1-bit Adam: Communication Efficient Large-Scale Training with Adam's Convergence Speed

Hanlin Tang [1 2] Shaoduo Gan [3] Ammar Ahmad Awan [1] Samyam Rajbhandari [1] Conglong Li [1] Xiangru Lian [2] Ji Liu [2] Ce Zhang [3] Yuxiong He [1]

Abstract

Scalable training of large models (like BERT and GPT-3) requires careful optimization rooted in model design, architecture, and system capabilities. From a system standpoint, communication has become a major bottleneck, especially on commodity systems with standard TCP interconnects that offer limited network bandwidth. Communication compression is an important technique to reduce training time on such systems. One of the most effective methods is error-compensated compression, which offers robust convergence speed even under 1-bit compression. However, state-of-the-art error compensation techniques only work with basic optimizers like **SGD** and **Momentum SGD**, which are linearly dependent on the gradients. They do not work with non-linear gradient-based optimizers like **Adam**, which offer state-of-the-art convergence efficiency and accuracy for models like BERT. In this paper, we propose **1-bit Adam** that reduces the communication volume by up to $5\times$, offers much better scalability, and provides the same convergence speed as uncompressed Adam. Our key finding is that Adam's variance (non-linear term) becomes stable during training, hence we can run Adam in the beginning (warmup phase) and use it as a precondition for Momentum SGD during the rest of the training (compression phase). Experiments on up to 256 GPUs show that 1-bit Adam enables up to $3.3\times$ higher throughput for BERT-Large pre-training and up to $2.9\times$ higher throughput for SQuAD fine-tuning. In addition, we provide theoretical analysis for our proposed work.

[1]Microsoft [2]Department of Computer Science, University of Rochester [3]Department of Computer Science, ETH Zurich. Correspondence to: Yuxiong He <yuxhe@microsoft.com>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

1. Introduction

Modern advancement of machine learning is heavily driven by the advancement of computational power and techniques. Nowadays, it is not unusual to train a single model using hundreds of computational devices such as GPUs. As a result, scaling up training algorithms in the distributed setting has attracted intensive interests over the years. One important direction is communication efficient distributed training, which enhances the scalability of the training system by reducing the communication cost. Example techniques include quantization (Zhang et al., 2017; Wangni et al., 2018), decentralization (Lian et al., 2017; Koloskova* et al., 2020; Li et al., 2018), and asynchronous communication (Zheng et al., 2016; Chaturapruek et al., 2015).

One widely used strategy for alleviating the communication overhead is gradient compression. Before communication, the original gradient $\boldsymbol{g}$ will be compressed into $\mathcal{C}_\omega[\boldsymbol{g}]$, where $\mathcal{C}_\omega[\cdot]$ is the compress operator[1]. As a result the communication volume could be greatly reduced. However, this gradient compression could slow down the convergence speed because important information might get lost during the compression. To recover this information lost, error-compensated compression strategy was proposed: Instead of compressing the gradient at t-th iteration directly, we would first add back the compression error from the last step and then do the compression. Recent studies (Stich et al., 2018) observed that by using error-compensated compression, the asymptotic convergence speed remains unchanged for SGD even using 1-bit compression.

On the other hand, many state-of-the-art models have to be trained using a more complicated variant, Adam (Kingma and Ba, 2014). For example, to train models such as BERT, one has to resort to the Adam optimizer, since training it with vanilla/momentum SGD has been shown to be less effective. Unfortunately, we find that error-compensated compression does not work for Adam, because Adam is non-linearly dependent on the gradient which affects the error compensation mechanism (see Section 3.2 and 4.2 for more details).

[1]$\mathcal{C}_\omega[\cdot]$ could also include randomness.

In this paper, we first analyze the limitation of directly applying existing compression technique to Adam. One of our key findings is that Adam's variance (the non-linear term) becomes stable at early stage of training (Section 3.3). This motivates us to design a new 2-stage algorithm, 1-bit Adam, which uses Adam (warmup stage) to "pre-condition" a communication compressed momentum SGD algoirthm (compression stage). We provide theoretical analysis on communication compressed momentum SGD, which is the core component of 1-bit Adam. We design a custom collective primitive using MPI to transfer the $5\times$ communication volume reduction (achieved by our algorithm) into actual runtime speedup, which is hard to accomplish using existing DL framework libraries. Experiments with BERT-Base, BERT-Large, SQuAD 1.1 and ResNet-18 training tasks on up to 256 GPUs show that 1-bit Adam converges as fast as uncompressed Adam, and runs up to $3.3\times$ faster than uncompressed algorithms.

(Contributions) We make the following contributions:

- We propose a new algorithm, 1-bit Adam, a communication efficient momentum SGD algorithm pre-conditioned with Adam optimizer, which to the best of our knowledge is the first work that apply a pre-conditioned strategy for compressed momentum SGD. We present theoretical analysis on the convergence of 1-bit Adam, and show that it admits the same asymptotic convergence rate as the uncompressed one.

- We conduct experiments on large scale ML tasks that are currently challenging for SGD to train. We show that on both BERT pre-training, SQuAD fine-tuning and ResNet-18, 1-bit Adam is able to achieve the same convergence behaviour and final accuracy as Adam, together with up to $5\times$ less communication volume and $3.3\times$ faster end-to-end throughput (including the full-precision warmup stage). To our best knowledge, this is the first distributed learning algorithm with communication compression that can train a model as demanding as BERT.

- We implement a custom collective communication primitive using Message Passing Interface (MPI) to provide a scalable and efficient communication system for 1-bit Adam.

- The 1-bit Adam optimizer and the communication primitive backend have been open sourced in a deep learning optimization library called DeepSpeed[2].

2. Related Work

Communication-efficient distributed learning: To further reduce the communication overhead, one promising direction is to compress the variables that are sent between different workers (Yu et al., 2019; Ivkin et al., 2019). Previous work has applied a range of techniques such as quantizaiton, sparsification, and sketching (Alistarh et al., 2017; Agarwal et al., 2018; Spring et al., 2019; Ye and Abbe, 2018; Shi et al., 2021). The compression is mostly assumed to be unbiased (Wangni et al., 2018; Shen et al., 2018; Zhang et al., 2017; Wen et al., 2017; Jiang and Agrawal, 2018). A general theoretical analysis of centralized compressed parallel SGD can be found in Alistarh et al. (2017). Beyond this, some biased compressing methods are also proposed and proven to be quite efficient in reducing the communication cost. One example is the **1-bit SGD** (Seide et al., 2014), which compresses the entries in gradient vector into ± 1 depends on its sign. The theoretical guarantee of this method is given in Bernstein et al. (2018).

Error-compensated compression: The idea of using error compensation for compression is proposed in Seide et al. (2014), where they find that by using error compensation the training could still achieves a very good speed even using 1-bit compression. Recent study indicates that this strategy admits the same asymptotic convergence rate as the uncompressed one (Stich et al., 2018), which means that the influence of compression is trivial. More importantly, by using error compensation, it has been proved that we can use almost any compression methods (Stich et al., 2018), whereas naive compression could only converge when the compression is unbiased (the expectation of the compressed tensor is the same as the original). This method can be combined with decentralized training (Vogels et al., 2020), local SGD (Xie et al., 2020), accelerated algorithms (Gorbunov et al., 2020). Due to the promising efficiency of this method, error compensation has been applied into many related area (Zheng et al., 2019; Phuong and Phong, 2020; Yu et al., 2019; Shi et al., 2019; Ivkin et al., 2019; Sun et al., 2019; Basu et al., 2019; Vogels et al., 2019) in order to reduce the communication cost.

Adam: Adam (Kingma and Ba, 2015) has shown promising speed for many deep learning tasks, and also admits a very good robustness to the choice of the hyper-parameters, such as learning rate. It can be viewed as an adaptive method that scales the learning rate with the magnitude of the gradients on each coordinate when running SGD. Beyond Adam, many other strategies that that shares the same idea of changing learning rate dynamically was studied. For example, Duchi et al. (2011) (**Adagrad**) and (Tieleman and Hinton, 2011) (**RMSprop**), use the gradient, instead of momentum, for updating the parameters; **Adadelta** (Zeiler, 2012) changes the variance term of Adam into a non-decreasing updating rule; Luo et al. (2019) proposed **AdaBound** that gives both upper bound and lower bound for the variance term. In Alacaoglu et al. (2020); Liu et al. (2020) authors

[2]https://github.com/microsoft/DeepSpeed

develop a novel analysis for the convergence rate of Adam.

