAlter, on the test dataset, with much less training time, and acceptable variance.

6.3. Discussion

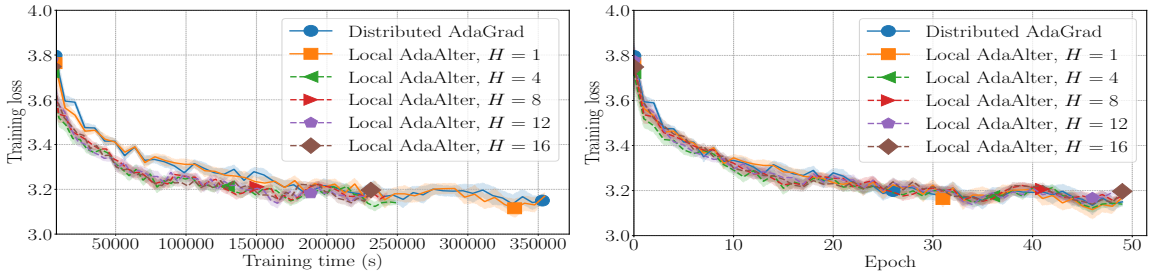
We can see that the fully synchronous AdaGrad or AdaAlter ($H = 1$) are very slow. Local AdaAlter reduces almost 30% of the training time compared to the fully synchronous AdaGrad or AdaAlter.

As we expected, Figure 2 and Table 2 show that larger H reduces more communication overhead, but also results in worse perplexity, which validates our theoretical analysis in Theorem 1: when H increases, the noise in the convergence also increases. Taking $H = 4$ gives the best trade-off between the communication overhead and the test perplexity.



(a) Test perplexity versus training time

(b) Test perplexity versus epochs



(c) Training loss versus training time

(d) Training loss versus epochs

Figure 2: Evaluation of different algorithms, for LSTM-2048-512 on 1B word dataset. We use a single machine with 8 GPU workers and local batch size 256 on each GPU. For all experiments, we take the learning rate $\eta = 0.5$. For local AdaAlter, we take $warm\_up\_steps = 600$ warm-up steps.

Interestingly, as shown in Figure 2(b), local AdaAlter with $H > 1$ has slightly better perplexity on the test dataset, compared to the fully synchronous AdaGrad and AdaAlter with $H = 1$. Although, our theoretical analysis indicates that local AdaAlter has larger variance compared to the fully synchronous version, such conclusion only applies to the training loss. In fact, there is previous work [17] showing that local SGD potentially generalizes better than the fully synchronous SGD. We also notice that when H is too large, such benefit will be overwhelmed by the large noise.

We also observe that almost all the algorithms do not scale well when scaling from 4 to 8 workers. The major reason is that all the workers are placed in the same machine, where CPU resources are limited. When there are too many workers, the data-loading becomes a bottleneck. That is also the reason why different H does not show much difference when using 8 GPU workers.

7. Conclusion

We propose a novel SGD algorithm: local AdaAlter, which reduces the communication overhead by skipping the synchronization rounds, and adopts adaptive learning rates. We show that the algorithm provably converges. Our empirical results also show accelerated training compared to baselines.

Acknowledgments

This work was funded in part by the following grants: NSF IIS 1909577, NSF CNS 1908888, NSF CCF 1934986 and a JP Morgan Chase Fellowship, along with computational resources donated by Intel, AWS, and Microsoft Azure.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zhang. Tensorflow: A system for large-scale machine learning. In *OSDI*, 2016.

[2] Alham Fikri Aji and Kenneth Heafield. Sparse communication for distributed gradient descent. In *EMNLP*, 2017.

[3] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. In *NIPS*, 2016.

[4] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Anima Anandkumar. signsgd: compressed optimisation for non-convex problems. In *ICML*, 2018.

[5] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, and Phillipp Koehn. One billion word benchmark for measuring progress in statistical language modeling. In *INTERSPEECH*, 2013.

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

[7] 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. In *AISTATS*, 2018.

[8] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *ArXiv*, abs/1706.02677, 2017.

[9] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B Gibbons, Garth A Gibson, Greg Ganger, and Eric P Xing. More effective distributed ml via a stale synchronous parallel parameter server. In *Advances in neural information processing systems*, pages 1223–1231, 2013.

[10] Peng Jiang and Gagan Agrawal. A linear speedup analysis of distributed deep learning with sparse and quantized communication. In *NeurIPS*, 2018.

[11] Rafal Józefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *ArXiv*, abs/1602.02410, 2016.

[12] Sai Praneeth Karimireddy, Quentin Rebjock, Sebastian U. Stich, and Martin Jaggi. Error feedback fixes signsgd and other gradient compression schemes. In *ICML*, 2019.

[13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[14] Jakub Konevcnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[15] 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.

[16] Mu Li, David G Andersen, Alexander J Smola, and Kai Yu. Communication efficient distributed machine learning with the parameter server. In *Advances in Neural Information Processing Systems*, pages 19–27, 2014.

[17] Tao Lin, Sebastian U. Stich, and Martin Jaggi. Don't use large mini-batches, use local sgd. *ArXiv*, abs/1808.07217, 2018.

[18] H. Brendan McMahan and Matthew J. Streeter. Adaptive bound optimization for online convex optimization. In *COLT*, 2010.

[19] 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.

[20] Feng Niu, Benjamin Recht, Christopher Ré, and Stephen J. Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, 2011.

[21] Y Peng, Y Zhu, Y Chen, Y Bao, B Yi, C Lan, C Wu, and C Guo. A generic communication scheduler for distributed dnn training acceleration. In *the 27th ACM Symposium on Operating Systems Principles (ACM SOSP 2019)*, 2019.

[22] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns. In *INTERSPEECH*, 2014.

[23] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *ArXiv*, abs/1802.05799, 2018.

[24] Benoit Steiner, Zachary DeVito, Soumith Chintala, Sam Gross, Adam Paszke, Francisco Massa, Adam Lerer, Gregory Chanan, Zeming Lin, Edward Yang, Alban Desmaison, Alykhan Tejani, Andreas Kopf, James Bradbury, Luca Antiga, Martin Raison, Natalia Gimelshein, Sasank Chilamkurthy, Trevor Killeen, Lu Fang, and Junjie Bai. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

[25] Sebastian U. Stich. Local sgd converges fast and communicates little. *ArXiv*, abs/1805.09767, 2018.

[26] Sebastian U. Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. Sparsified sgd with memory. *ArXiv*, abs/1809.07599, 2018.

[27] Nikko Strom. Scalable distributed dnn training using commodity gpu cloud computing. In *INTERSPEECH*, 2015.

[28] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2): 26–31, 2012.

[29] David W Walker and Jack J Dongarra. Mpi: a standard message passing interface. *Supercomputer*, 12:56–68, 1996.

[30] Jianyu Wang and Gauri Joshi. Cooperative sgd: A unified framework for the design and analysis of communication-efficient sgd algorithms. *ArXiv*, abs/1808.07576, 2018.

[31] Rachel Ward, Xiaoxia Wu, and Leon Bottou. Adagrad stepsizes: sharp convergence over nonconvex landscapes. In *International Conference on Machine Learning*, pages 6677–6686, 2019.

[32] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Terngrad: Ternary gradients to reduce communication in distributed deep learning. In *NIPS*, 2017.

[33] Cong Xie, Shuai Zheng, Oluwasanmi O Koyejo, Indranil Gupta, Mu Li, and Haibin Lin. Cser: Communication-efficient sgd with error reset. In *Advances in Neural Information Processing Systems*, 2020.

[34] Yang You, Igor Gitman, and Boris Ginsburg. Scaling sgd batch size to 32k for imagenet training. *ArXiv*, abs/1708.03888, 2017.

[35] Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel, and Kurt Keutzer. Imagenet training in minutes. In *ICPP*, 2017.

[36] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962*, 2019.

[37] Hao Yu, Sen Xiang Yang, and Shenghuo Zhu. Parallel restarted sgd with faster convergence and less communication: Demystifying why model averaging works for deep learning. In *AAAI*, 2018.

[38] Hao Yu, Rong Jin, and Sen Xiang Yang. On the linear speedup analysis of communication efficient momentum sgd for distributed non-convex optimization. In *ICML*, 2019.

[39] Matthew D. Zeiler. Adadelta: An adaptive learning rate method. *ArXiv*, abs/1212.5701, 2012.