3. Motivation and Insights

3.1. Communication overhead affects the efficiency of distributed training

To demonstrate the opportunity for communication compression, we conduct performance profiling experiments that measures the impact of communication time with respect to the total training time per step. Here we use BERT-Large pre-training task as an example (sequence length 128, detailed training parameters can be found at Section 7.1), since BERT and transformer models in general are the state-of-the-art approaches in natural language processing and many other areas. We evaluate two different kinds of clusters: the first cluster has 4 NVIDIA Tesla V100 GPUs per node, and different nodes are connected by 40 Gigabit Ethernet (effective bandwidth is 4.1 Gbps based on iperf benchmark); the second cluster has 8 V100 GPUs per node, and different nodes are connected by 100 Gigabit InfiniBand EDR (effective bandwidth is close to theoretical peak based on microbenchmark). We perform BERT-Large pre-training using the two clusters with different number of nodes and GPUs, batch sizes, and gradient accumulation steps. And we measure the average latency of forward, backward (allreduce and everything else), and step function calls. Table 1 presents the profiling results.

Results show that allreduce communication contributes to a great portion of the training time per step, up to 94% and 75% for our experiments on two different kinds of inter-node networks. As expected, communication overhead is proportionally larger when the number of nodes is larger, when the batch size/gradient accumulation step is smaller, and when the network bandwidth is lower. These are the situations where communication compression could provide the most benefit.

3.2. Basic compression affects Adam's convergence

Given the great opportunity for communication compression, we investigate whether existing error-compensated gradient compression strategy can be applied to Adam, an important optimization algorithm for large model distributed training. We implement a basic compression strategy for Adam based on the compression-based SGD approach (Stich et al., 2018), where we perform error-compensated 1-bit compression over the gradient, and update both the momentum and variance based on the compressed gradient. We compare the BERT-Large pre-training (sequence length 128) training loss when using vanilla Adam and Adam with our basic compression strategy in Figure 1.

Results show that basic compression based on existing work

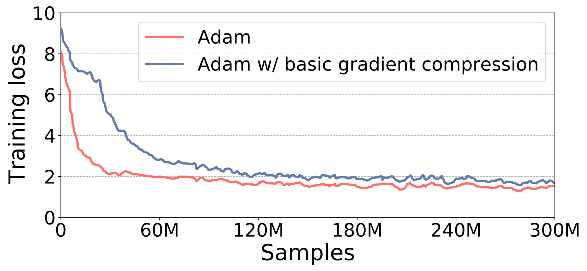


Figure 1. Training loss for BERT-Large pre-training using vanilla Adam and Adam with error compensated gradient compression.

greatly affects the convergence speed for Adam. The main reason is that Adam is non-linearly dependent to the gradients (see Section 4.2 for more details). This motivates us to look for novel compression strategy that overcomes the non-linear gradient dependency challenge, and at the same time achieves the same convergence speed as Adam.

3.3. Adam's variance becomes stable during training

Unlike SGD, which directly uses the gradient $\boldsymbol{g}$ to update the model $\boldsymbol{x}$, Adam uses two auxiliary variables $\boldsymbol{m}$ and $\boldsymbol{v}$ for the update. The mathematical updating rule of original Adam can be summarized as:

$$\begin{aligned}\boldsymbol{m}_{t+1} =&\beta_1\boldsymbol{m}_t + (1-\beta_1)\boldsymbol{g}_t\\ \boldsymbol{v}_{t+1} =&\beta_2\boldsymbol{v}_t + (1-\beta_2)(\boldsymbol{g}_t)^2,\\ \boldsymbol{x}_{t+1} =&\boldsymbol{x}_t - \gamma\frac{\boldsymbol{m}_{t+1}}{\sqrt{\boldsymbol{v}_{t+1}}+\eta}\end{aligned} \tag{1}$$

Here $\boldsymbol{x}_t$ is the model at t-iteration, $\boldsymbol{g}_t = \nabla F(\boldsymbol{x}_t;\boldsymbol{\zeta}_t)$ is the stochastic gradient, γ is the learning rate, η usually is a very small constant, β_1 and β_2 are decaying factor that controls the speed of forgetting history information. Notice here we disable the bias correction term in the original Adam, which is consistent with exact optimizer for training BERT (Devlin et al., 2019).

Here we refer $\boldsymbol{m}_t$ as the momentum term and $\boldsymbol{v}_t$ as the variance term. Notice that when $\boldsymbol{v}_t$ is changed into a constant $\boldsymbol{v}$, then Adam becomes equivalent to Momentum SGD under a coordinate-dependent learning rate $\frac{\gamma}{\sqrt{\boldsymbol{v}}+\eta}$.

To investigate the non-linear gradient dependency challenge, we analyze Adam's variance during BERT-Large pre-training (seqlen 128). At each step, we fuse the variance of all parameters, and calculate the norm of the fused variance. Figure 2 presents this fused variance norm at each step. Results show that the variance norm becomes stable after around $23K$ steps. This motivates our approach 1-bit Adam to "freeze" the Adam variance after it becomes stable, and then use it as a precondition during 1-bit compression stage.

Table 1. BERT-Large pre-training sequence 128 profiling results.

Cluster Network Type	Num. node	Num. GPU	Batch size per GPU	Batch size	Grad accum. step	Forward (ms)	Backward allreduce (ms)	Backward everything else (ms)	Step (ms)	allreduce%
Ethernet	16	64	1	64	1	36.65	2205.86	33.63	74.96	**94%**
Ethernet	16	64	16	1024	1	35.71	2275.43	60.81	75.59	93%
Ethernet	16	64	16	4096	4	137.80	2259.36	243.72	74.92	83%
Ethernet	8	32	16	512	1	37.91	2173.35	60.71	75.63	93%
Ethernet	4	16	16	256	1	36.94	2133.24	62.82	76.85	92%
Ethernet	2	8	16	128	1	34.95	1897.21	61.23	75.26	92%
Ethernet	1	4	16	64	1	35.99	239.76	59.95	74.21	58%
InfiniBand	8	64	1	64	1	25.36	316.18	23.25	58.49	**75%**
InfiniBand	8	64	16	1024	1	32.81	336.40	59.99	57.79	69%
InfiniBand	8	64	16	4096	4	131.04	339.52	237.92	56.91	44%
InfiniBand	4	32	16	512	1	33.45	297.28	56.81	57.98	67%
InfiniBand	2	16	16	256	1	32.86	183.74	56.49	58.60	55%
InfiniBand	1	8	16	128	1	32.74	28.18	59.73	57.29	16%

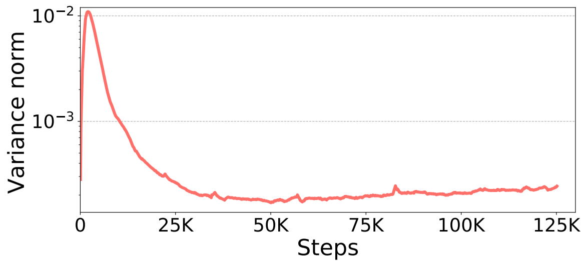


Figure 2. Norm of fused variance for BERT-Large pre-training using vanilla Adam. The y-axis is in log scale.

4. 1-bit Adam Algorithm

In this section, we start with some background introduction for error compensated compression and why it is incompatible with Adam. Then we give full description of 1-bit Adam.

Problem setting In this paper, we focus on the following optimization task and rely on the following notions and definitions:

$$\min_{\boldsymbol{x}\in\mathcal{R}^d} \quad f(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} \underbrace{\mathbb{E}_{\boldsymbol{\zeta}^{(i)}\sim\mathcal{D}_i} F(\boldsymbol{x};\boldsymbol{\zeta}^{(i)})}_{:=f_i(\boldsymbol{x})}, \qquad (2)$$

where d is the dimension of the input model $\boldsymbol{x}$, n is the number of workers included, $\mathcal{D}_i$ is the data distribution of individual data sample $\boldsymbol{\zeta}^{(i)}$ on the i-th worker, $F(\boldsymbol{x};\boldsymbol{\zeta})$ is the loss function.

Notations and definitions Throughout this paper, we use the following notations:

- $\nabla f(\cdot)$ denotes the gradient of a function f.
- f^* denotes the optimal value of the minimization problem (2).
- $f_i(\boldsymbol{x}) := \mathbb{E}_{\boldsymbol{\zeta}^{(i)}\sim\mathcal{D}_i} F(\boldsymbol{x};\boldsymbol{\zeta}^{(i)})$.
- $\|\cdot\|$ denotes the ℓ_2 norm for vectors and the spectral norm for matrices.
- $\|X\|_A := \mathrm{Tr}(X^\top AX)$.
- $\boldsymbol{C}_\omega(\cdot)$ denotes the randomized compressing operator.
- $\sqrt{\cdot}$ denotes the square root of the argument. In this paper if the argument is a vector, then it returns a vector taking the element-wise square root.
- $(\boldsymbol{x})^2$ denotes the element-wise square operation if $\boldsymbol{x}$ is a vector.
- $\frac{\boldsymbol{a}}{\boldsymbol{b}}$ or $\boldsymbol{a}/\boldsymbol{b}$ denotes the element-wise division operation if both $\boldsymbol{a}$ and $\boldsymbol{b}$ are vectors and their dimension matches.