[40] Sixin Zhang, Anna Choromanska, and Yann LeCun. Deep learning with elastic averaging sgd. In *ICLR*, 2014.

[41] Shen-Yi Zhao and Wu-Jun Li. Fast asynchronous parallel stochastic gradient descent: A lock-free approach with convergence guarantee. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[42] Shuai Zheng, Ziyue Huang, and James T. Kwok. Communication-efficient distributed blockwise momentum sgd with error-feedback. *ArXiv*, abs/1905.10936, 2019.

[43] Martin Zinkevich, Alexander J. Smola, and John Langford. Slow learners are fast. In *NIPS*, 2009.

[44] Fangyu Zou, Li Shen, Zequn Jie, Weizhong Zhang, and Wei Liu. A sufficient condition for convergences of adam and rmsprop. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11127–11135, 2019.

Appendix

Appendix A. Proofs

Lemma 1 *([44], Lemma 15) For any non-negative sequence $a_0, a_1, \ldots, a_T$, we have*

$$\sum_{t=1}^{T} \frac{a_t}{a_0 + \sum_{s=1}^{t} a_s} \leq \log\left(a_0 + \sum_{t=1}^{T} a_t\right) - \log(a_0).$$

To analyze Algorithm 3, we introduce the following auxiliary variable:

$$\bar{x}_t = \frac{1}{n} \sum_{i \in [n]} x_{i,t}.$$

Also, note that in Algorithm 3, $B_{i,t-t'}$ is synchronized. Thus, we denote

$$\bar{B}_{t-t'} = B_{1,t-t'} = \ldots = B_{n,t-t'}.$$

Theorem 2 *Taking arbitrary $\epsilon > 0$ in Algorithm 3, and $b_0 \geq 1$. Under Assumption 1 and 2, Algorithm 3 converges to a critical point: By telescoping and taking total expectation, we have*

$$\begin{aligned}
&\frac{\mathbb{E}\left[\sum_{t=1}^{T} \|\nabla F(\bar{x}_{t-1})\|^2\right]}{T} \\
&\leq \frac{2\sqrt{b_0^2 + \frac{T\epsilon^2}{p^2}}\,\mathbb{E}\left[F(\bar{x}_{t0}) - F(\bar{x}_T)\right]}{\eta T} + \left[4\eta^2 L^2 H^2 + \frac{1}{n} L\eta\right] \frac{d \log\left(b_0^2 + T\rho^2\right) \sqrt{b_0^2 + \frac{T\epsilon^2}{p^2}}}{Tp^2} \\
&\leq \mathcal{O}\left(\frac{1}{\eta\sqrt{T}}\right) + \mathcal{O}\left(\frac{\eta^2 H^2 \log(T)}{\sqrt{T}}\right) + \mathcal{O}\left(\frac{\eta \log(T)}{n\sqrt{T}}\right).
\end{aligned}$$

Proof

For convenience, we write the stochastic gradient $\nabla f(x_{i,t-1}; z_{i,t})$ as $\nabla f_i(x_{i,t-1})$, and we have $\mathbb{E}[\nabla f_i(x_{i,t-1})] = \nabla F_i(x_{i,t-1})$. Using L-smoothness, we have

$$\begin{aligned}
&F(\bar{x}_t) - F(\bar{x}_{t-1}) \\
&\leq -\eta \left\langle \nabla F(\bar{x}_{t-1}), \frac{1}{n} \sum_{i \in [n]} \frac{G_{i,t}}{\sqrt{B_{i,t-t'}^2 + t'\epsilon^2 \mathbf{1}}} \right\rangle + \frac{L\eta^2}{2} \left\| \frac{1}{n} \sum_{i \in [n]} \frac{G_{i,t}}{\sqrt{B_{i,t-t'}^2 + t'\epsilon^2 \mathbf{1}}} \right\|^2 \\
&\leq \sum_{j=1}^{d} \underbrace{-\eta \frac{1}{n} \sum_{i \in [n]} \frac{(\nabla F(\bar{x}_{t-1}))_j (G_{i,t})_j}{\sqrt{\left(\bar{B}_{t-t'}\right)_j^2 + t'\epsilon^2}}}_{①} + \frac{L\eta^2}{2} \sum_{j=1}^{d} \underbrace{\frac{\left(\frac{1}{n}\sum_{i \in [n]} G_{i,t}\right)_j^2}{\left(\bar{B}_{t-t'}\right)_j^2 + t'\epsilon^2}}_{②}.
\end{aligned}$$