4.1. Why error compensation works for SGD

For SGD , since the update is linearly dependent to the gradient, using error compensation could potentially remove the side-effect of the history compression error. The updating rule of **vanilla SGD** follows

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \gamma\boldsymbol{g}_t = \boldsymbol{x}_0 - \gamma\sum_{s=0}^{t}\boldsymbol{g}_s. \qquad (3)$$

When directly compressing the gradient without error compensation, the updating rule becomes

$$\begin{aligned}\boldsymbol{x}_{t+1} =& \boldsymbol{x}_t - \gamma C_\omega[\boldsymbol{g}_t] = \boldsymbol{x}_t - \gamma(\boldsymbol{g}_t - \boldsymbol{\delta}_t)\\ =& \boldsymbol{x}_0 - \gamma\sum_{s=0}^{t}\boldsymbol{g}_s + \underbrace{\gamma\sum_{s=0}^{t}\boldsymbol{\delta}_s}_{\text{history compression error}}.\end{aligned} \qquad (4)$$

As we can see in (4), the history compression error would get accumulated and therefore slow down the convergence rate. Moreover, previous work (Alistarh et al., 2017) indicates that when using biased compression operator, the training convergence cannot be guaranteed.

Now if we apply error compensation at each compression step, the updating rule becomes

$$\begin{aligned}\boldsymbol{x}_{t+1} =& \boldsymbol{x}_t - \gamma C_\omega[\boldsymbol{g}_t + \boldsymbol{\delta}_{t-1}] = \boldsymbol{x}_t - \gamma(\boldsymbol{g}_t - \underbrace{\boldsymbol{\delta}_t + \boldsymbol{\delta}_{t-1}}_{\text{error cancellation}})\\ =& \boldsymbol{x}_0 - \gamma\sum_{s=0}^{t}\boldsymbol{g}_s + \gamma\sum_{s=0}^{t}(\boldsymbol{\delta}_s - \boldsymbol{\delta}_{s-1})\\ =& \boldsymbol{x}_0 - \gamma\sum_{s=0}^{t}\boldsymbol{g}_s + \gamma\boldsymbol{\delta}_t. \end{aligned} \tag{5}$$

This demonstrates that by using error compensation, each step's compression error would get cancelled in the next step instead of getting accumulated over steps. To make the error compensation work correctly, it is necessary that we ensure an error cancellation term $\boldsymbol{\delta}_t + \boldsymbol{\delta}_{t-1}$ in the updating rule. Below we are going to see that this cannot be achieved for Adam.

4.2. Why Adam cannot be combined with error compensation

As we can see, Adam is non-linearly dependent to the gradient, and this non-linearity is widely believed to be essential for the superiority of Adam. Below we are going to first intuitively explain why error compensation works well for SGD, and then discuss two major reasons why this non-linearity makes Adam incompatible with error compensation.

Difficulty for estimating the variance term $\boldsymbol{v}$. Notice that for Adam, it is necessary to communicate the gradient $\boldsymbol{g}_t$ or momentum $\boldsymbol{m}_t$, and the variance term can be updated using $\boldsymbol{g}_t$. However, when using error-compensated gradient to update $\boldsymbol{v}_t$, the updating rule follows:

$$\begin{aligned}\boldsymbol{v}_{t+1} =& \beta_2\boldsymbol{v}_t + (1-\beta_2)\left(C_\omega[\boldsymbol{g}_t + \boldsymbol{\delta}_{t-1}]\right)^2\\ =& \beta_2\boldsymbol{v}_t + (1-\beta_2)\left(\boldsymbol{g}_t + \boldsymbol{\delta}_{t-1} - \boldsymbol{\delta}_t\right)^2\\ =& \beta_2\boldsymbol{v}_t + (1-\beta_2)\left(\boldsymbol{g}_t\right)^2 + \underbrace{\left(\boldsymbol{\delta}_{t-1} - \boldsymbol{\delta}_t\right)^2}_{\text{non-linear error correction}}\\ &+ 2\langle\boldsymbol{g}_t, \boldsymbol{\delta}_{t-1} - \boldsymbol{\delta}_t\rangle.\end{aligned}$$

Here the quadratic term $(\boldsymbol{\delta}_{t-1} - \boldsymbol{\delta}_t)^2$ cannot be cancelled by itself, therefore it will be hard to get an accurate estimation of $\boldsymbol{v}_t$ with history error being cancelled.

Difficulty for setting the correction factor. Another problem is that for SGD , when applying error compensation under a time varying learning rate γ_t, we need to compensate the history error using

$$C\left[\boldsymbol{g}_t + \frac{\gamma_t}{\gamma_{t-1}}\boldsymbol{\delta}_{t-1}\right],$$

instead of adding back $\boldsymbol{\delta}_{t-1}$ directly. In this case, if we view $\frac{\gamma}{\sqrt{\boldsymbol{v}_t}+\eta}$ as a coordinate-dependent learning rate, which makes Adam equivalent to Momentum SGD with time-varying learning rate, we need to apply the scale factor according to

$$\boldsymbol{m}_{t+1} = C_\omega\left[\beta_1\boldsymbol{m}_t + (1-\beta_1)\boldsymbol{g}_t + \frac{\sqrt{\boldsymbol{v}_{t-1}}+\eta}{\sqrt{\boldsymbol{v}_t}+\eta}\boldsymbol{\delta}_{t-1}\right].$$

The problem is that we cannot get the value of $\boldsymbol{v}_t$ after the compression, which makes it impossible to set the scale factor for error compensation.

4.3. 1-bit Adam

Based on our findings (Section 3.3) that Adam's variance term becomes stable at an early stage, we propose 1-bit Adam summarized in Algorithm 1. First we use vanilla Adam for a few epochs as a warm-up. After the warm-up stage, the compression stage starts and we stop updating the variance term $\boldsymbol{v}$ and use it as a fixed precondition. At the compression stage, we communicate based on the momentum applied with error-compensated 1-bit compression. The momentums are quantized into 1-bit representation (the sign of each element). Accompanying the vector, a scaling factor is computed as $\frac{\text{magnitude of compensated gradient}}{\text{magnitude of quantized gradient}}$. This scaling factor ensures that the compressed momentum has the same magnitude as the uncompressed momentum. This 1-bit compression could reduce the 97% communication cost of the original for float32 type training and 94% for float16 type training.

5. Theoretical Analysis

Notice that for 1-bit Adam, we only use original Adam at warm-up, and then we essentially run error-compensated momentum SGD with coordinate-dependent learning rate $\frac{\gamma}{\sqrt{\boldsymbol{v}_{T_w}}}$. Therefore here we consider the Adam-based warm-up phase as a way to find a good precondition variance term $\boldsymbol{v}_{T_w}$ to be used in the compression phase. Below we are going to introduce the convergence rate for the compression phase after warm-up. We first introduce some necessary assumptions, then we present the theoretical guarantee of the convergence rate for 1-bit Adam.

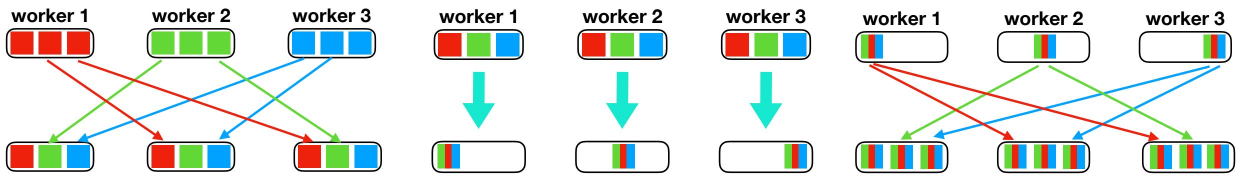
Assumption 1. *We make the following assumptions:*

1. ***Lipschitzian gradient:*** *$f(\cdot)$ is assumed to be with L-Lipschitzian gradients, which means*

$$\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\| \le L\|\boldsymbol{x} - \boldsymbol{y}\|, \quad \forall\boldsymbol{x}, \forall\boldsymbol{y},$$

2. ***Bounded variance:*** *The variance of the stochastic gradient is bounded*

$$\mathbb{E}_{\boldsymbol{\zeta}^{(i)}\sim\mathcal{D}_i}\|\nabla F(\boldsymbol{x};\boldsymbol{\zeta}^{(i)}) - \nabla f(\boldsymbol{x})\|^2 \le \sigma^2, \quad \forall\boldsymbol{x}, \forall i.$$



(a) **Gather step**: Each worker sends its i-th chunk to worker i. (b) **Average step**: Each worker averages all chunks it receives. (c) **Scatter step**: Each worker receives the i-th chunk from worker i.