Conditional on the previous states, taking expectation on both sides, we have

$$
\begin{aligned}
&\mathbb{E}\left[①\right] \\
&= -\eta \frac{\left(\nabla F(\bar{x}_{t-1})\right)_j \left(\frac{1}{n}\sum_{i\in[n]} \nabla F_i(x_{i,t-1})\right)_j}{\sqrt{\left(\bar{B}_{t-t'}\right)_j^2 + t'\epsilon^2}} \\
&= \underbrace{-\frac{\eta}{2} \frac{\left(\nabla F(\bar{x}_{t-1})\right)_j^2}{\sqrt{\left(\bar{B}_{t-t'}\right)_j^2 + t'\epsilon^2}}}_{③} \\
&\quad \underbrace{-\frac{\eta}{2} \frac{\left(\frac{1}{n}\sum_{i\in[n]} \nabla F_i(x_{i,t-1})\right)_j^2}{\sqrt{\left(\bar{B}_{t-t'}\right)_j^2 + t'\epsilon^2}}}_{④} \\
&\quad \underbrace{+\frac{\eta}{2} \frac{\left(\nabla F(\bar{x}_{t-1}) - \frac{1}{n}\sum_{i\in[n]} \nabla F_i(x_{i,t-1})\right)_j^2}{\sqrt{\left(\bar{B}_{t-t'}\right)_j^2 + t'\epsilon^2}}}_{⑤}.
\end{aligned}
$$

Again, conditional on the previous states, taking expectation on both sides, we have

$$
\begin{aligned}
&\mathbb{E}\left[②\right] \\
&= \mathbb{E}\left[\frac{\left(\frac{1}{n}\sum_{i\in[n]} \nabla f_i(x_{i,t-1})\right)_j^2}{\left(\bar{B}_{t-t'}\right)_j^2 + t'\epsilon^2}\right] \\
&= \mathbb{E}\left[\frac{\left(\frac{1}{n}\sum_{i\in[n]} (\nabla f_i(x_{i,t-1}) - \nabla F_i(x_{i,t-1}) + \nabla F_i(x_{i,t-1}))\right)_j^2}{\left(\bar{B}_{t-t'}\right)_j^2 + t'\epsilon^2}\right] \\
&= \underbrace{\mathbb{E}\left[\frac{\left(\frac{1}{n}\sum_{i\in[n]} (\nabla f_i(x_{i,t-1}) - \nabla F_i(x_{i,t-1}))\right)_j^2}{\left(\bar{B}_{t-t'}\right)_j^2 + t'\epsilon^2}\right]}_{⑥} \\
&\quad + \underbrace{\mathbb{E}\left[\frac{\left(\frac{1}{n}\sum_{i\in[n]} \nabla F_i(x_{i,t-1})\right)_j^2}{\left(\bar{B}_{t-t'}\right)_j^2 + t'\epsilon^2}\right]}_{⑦}
\end{aligned}
$$

In the following steps, we bound the terms ③-⑦, respectively.

Taking $p = \min(\frac{\epsilon}{\rho}, 1) \leq 1$, we have $\epsilon \geq p\rho$, or $\rho \leq \frac{\epsilon}{p}$. Thus, we have $\left(\bar{B}_{t-t'}\right)_j^2 + t'\epsilon^2 \leq b_0^2 + (t-t')\rho^2 + t'\epsilon^2 \leq b_0^2 + (t-t')\frac{\epsilon^2}{p^2} + t'\epsilon^2 \leq b_0^2 + t\frac{\epsilon^2}{p^2} \leq b_0^2 + T\frac{\epsilon^2}{p^2}$

$$\sum_{j=1}^{d} ③ \leq \sum_{j=1}^{d} -\frac{\eta}{2}\frac{(\nabla F(\bar{x}_{t-1}))_j^2}{\sqrt{b_0^2 + T\frac{\epsilon^2}{p^2}}} = -\frac{\eta}{2}\frac{\|\nabla F(\bar{x}_{t-1})\|^2}{\sqrt{b_0^2 + T\frac{\epsilon^2}{p^2}}}.$$