Figure 3. Efficient system design for communication (compressed_allreduce)

Algorithm 1 1-bit Adam

```
1: Initialize: $\boldsymbol{x}_0$, learning rate $\gamma$, initial error $\boldsymbol{\delta} = \boldsymbol{0}$, $\boldsymbol{m}_0 = \boldsymbol{0}$, $\boldsymbol{v}_0 = \boldsymbol{0}$, number of total iterations $T$, warm-up steps $T_w$, two decaying factor $\beta_1$, $\beta_2$ and $\eta$ for Adam.
2: Running the original Adam for $T_w$ steps, then store the variance term (defined as $\boldsymbol{v}_t$ in (1)) $\boldsymbol{v}_{T_w}$.
3: for $t = T_w, \dots, T$ do
4:     (On $i$-th node)
5:     Randomly sample $\boldsymbol{\zeta}_t^{(i)}$ and compute local stochastic gradient $\boldsymbol{g}_t^{(i)} := \nabla F_i(\boldsymbol{x}_t^{(i)}, \boldsymbol{\zeta}_t^{(i)})$.
6:     Update the local momentum variable $\boldsymbol{m}_{t-1}$ according to $\boldsymbol{m}_t^{(i)} = \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t^{(i)}$.
7:     Compress $\boldsymbol{m}_t^{(i)}$ into $\hat{\boldsymbol{m}}_t^{(i)} = \boldsymbol{C}_\omega\left[\boldsymbol{m}_t^{(i)} + \boldsymbol{\delta}_{t-1}^{(i)}\right]$, and update the compression error by $\boldsymbol{\delta}_t^{(i)} = \boldsymbol{m}_t^{(i)} + \boldsymbol{\delta}_{t-1}^{(i)} - \hat{\boldsymbol{m}}_t^{(i)}$.
8:     Send the $\hat{\boldsymbol{m}}_t^{(i)}$ to the server.
9:     (On server)
10:    Take the average over all $\hat{\boldsymbol{m}}_t^{(i)}$ it receives and compress it into $\overline{\boldsymbol{m}}_t = \boldsymbol{C}_\omega\left[\frac{1}{n}\sum_{i=1}^n \hat{\boldsymbol{m}}_t^{(i)} + \overline{\boldsymbol{\delta}}_{t-1}\right]$, and update the compression error accordingly by $\overline{\boldsymbol{\delta}}_t = \frac{1}{n}\sum_{i=1}^n \hat{\boldsymbol{m}}_t^{(i)} + \overline{\boldsymbol{\delta}}_{t-1} - \overline{\boldsymbol{m}}_t$.
11:    Send $\overline{\boldsymbol{m}}_t$ to all the workers.
12:    (On $i$-th node)
13:    Set $\boldsymbol{m}_t = \overline{\boldsymbol{m}}_t$, and update local model $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \gamma \boldsymbol{m}_t / \sqrt{\boldsymbol{v}_{T_w}}$.
14: end for
15: Output: $\boldsymbol{x}$.
```

3. ***Bounded magnitude of error for*** $\mathcal{C}_\omega[\cdot]$***:*** *The magnitude of worker's local errors* $\boldsymbol{\delta}_t^{(i)}$ *and the server's global error* $\overline{\boldsymbol{\delta}}_t$, *are assumed to be bounded by a constant* ϵ

$$\sum_{k=1}^n \mathbb{E}_\omega \left\|\boldsymbol{\delta}_t^{(i)}\right\| \le \frac{\epsilon}{2}, \quad \sum_{i=1}^n \mathbb{E}_\omega \left\|\overline{\boldsymbol{\delta}}_t\right\| \le \frac{\epsilon}{2}, \quad \forall t, \forall i.$$

Next we present the main theorem for 1-bit Adam.

Theorem 1. *Under Assumption 1, for 1-bit Adam, we have the following convergence rate*

$$\left(1 - \frac{\gamma L}{v_{\min}} - \frac{2\gamma^2 L^2}{(1-\beta)^2 v_{\min}^2}\right)\sum_{t=0}^T \mathbb{E}\|\nabla f(\boldsymbol{x}_t)\|_V^2$$
$$\le \frac{2\mathbb{E}f(\boldsymbol{x}_0) - 2f(\boldsymbol{x}^*)}{\gamma} + \frac{6\gamma^2 L^2 \epsilon^2 T}{(1-\beta)^2 v_{\min}^3} +$$
$$\frac{L\gamma\sigma^2 T}{n v_{\min}} + \frac{2\gamma^2 L^2 \sigma^2 T}{n(1-\beta)^2 v_{\min}^2}, \tag{6}$$

where $V = diag\left(1/\boldsymbol{v}_{T_w}^{(1)}, 1/\boldsymbol{v}_{T_w}^{(2)}, \cdots, 1/\boldsymbol{v}_{T_w}^{(d)}\right)$ *is a diagonal matrix spanned by* $\boldsymbol{v}_{T_w}$ *and* $v_{\min} = \min\{\boldsymbol{v}_{T_w}^{(1)}, \boldsymbol{v}_{T_w}^{(2)}, \cdots, \boldsymbol{v}_{T_w}^{(d)}\}$ *is the mimimum value in* $\boldsymbol{v}_{T_w}$

Given the generic result in Theorem 1, we obtain the convergence rate for 1-bit Adam with appropriately chosen learning rate γ.

Corollary 1. *Under Assumption 1, for 1-bit Adam, choosing* $\gamma = \frac{1}{4L(v_{\min})^{-1} + \sigma\sqrt{\frac{T}{n}} + \epsilon^{\frac{2}{3}} T^{\frac{1}{3}} (v_{\min})^{-1}}$, *we have the following convergence rate*

$$\frac{1}{T v_{\min}} \sum_{t=0}^{T-1} \mathbb{E}\|\nabla f(\boldsymbol{x}_t)\|_V^2 \lesssim \frac{\sigma}{\sqrt{nT}} + \frac{\epsilon^{\frac{2}{3}}}{T^{\frac{2}{3}}} + \frac{1}{T},$$

where we treat $f(\boldsymbol{x}_1) - f^*$, β *and* L *as constants.*

This result suggests that: 1-bit Adam essentially admits the same convergence rate as distributed SGD in the sense that both of them admit the asymptotical convergence rate $O(1/\sqrt{nT})$, which means we can still achieve linear speedup w.r.t. the number of workers n.

6. Efficient system design for compressed communication

NVIDIA NCCL is an efficient and widely used communication library that has been tightly integrated in DL frameworks like PyTorch and TensorFlow. However, NCCL library cannot be used directly for performing communication based on 1-bit compression. This is because the collective communication primitives like Allreduce and Allgather are at a higher level of abstraction and can only perform data movement and/or simple operations like sum, min, max etc.

In addition, NCCL library (before v2.7) did not expose either an Alltoall primitive or any point-to-point (send/recv) communication primitives that can be used to implement an Alltoall. Thus for 1-bit Adam, we designed a custom collective primitive using Message Passing Interface (MPI). We call it "compressed allreduce" and it has three phases as shown in Figure 3: 1) The gather step, which we have implemented using the MPI_Alltoall (personalized exchange) primitive, 2) The average step, where 1-bit Adam computes the average of compressed local momentums, and 3) The scatter step, which we implement using MPI_Allgather. We develop two versions of compressed allreduce: 1) CUDA-Aware version that exploits GPUDirect features and requires CUDA-Aware libraries like MVAPICH2-GDR and 2) Basic version that can be used with any MPI library but copies data between GPU and CPU buffers. The CUDA-Aware version works only on systems with InfiniBand whereas the basic version can run on any system with Ethernet interconnect.

7. Experiments

We evaluate 1-bit Adam and existing approaches using BERT-Base, BERT-Large, SQuAD 1.1 and ResNet training tasks on up to 256 GPUs. We show that 1-bit Adam converges as fast as uncompressed Adam, and runs up to 3.3 times faster than uncompressed algorithms under limited bandwidth.

7.1. BERT pre-training and fine-tuning

Dataset and models We evaluate the convergence and performance of 1-bit Adam and uncompressed Adam for BERT-Base ($L = 12$, $H = 768$, $A = 12$, $110M$ params) and BERT-Large ($L = 24$, $H = 1024$, $A = 16$, $340M$ params) pre-training tasks. We use the same dataset as Devlin et al. (2019), which is a concatenation of Wikipedia and BooksCorpus with $2.5B$ and $800M$ words respectively. We use the GLUE fine-tuning benchmark(Wang et al., 2018) to evaluate the convergence of the BERT models trained by Adam and 1-bit Adam.

In addition, we also evaluate the convergence and performance of 1-bit Adam for SQuAD 1.1 fine-tuning task[3] using a pre-trained BERT model checkpoint from HuggingFace[4].

Hardware For all experiments in this Section 7.1 we use the two clusters described in Section 3.1. We use up to 256 GPUs for pre-training tasks and up to 32 GPUs for fine-tuning tasks.

Training parameters For BERT pre-training, the learning rate linearly increases to 4×10^{-4} as a warmup in the first $12.5K$ steps, then decays into 0.99 of the original after every 520 steps. We set the two parameters in Algorithm 1 as $\beta_1 = 0.9$ and $\beta_2 = 0.999$ for 1-bit Adam and Adam. For convergence test, we set total batch size as $4K$ for BERT-Base and BERT-Large. For performance test, we test different batch sizes. Table 2 summarizes the total number of steps for BERT sequence length 128 and 512 phases, together with the number of warmup steps for 1-bit Adam. We manually tuned the number of warmup steps for 1-bit Adam evaluations. On the other hand, we find that this configuration can be auto-tuned: First, the number of 1-bit Adam warmup steps should be no less than the number of learning rate warmup steps, since Adam's variance term is unstable during LR warmup. Second, we find that the ratio $\frac{\|\boldsymbol{v}_t\|_1}{\|\boldsymbol{v}_{t-\Delta}\|_1}$ (where $\|\cdot\|_1$ is the l_1 norm of the vector and we set $\Delta = \frac{1}{1-\beta_2}$) is a good indicator of how stable the variance term is. For BERT-Large pre-training seqlen 128, when we set a threshold of ≥ 0.96 for this ratio, the warmup will stop at step 22173, which is very close to our manualy tuned $23K$ warmup steps.

Table 2. Number of steps for BERT pre-training tasks.

	Seqlen 128 (warmup)	Seqlen 512 (warmup)
BERT-Base Adam	$118K$ (N/A)	$22K$ (N/A)
BERT-Base 1-bit Adam	$118K$ ($16K$)	$22K$ ($1.5K$)
BERT-Large Adam	$152K$ (N/A)	$10K$ (N/A)
BERT-Large 1-bit Adam	$152K$ ($23K$)	$10K$ ($1.5K$)

For GLUE benchmarks we use original Adam optimizer and perform single-task training on the dev set. We search over the hyperparameter space with batch sizes $\in \{8, 16\}$ and learning rates $\in \{1 \times 10^{-5}, 3 \times 10^{-5}, 5 \times 10^{-5}, 8 \times 10^{-5}\}$. Other setting are the same as pre-training task.

For SQuAD fine-tuning we use the same parameters as published by HuggingFace (batch size = 24, learning rate=$3e{-}5$, dropout=0.1, 2 epochs), except that we increase the batch size to 96 (using 32 GPUs). The first 400 steps out of total 1848 steps are used as the warmup stage for 1-bit Adam.

Convergence results Figure 4(a) presents the sample-wise convergence results. We use the BertAdam (Devlin et al., 2019) optimizer as the uncompressed baseline. For both BERT-Base and BERT-Large and for both sequence length phases, we find that 1-bit Adam provides the same convergence speed as baseline, while the communication volume is reduced into 6% of the original during the compression stage.

Table 3 presents the GLUE results using the checkpoints from our pre-training experiments. 1-bit Adam achieves similar accuracy compared to the uncompressed baseline and the numbers reported in previous work.

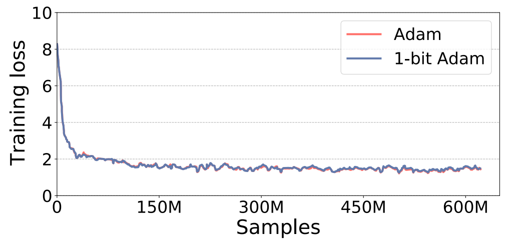
For SQuAD 1.1 fine-tuning task using checkpoint from

[3]https://rajpurkar.github.io/SQuAD-explorer/

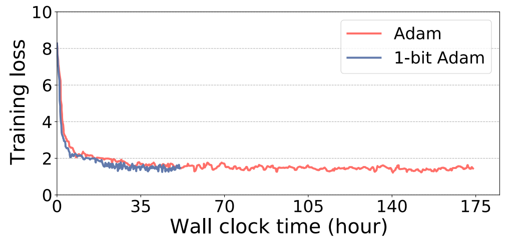
[4]https://github.com/huggingface/transformers

Table 3. GLUE development set results. BERT-Base/Large(original) results are from Devlin et al. (2019). BERT-Base/Large (uncompressed) results use the full-precision **BertAdam** with the same training parameters as the 1-bit Adam case. BERT-Base/Large (compressed) are the results using 1-bit Adam. The scores are the median scores over 10 runs.

Model	RTE	MRPC	CoLA	SST-2	QNLI	QQP	MNLI-(m/mm)
BERT-Base (original)	66.4	84.8	52.1	93.5	90.5	89.2	84.6/83.4
BERT-Base (uncompressed)	68.2	84.8	56.8	91.8	90.9	90.9	83.6/83.5
BERT-Base (compressed)	69.0	84.8	55.6	91.6	90.8	90.9	83.6/83.9
BERT-Large (original)	70.1	85.4	60.5	94.9	92.7	89.3	86.7/85.9
BERT-Large (uncompressed)	70.3	86.0	60.3	93.1	92.2	91.4	86.1/86.2
BERT-Large (compressed)	70.4	86.1	62.0	93.8	91.9	91.5	85.7/85.4



(a) Sample-wise



(b) Time-wise

Figure 4. Sample-wise and time-wise convergence speed for BERT-Large pre-training sequence length 128 using 64 GPUs on the Ethernet cluster. 1-bit Adam and Adam also achieve the same sample-convergence speed for BERT-Base pre-training.

HuggingFace, 1-bit Adam achieves similar F1 score (93.32) compared to the score reported by HuggingFace (93.33) using same number of samples and trainig parameters.

Performance results Computed as 1/(warmup ratio + (1 - warmup ratio)/16) for FP16 training, 1-bit Adam offers up to 5x less end-to-end communication volume for BERT-Base and BERT-Large. This leads to to 3.3x higher throughput for BERT-Large sequence length 128 pre-training and up to 2.9x higher throughput for SQuAD fine-tuning. This end-to-end throughput improvement is enabled by the 5.48x (Figure 5(a)) and 6.17x (Figure 5(c)) speedup observed during the compression stage. Figure 5(b) shows that 1-bit Adam also provides better scalability: Adam's throughput reaches peak at 32 GPUs on Ethernet, while 1-bit Adam's throughput keeps increasing until 128 GPUs. It is also worth mentioning that 1-bit Adam on Ethernet (4.1 Gbps effective bandwidth, 4 GPUs per node) is able to achieve comparable throughput as Adam on InfiniBand (near 100 Gbps effective bandwidth, 8 GPUs per node), which demonstrates 1-bit Adam's efficiency considering the hardware differences.

In Figure 4(b) we also measured the total training time of BERT-Large pre-training seqlen 128 when using batch size $4K$ on 64 GPUs on the Ethernet cluster. It takes 174.3 hours for baseline Adam to complete the training, while 1-bit Adam only needs 51.5 hours. This 3.4x speedup is consistent with the speedup computed based on the throughput analysis above.

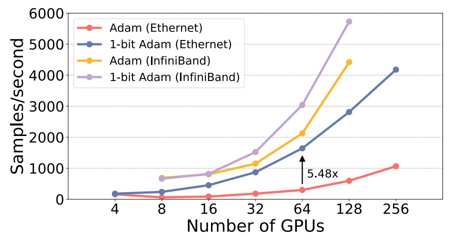
7.2. ResNet on CIFAR10 and ImageNet

To further evaluate the convergence speed of 1-bit Adam and related works, we train CIFAR10 using ResNet-18(He et al., 2016). The dataset has a training set of 50000 images and a test set of 10000 images, where each image is given one of the 10 labels. We run the experiments on 8 1080Ti GPUs where each GPU is used as one worker. The batch size on each worker is 128 and the total batch size is 1024.

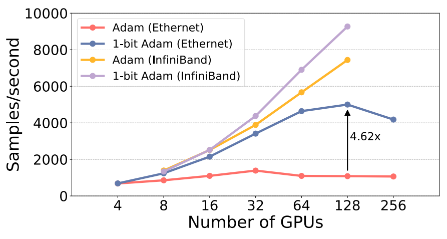
We evaluate five implementations for comparison: 1) Original SGD. 2) Original Adam (Kingma and Ba, 2014). 3) 1-bit Adam where we use 13 out of 200 epochs as warmup. 4) **1-bit Adam (32-bits)** where we do not compress the momentum while still freezing the variance. 5) **Adam(1-bit Naive)** where we compress the gradient instead of momentum, and don't freeze the variance. We set the learning rate as 1×10^{-1} for SGD and 1×10^{-4} for the other 4 cases. For all five cases, the learning rate is decayed into 10% of the original after every 100 epochs.

As illustrated in Figure 6, 1-bit Adam achieves similar convergence speed as Adam and 1-bit Adam (32-bits). SGD has a slightly slower convergence speed while Adam(1-bit Naive) is much worse. This and Section 3.2 demonstrate that existing compression method doesn't work for Adam. In the supplementary materials we further compare 1-bit Adam with other related works using ResNet-18.