Since $\left(\bar{B}_{t-t'}\right)_j^2 + t'\epsilon^2 \geq b_0^2 \geq 1$, taking $\eta \leq \frac{1}{L}$, we have

$$\begin{aligned}
&④ + \frac{L\eta^2}{2}⑦ \\
&= -\frac{\eta}{2}\frac{\left(\frac{1}{n}\sum_{i\in[n]}\nabla F_i(x_{i,t-1})\right)_j^2}{\sqrt{\left(\bar{B}_{t-t'}\right)_j^2 + t'\epsilon^2}} + \frac{L\eta^2}{2}\frac{\left(\frac{1}{n}\sum_{i\in[n]}\nabla F_i(x_{i,t-1})\right)_j^2}{\left(\bar{B}_{t-t'}\right)_j^2 + t'\epsilon^2} \\
&\leq -\frac{\eta}{2}\frac{\left(\frac{1}{n}\sum_{i\in[n]}\nabla F_i(x_{i,t-1})\right)_j^2}{\sqrt{\left(\bar{B}_{t-t'}\right)_j^2 + t'\epsilon^2}} + \frac{\eta}{2}\frac{\left(\frac{1}{n}\sum_{i\in[n]}\nabla F_i(x_{i,t-1})\right)_j^2}{\left(\bar{B}_{t-t'}\right)_j^2 + t'\epsilon^2} \\
&\leq 0.
\end{aligned}$$

Using $\epsilon \geq p\rho$ and $p \leq 1$, we have

$$\begin{aligned}
⑥ &= \mathbb{E}\left[\frac{\left(\frac{1}{n}\sum_{i\in[n]}(\nabla f_i(x_{i,t-1}) - \nabla F_i(x_{i,t-1}))\right)_j^2}{\left(\bar{B}_{t-t'}\right)_j^2 + t'\epsilon^2}\right] \\
&= \frac{1}{n^2}E\left[\frac{\sum_{i\in[n]}(\nabla f_i(x_{i,t-1}) - \nabla F_i(x_{i,t-1}))_j^2}{\left(\bar{B}_{t-t'}\right)_j^2 + t'\epsilon^2}\right] \\
&\leq \frac{1}{n}E\left[\frac{\frac{1}{n}\sum_{i\in[n]}(\nabla f_i(x_{i,t-1}))_j^2}{\left(\bar{B}_{t-t'}\right)_j^2 + t'\epsilon^2}\right] \\
&\leq \frac{1}{n}E\left[\frac{\frac{1}{n}\sum_{i\in[n]}(\nabla f_i(x_{i,t-1}))_j^2}{p^2\left(\bar{B}_{t-t'}\right)_j^2 + t'p^2\rho^2}\right] \\
&\leq \frac{1}{np^2}E\left[\frac{\frac{1}{n}\sum_{i\in[n]}(\nabla f_i(x_{i,t-1}))_j^2}{\left(\bar{B}_t\right)_j^2}\right],
\end{aligned}$$

where $\left(\bar{B}_t\right)_j^2 = b_0^2 + \sum_{s=1}^{t}\frac{1}{n}\sum_{i\in[n]}(G_{i,s})_j^2$.

Finally, using smoothness, we have

$$\sum_{j=1}^{d} ⑤ = \frac{\eta}{2}\sum_{j=1}^{d}\frac{\left(\nabla F(\bar{x}_{t-1}) - \frac{1}{n}\sum_{i\in[n]}\nabla F_i(x_{i,t-1})\right)_j^2}{\sqrt{\left(\bar{B}_{t-t'}\right)_j^2 + t'\epsilon^2}}$$

$$
\begin{aligned}
&\leq \frac{\eta}{2}\sum_{j=1}^{d}\left(\frac{1}{n}\sum_{i\in[n]}\nabla F_i(\bar{x}_{t-1}) - \frac{1}{n}\sum_{i\in[n]}\nabla F_i(x_{i,t-1})\right)_j^2 \\
&\leq \frac{\eta}{2}\frac{1}{n}\sum_{i\in[n]}\|\nabla F_i(\bar{x}_{t-1}) - \nabla F_i(x_{i,t-1})\|^2 \\
&\leq \frac{\eta L^2}{2n}\sum_{i\in[n]}\|\bar{x}_{t-1} - x_{i,t-1}\|^2.
\end{aligned}
$$