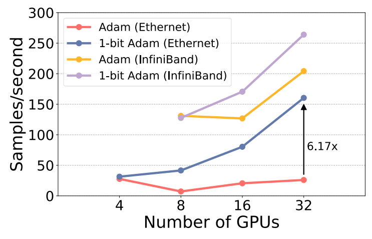
Moreover, to see how 1-bit Adam could speedup the training in this case, we report speedup results of training ResNet-152 on ImageNet (Russakovsky et al., 2015) using different numbers of GPUs, in Figure 7. As we can see that 1-bit Adam could potentially speedup the training especially when the bandwidth is limited.



(a) Bert-Large pre-training, batch size = number of GPUs × 16

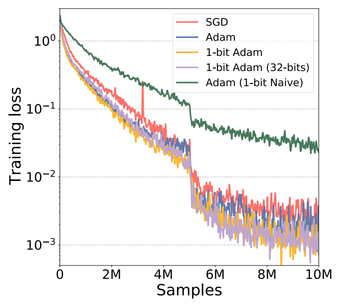


(b) Bert-Large pre-training, batch size = 4K

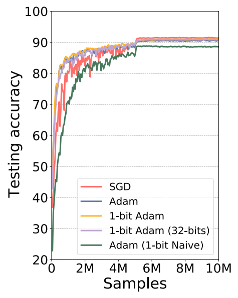


(c) SQuAD fine-tuning, batch size = number of GPUs × 3

Figure 5. Scalability of 1-bit Adam for BERT-Large pre-training sequence length 128 and SQuAD 1.1 fine-tuning on V100 GPUs. Adam lines represent the throughput at 1-bit Adam's warmup stage (i.e., baseline Adam's throughput). 1-bit Adam lines represent the throughput at compression stage. Annotations represent the highest speedup achieved in each figure. Note that this is the speedup between warmup and compression stage. The end-to-end speedup also depends on the percentage of warmup.



(a) Training loss



(b) Testing accuracy

Figure 6. Sample-wise convergence speed for ResNet-18.

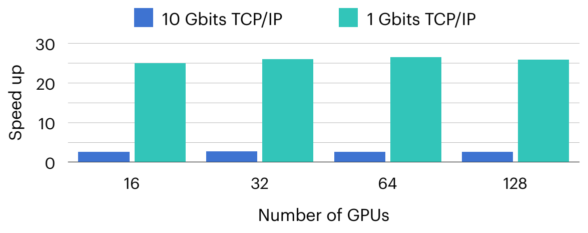
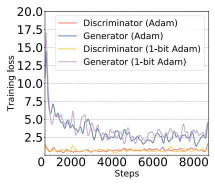


Figure 7. Speedup of ResNet-152 on ImageNet. Each server has 8 V100 GPUs interconnected by NVLink, servers are connected by 10Gbits or 1Gbits TCP/IP network.

7.3. Deep Convolutional Generative Adversarial Networks

To further understand the correctness of 1-bit Adam on more tasks, we apply it to the training of Generative Adversarial Networks (GAN). We choose Deep Convolutional GAN (Radford et al., 2015) as the model, which adopts convolutional and convolutional-transpose layers for the discriminator and generator. We use CelebFaces Attributes Dataset (CelebA) (Liu et al., 2015) as the training data, which contains more than 200K celebrity images. The task is to train the discriminator and generator in an adversarial way, such that the generator can create fake but vivid face images. Figure 8 shows the training loss and generated images by using original Adam optimizer and 1-bit Adam. The results show that 1-bit Adam can achieve almost the same training accuracy as the Adam optimizer.



(a) Training loss

(b) Generated images (Adam)

(c) Generated images (1-bit Adam)

Figure 8. Comparison of Adam and 1-bit Adam (20% warmup steps) for training Deep Convolutional Generative Adversarial Networks (DCGAN).

8. Conclusions

In this paper, we propose an error-compensated Adam preconditioned momentum SGD algorithm, 1-bit Adam, which provides both communication efficiency and Adam's convergence speed. Our theoretical analysis demonstrates that 1-bit Adam admits a linear speed w.r.t the number of workers in the network, and is robust to any compression method. We validate the performance of 1-bit Adam empirically on BERT, SQuAD and ResNet training tasks on up to 256 GPUs. Results show that 1-bit Adam converges as fast as uncompressed Adam, reduces communication volume by up to 5x, and runs up to 3.3 times faster than uncompressed algorithms. Beyond those results, it's interesting to see the performance of 1-bit Adam on wider variety of tasks, e.g., reinforcement learning, which we leave for the future work.

References

N. Agarwal, A. T. Suresh, F. X. X. Yu, S. Kumar, and B. McMahan. cpSGD: Communication-efficient and differentially-private distributed SGD. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 7564–7575. Curran Associates, Inc., 2018.

A. Alacaoglu, Y. Malitsky, P. Mertikopoulos, and V. Cevher. A new regret analysis for adam-type algorithms, 2020.

D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic. QSGD: Communication-Efficient SGD via gradient quantization and encoding. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 1709–1720. Curran Associates, Inc., 2017.

D. Basu, D. Data, C. Karakus, and S. Diggavi. Qsparse-local-sgd: Distributed sgd with quantization, sparsification and local computations. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 14695–14706. Curran Associates, Inc., 2019.

J. Bernstein, J. Zhao, K. Azizzadenesheli, and A. Anandkumar. signsgd with majority vote is communication efficient and byzantine fault tolerant. 10 2018.

S. Chaturapruek, J. C. Duchi, and C. Ré. Asynchronous stochastic convex optimization: the noise is in the noise and sgd don t care. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1531–1539. Curran Associates, Inc., 2015.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.

E. Gorbunov, D. Kovalev, D. Makarenko, and P. Richtárik. Linearly converging error compensated sgd, 2020.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

N. Ivkin, D. Rothchild, E. Ullah, V. braverman, I. Stoica, and R. Arora. Communication-efficient distributed sgd with sketching. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 13144–13154. Curran Associates, Inc., 2019.

P. Jiang and G. Agrawal. A linear speedup analysis of distributed deep learning with sparse and quantized communication. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 2530–2541. Curran Associates, Inc., 2018.

D. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

A. Koloskova*, T. Lin*, S. U. Stich, and M. Jaggi. Decentralized deep learning with arbitrary communication compression. In *International Conference on Learning Representations*, 2020.

Y. Li, M. Yu, S. Li, S. Avestimehr, N. S. Kim, and A. Schwing. Pipe-sgd: A decentralized pipelined sgd framework for distributed deep net training. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 8056–8067. Curran Associates, Inc., 2018.

X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5330–5340. Curran Associates, Inc., 2017.

R. Liu, T. Wu, and B. Mozafari. Adam with bandit sampling for deep learning, 2020.

Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

L. Luo, Y. Xiong, and Y. Liu. Adaptive gradient methods with dynamic bound of learning rate. In *International Conference on Learning Representations*, 2019.

T. T. Phuong and L. T. Phong. Distributed sgd with flexible gradient compression. *IEEE Access*, 8:64707–64717, 2020.

A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu. 1-bit stochastic gradient descent and application to data-parallel distributed training of speech dnns. In *Interspeech 2014*, September 2014.

Z. Shen, A. Mokhtari, T. Zhou, P. Zhao, and H. Qian. Towards more efficient stochastic decentralized learning: Faster convergence and sparse communication. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4624–4633, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

S. Shi, Q. Wang, K. Zhao, Z. Tang, Y. Wang, X. Huang, and X. Chu. A distributed synchronous sgd algorithm with global top-k sparsification for low bandwidth networks. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 2238–2247, 2019.

S. Shi, X. Zhou, S. Song, X. Wang, Z. Zhu, X. Huang, X. Jiang, F. Zhou, Z. Guo, L. Xie, R. Lan, X. Ouyang, Y. Zhang, J. Wei, J. Gong, W. Lin, P. Gao, P. Meng, X. Xu, C. Guo, B. Yang, Z. Chen, Y. Wu, and X. Chu. Towards scalable distributed training of deep learning on public cloud clusters. In *Proceedings of Machine Learning and Systems*, 2021.

R. Spring, A. Kyrillidis, V. Mohan, and A. Shrivastava. Compressing gradient optimizers via Count-Sketches. *Proceedings of the 36th International Conference on Machine Learning*, 97:5946–5955, 2019.

S. U. Stich. Local sgd converges fast and communicates little, 2019.

S. U. Stich, J.-B. Cordonnier, and M. Jaggi. Sparsified sgd with memory. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 4447–4458. Curran Associates, Inc., 2018.

J. Sun, T. Chen, G. Giannakis, and Z. Yang. Communication-efficient distributed learning via lazily aggregated quantized gradients. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 3370–3380. Curran Associates, Inc., 2019.

H. Tang, C. Yu, X. Lian, T. Zhang, and J. Liu. `DoubleSqueeze`: Parallel stochastic gradient descent with double-pass error-compensated compression. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6155–6165, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

T. Tieleman and G. Hinton. Rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 2011.

T. Vogels, S. P. Karimireddy, and M. Jaggi. Powersgd: Practical low-rank gradient compression for distributed optimization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 14259–14268. Curran Associates, Inc., 2019.

T. Vogels, S. P. Karimireddy, and M. Jaggi. Powergossip: Practical low-rank communication compression in decentralized deep learning, 2020.

A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium, Nov. 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-5446.

J. Wangni, J. Wang, J. Liu, and T. Zhang. Gradient sparsification for Communication-Efficient distributed optimization. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 1299–1309. Curran Associates, Inc., 2018.

W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li. Terngrad: Ternary gradients to reduce communication in distributed deep learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 1509–1519. Curran Associates, Inc., 2017.

C. Xie, S. Zheng, O. Koyejo, I. Gupta, M. Li, and H. Lin. Cser: Communication-efficient sgd with error reset, 2020.

M. Ye and E. Abbe. Communication-Computation efficient gradient coding. *Proceedings of the 35th International Conference on Machine Learning*, 80:5610–5619, 2018.

Y. Yu, J. Wu, and L. Huang. Double quantization for communication-efficient distributed optimization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 4438–4449. Curran Associates, Inc., 2019.

M. D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

H. Zhang, J. Li, K. Kara, D. Alistarh, J. Liu, and C. Zhang. ZipML: Training linear models with end-to-end low precision, and a little bit of deep learning. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 4035–4043, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

S. Zheng, Q. Meng, T. Wang, W. Chen, N. Yu, Z. Ma, and T. Liu. Asynchronous stochastic gradient descent with delay compensation for distributed deep learning. *CoRR*, abs/1609.08326, 2016.

S. Zheng, Z. Huang, and J. Kwok. Communication-efficient distributed blockwise momentum sgd with error-feedback. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 11450–11460. Curran Associates, Inc., 2019.

Supplementary

9. Bert Pre-training: speedup under different network bandwidths

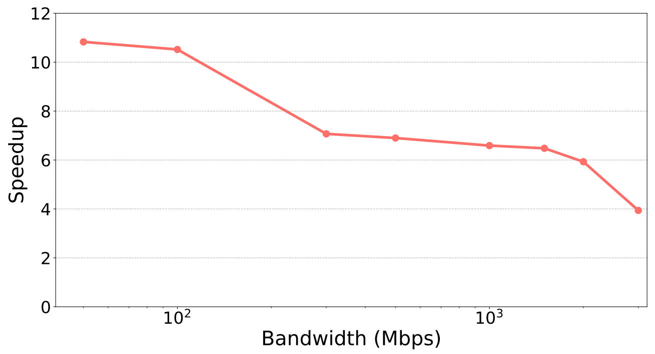


Figure 9. Throughput speedup (on 256 V100 GPUs) between Adam and 1-bit Adam compression stage for BERT-Large pre-training under different network bandwidths, from 50Mbits to 3Gbits. The x-axis is in log scale.

In Figure 9, we evaluate the throughput speedup between Adam and 1-bit Adam compression stage under different network conditions. Specifically, we use traffic control utility *tc* to shape the bandwidth from 50Mbits to 3Gbits on Ethernet. With the network going slow, 1-bit Adam compression stage can achieve a speedup up to $10.83\times$ over the uncompressed Adam ($6.59\times$ speedup at 1Gbits bandwidth and $5.93\times$ at 2Gbits bandwidth).

10. ResNet: comparing with additional communication efficient algorithms

Beyond Adam, there have been many communication efficient optimization algorithms proposed for SGD and Momentum SGD. We also compare the convergence speed of those algorithms with 1-bit Adam for training ResNet-18 on CIFAR10.

We evaluate five implementations for comparison:

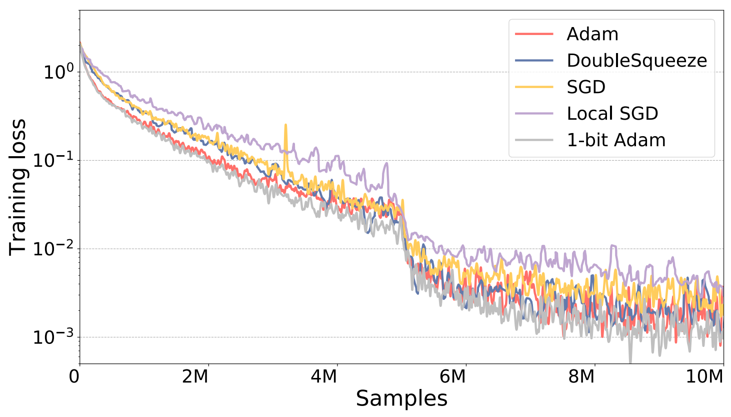
1. **DoubleSqueeze** (Tang et al., 2019): In DoubleSqueeze, the stochastic gradient $\boldsymbol{g}_t$ is compressed with error compensation. We use 1-bit compression here.
2. **Momentum SGD**: The updating rule of Momentum SGD admits

$$\boldsymbol{m}_{t+1} = \beta\boldsymbol{m}_t + (1-\beta)\boldsymbol{g}_t,$$
$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \gamma\boldsymbol{m}_{t+1},$$

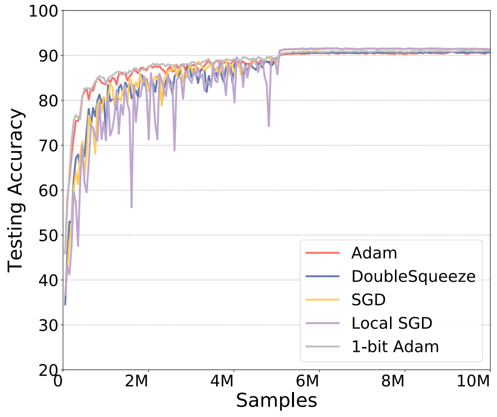
   where $\boldsymbol{g}_t$ is the stochastic gradient and $\boldsymbol{m}_t$ is the momentum.
3. **Error-Feedback Momentum SGD (EF Momentum SGD)** (Zheng et al., 2019): This algorithm is similar with Zheng et al. (2019), where the momentum $\boldsymbol{m}_t$ is being compressed with error compensation. We use 1-bit compression here.
4. **Local SGD** (Stich, 2019): In local SGD, the model would get updated using local gradient following SGD, and after every τ step, the model would be averaged over workers.
5. **Local SGD with Momentum**: In local SGD, the model would get updated using local gradient following Momentum SGD, and after every τ step, both model and momentum would be averaged over workers.

We set the momentum $\beta = 0.9$ here. In order to get the comparable communication reduction, we set $\tau = 4$ for Local SGD. The learning rate is grid searched from $\gamma \in \{0.5, 0.1, 0.01, 0.001\}$ for each algorithm except Adam and 1-bit Adam (in which we set learning rate $\gamma = 1 \times 10^{-4}$), and we find that $\gamma = 0.1$ works best for all of them.

In Figure 10 we compare the convergence speed of 1-bit Adam with DoubleSqueeze and Local SGD, and in Figure 11, we compare the convergence speed of 1-bit Adam with EF Momentum SGD and Local SGD with Momentum.

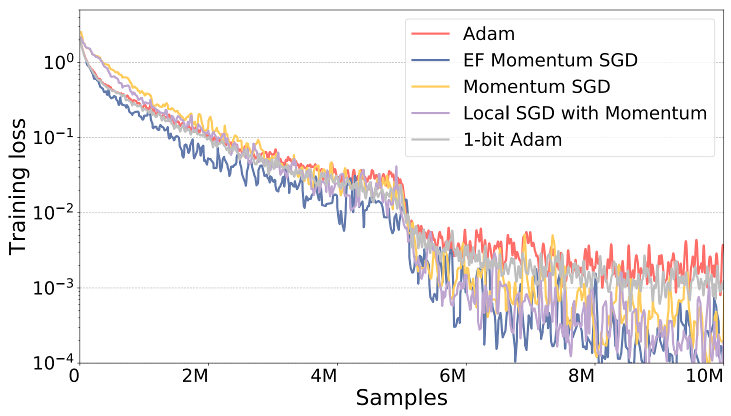


(a) Training loss

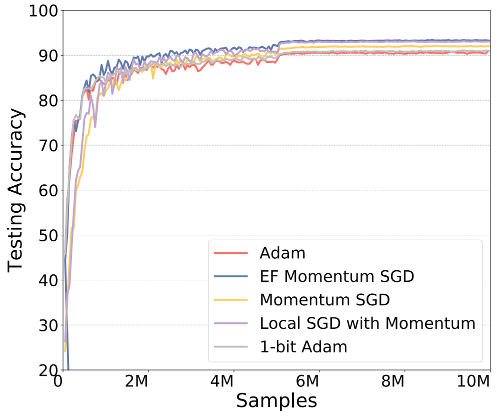


(b) Testing accuracy

Figure 10. Epoch-wise convergence speed for ResNet-18. We compare 1-bit Adam with SGD-type of communication efficient algorithms.