Note that $\bar{x}_{t-1}$ is synchronized across the workers. Thus, we have

$$
\begin{aligned}
\bar{x}_{t-1} &= \bar{x}_{t-t'} - \eta\sum_{s=1}^{t'-1}\frac{1}{n}\sum_{i\in[n]}\frac{G_{i,t-t'+s}}{B_{i,t-t'}^2 + s\epsilon^2\mathbf{1}}, \\
x_{i,t-1} &= \bar{x}_{t-t'} - \eta\sum_{s=1}^{t'-1}\frac{G_{i,t-t'+s}}{B_{i,t-t'}^2 + s\epsilon^2\mathbf{1}}.
\end{aligned}
$$

Then, we have

$$
\begin{aligned}
&\sum_{j=1}^{d}⑤ \\
&\leq \frac{\eta L^2}{2n}\sum_{j=1}^{d}\sum_{i\in[n]}\left(\bar{x}_{t-1} - x_{i,t-1}\right)_j^2 \\
&\leq \frac{\eta^3 L^2}{2n}\sum_{j=1}^{d}\sum_{i\in[n]}\left[\sum_{s=1}^{t'-1}\left(\frac{1}{n}\sum_{k\in[n]}\frac{G_{k,t-t'+s}}{\bar{B}_{t-t'}^2 + s\epsilon^2\mathbf{1}} - \frac{G_{i,t-t'+s}}{\bar{B}_{t-t'}^2 + s\epsilon^2\mathbf{1}}\right)\right]_j^2 \\
&\leq \frac{2\eta^3 L^2}{n}\sum_{j=1}^{d}\sum_{i\in[n]}\left(\sum_{s=1}^{t'-1}\frac{G_{i,t-t'+s}}{\bar{B}_{t-t'}^2 + s\epsilon^2\mathbf{1}}\right)_j^2 \\
&\leq \frac{2\eta^3 L^2 H}{n}\sum_{j=1}^{d}\sum_{i\in[n]}\sum_{s=1}^{t'-1}\frac{\left(G_{i,t-t'+s}\right)_j^2}{\left(\bar{B}_{t-t'}^2\right)_j^2 + s\epsilon^2} \\
&\leq \frac{2\eta^3 L^2 H}{np^2}\sum_{j=1}^{d}\sum_{i\in[n]}\sum_{s=1}^{H}\frac{\left(G_{i,t-t'+s}\right)_j^2}{\left(\bar{B}_{t-t'+s}^2\right)_j^2}.
\end{aligned}
$$

Now, we combine all the ingredients above:

$$
\begin{aligned}
&\mathbb{E}\left[F(\bar{x}_t) - F(\bar{x}_{t-1})\right] \\
&\leq \sum_{j=1}^{d}\mathbb{E}\left[①\right] + \frac{L\eta^2}{2}\sum_{j=1}^{d}\mathbb{E}\left[②\right] \\
&\leq \sum_{j=1}^{d}\mathbb{E}\left[③ + ④ + ⑤\right] + \frac{L\eta^2}{2}\sum_{j=1}^{d}\mathbb{E}\left[⑥ + ⑦\right]
\end{aligned}
$$

$$\leq -\frac{\eta}{2}\frac{\|\nabla F(\bar{x}_{t-1})\|^2}{\sqrt{b_0^2+T\frac{\epsilon^2}{p^2}}}+\sum_{j=1}^{d}\mathbb{E}\left[\frac{2\eta^3L^2H}{np^2}\sum_{i\in[n]}\sum_{s=1}^{H}\frac{\left(G_{i,t-t'+s}\right)_j^2}{\left(\bar{B}_{t-t'+s}^2\right)_j^2}\right]$$
$$+\sum_{j=1}^{d}\mathbb{E}\left[\frac{L\eta^2}{2np^2}\frac{\frac{1}{n}\sum_{i\in[n]}\left(\nabla f_i(x_{i,t-1})\right)_j^2}{\left(\bar{B}_t\right)_j^2}\right].$$

By re-arranging the terms, we have

$$\|\nabla F(\bar{x}_{t-1})\|^2$$
$$\leq \frac{2\sqrt{b_0^2+T\frac{\epsilon^2}{p^2}}\mathbb{E}\left[F(\bar{x}_{t-1})-F(\bar{x}_t)\right]}{\eta}$$
$$+\frac{4\eta^2L^2H\sqrt{b_0^2+T\frac{\epsilon^2}{p^2}}}{p^2}\sum_{j=1}^{d}\mathbb{E}\left[\sum_{s=1}^{H}\frac{\frac{1}{n}\sum_{i\in[n]}\left(G_{i,t-t'+s}\right)_j^2}{\left(\bar{B}_{t-t'+s}^2\right)_j^2}\right]$$
$$+\frac{L\eta\sqrt{b_0^2+T\frac{\epsilon^2}{p^2}}}{np^2}\sum_{j=1}^{d}\mathbb{E}\left[\frac{\frac{1}{n}\sum_{i\in[n]}\left(\nabla f_i(x_{i,t-1})\right)_j^2}{\left(\bar{B}_t\right)_j^2}\right].$$

By telescoping and taking total expectation, we have

$$\frac{\mathbb{E}\left[\sum_{t=1}^{T}\|\nabla F(\bar{x}_{t-1})\|^2\right]}{T}$$
$$\leq \frac{2\sqrt{b_0^2+T\frac{\epsilon^2}{p^2}}\mathbb{E}\left[F(\bar{x}_{t0})-F(\bar{x}_T)\right]}{\eta T}$$
$$+\frac{4\eta^2L^2H\sqrt{b_0^2+T\frac{\epsilon^2}{p^2}}}{Tp^2}\sum_{j=1}^{d}\mathbb{E}\left[\sum_{t=1}^{T}\sum_{s=1}^{H}\frac{\frac{1}{n}\sum_{i\in[n]}\left(G_{i,t-t'+s}\right)_j^2}{\left(\bar{B}_{t-t'+s}^2\right)_j^2}\right]$$
$$+\frac{L\eta\sqrt{b_0^2+T\frac{\epsilon^2}{p^2}}}{nTp^2}\sum_{j=1}^{d}\mathbb{E}\left[\sum_{t=1}^{T}\frac{\frac{1}{n}\sum_{i\in[n]}\left(\nabla f_i(x_{i,t-1})\right)_j^2}{\left(\bar{B}_t\right)_j^2}\right]$$
$$\leq \frac{2\sqrt{b_0^2+T\frac{\epsilon^2}{p^2}}\mathbb{E}\left[F(\bar{x}_{t0})-F(\bar{x}_T)\right]}{\eta T}$$
$$+\frac{4\eta^2L^2H^2\sqrt{b_0^2+T\frac{\epsilon^2}{p^2}}}{Tp^2}\sum_{j=1}^{d}\mathbb{E}\left[\sum_{t=1}^{T}\frac{\frac{1}{n}\sum_{i\in[n]}\left(\nabla f_i(x_{i,t-1})\right)_j^2}{\left(\bar{B}_t\right)_j^2}\right]$$
$$+\frac{L\eta\sqrt{b_0^2+T\frac{\epsilon^2}{p^2}}}{nTp^2}\sum_{j=1}^{d}\mathbb{E}\left[\sum_{t=1}^{T}\frac{\frac{1}{n}\sum_{i\in[n]}\left(\nabla f_i(x_{i,t-1})\right)_j^2}{\left(\bar{B}_t\right)_j^2}\right]$$
$$\leq \frac{2\sqrt{b_0^2+T\frac{\epsilon^2}{p^2}}\mathbb{E}\left[F(\bar{x}_{t0})-F(\bar{x}_T)\right]}{\eta T}$$

$$+\frac{4\eta^2L^2H^2\sqrt{b_0^2+T\frac{\epsilon^2}{p^2}}}{Tp^2}d\log\left(b_0^2+T\rho^2\right)$$
$$+\frac{L\eta\sqrt{b_0^2+T\frac{\epsilon^2}{p^2}}}{nTp^2}d\log\left(b_0^2+T\rho^2\right).$$

■