(a) Training loss



(b) Testing accuracy

Figure 11. Epoch-wise convergence speed for ResNet-18. We compare 1-bit Adam with Momentum SGD-type of communication efficient algorithms, and the momentum is set to be $\beta = 0.9$.

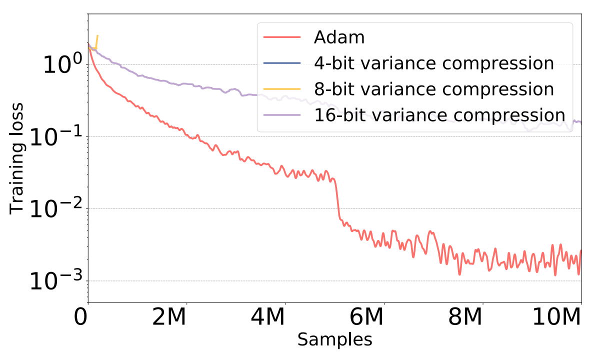
Notice that for training ResNet-18, both EF Momentum SGD and Local Momentum SGD admits a faster convergence speed than 1-bit Adam, this is because the uncompressed Momentum SGD runs faster than Adam.

Meanwhile, for Adam, we also evaluate the influence of the variance term to the convergence speed with some following tryouts:

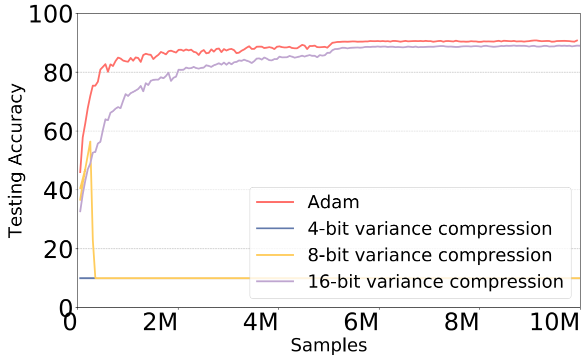
- **Adam with n-bits Variance Compression**: This algorithm would allreduce both the momentum term and variance term, with variance term being compressed into n-bits representation (Alistarh et al., 2017). This design is to see whether we could still achieve comparable convergence speed with the variance term being compressed. The results is in Figure 12.

- **Adam with Lazily Updated Variance**: Here we only allreduce the variance term without compression after every τ steps, and the the variance term would get updated continuously using local gradients. The results is in Figure 13.

Unfortunately, both methods fail to achieve a comparable convergence speed with Adam. Therefore the 1-bit Adam method proposed in the paper is the only solution we found that could provide the same convergence speed with vanilla Adam.

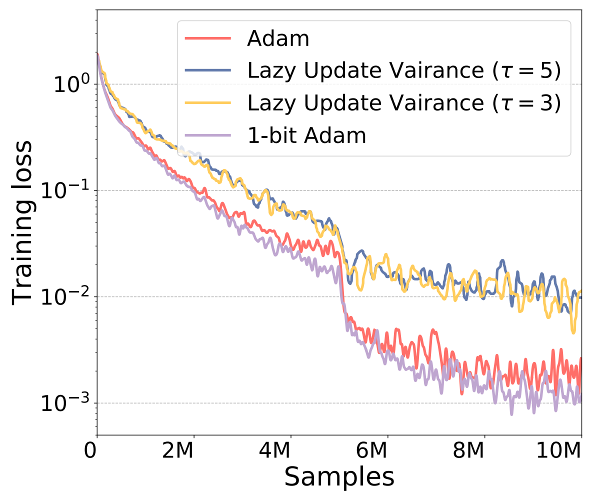


(a) Training loss

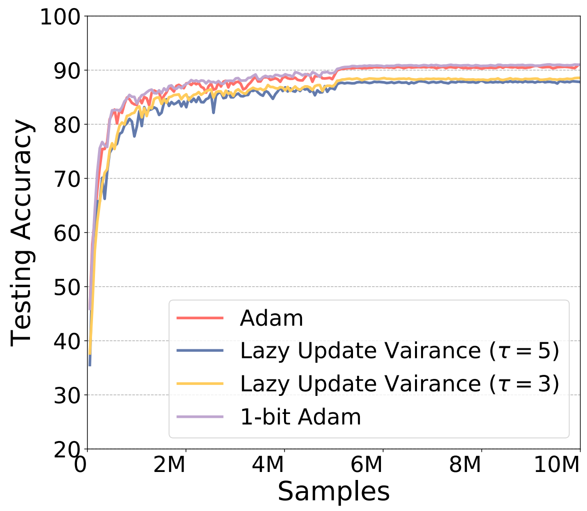


(b) Testing accuracy

Figure 12. ResNet-18 on CIFAR10. Momentum and variance are compressed into 1-bit and n-bit. When $n \leq 8$, the training cannot converge, so we do not include the result above.



(a) Training loss



(b) Testing accuracy

Figure 13. ResNet-18 on CIFAR10. The variance term would get averaged after every τ stpes.

11. Proof to the updating form

Since our algorithm is equivalent to running a parameter-server prototype communication on each chunk of the gradient, so below we will assume a parameter-server model (which means the tensor is not required to be divided into n chunks) for simplicity.

According to the algorithm description in Section 4.3, at iteration $t+1$, the updating step of the momentum term $\boldsymbol{m}_{t+1}$ can be divided into two steps:

1. Local Update and Compress: each worker locally update $\boldsymbol{m}_t$ and use the error-compensate strategy for compressing.

$$\begin{aligned}\boldsymbol{m}_t^{(i)} =& \beta\boldsymbol{m}_t + (1-\beta)\boldsymbol{g}_t^{(i)}\\ \boldsymbol{m}_{t+\frac{1}{2}}^{(i)} =& \mathcal{C}_\omega[\boldsymbol{m}_t^{(i)} + \boldsymbol{\delta}_t^{(i)}]\\ \boldsymbol{\delta}_{t+1}^{(i)} =& \boldsymbol{m}_t^{(i)} + \boldsymbol{\delta}_t^{(i)} - \boldsymbol{m}_{t+\frac{1}{2}}^{(i)}.\end{aligned}$$

2. All workers send its $\boldsymbol{m}_{t+\frac{1}{2}}^{(i)}$ to the server. The server takes the average over them and compress it again using

error-compensation.

$$\begin{aligned}\boldsymbol{m}_{t+\frac{1}{2}} =&\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{m}_{t+\frac{1}{2}}^{(i)}\\ \boldsymbol{m}_{t+1} =&\mathcal{C}_{\omega}[\boldsymbol{m}_{t+\frac{1}{2}}+\boldsymbol{\delta}_t]\\ \boldsymbol{\delta}_{t+1} =&\boldsymbol{m}_{t+\frac{1}{2}}+\boldsymbol{\delta}_t-\boldsymbol{m}_{t+1}.\end{aligned}$$

3. The server broadcast $\boldsymbol{m}_{t+1}$ to all workers, and all workers update the local model according to

$$\boldsymbol{x}_{t+1}=\boldsymbol{x}_t-\gamma\boldsymbol{m}_{t+1}\oslash\sqrt{\boldsymbol{v}_{T_w}^2}.$$

So actually the updating rule above can be summarized as

$$\begin{aligned}\boldsymbol{m}_{t+1} =&\mathcal{C}_{\omega}[\boldsymbol{m}_{t+\frac{1}{2}}+\boldsymbol{\delta}_t]\\ =&\boldsymbol{m}_{t+\frac{1}{2}}+\boldsymbol{\delta}_t-\boldsymbol{\delta}_{t+1}\quad\text{(from the definition of }\boldsymbol{\delta}_{t+1})\\ =&\frac{1}{n}\sum_{i=1}^{n}\mathcal{C}_{\omega}[\boldsymbol{m}_t^{(i)}+\boldsymbol{\delta}_t^{(i)}]+\boldsymbol{\delta}_t-\boldsymbol{\delta}_{t+1}\\ =&\frac{1}{n}\sum_{i=1}^{n}\left(\boldsymbol{m}_t^{(i)}+\boldsymbol{\delta}_t^{(i)}-\boldsymbol{\delta}_{t+1}^{(i)}\right)+\boldsymbol{\delta}_t-\boldsymbol{\delta}_{t+1}\quad\text{(from the definition of }\boldsymbol{\delta}_{t+1}^{(i)})\\ =&\beta\boldsymbol{m}_t+\frac{1-\beta}{n}\sum_{i=1}^{n}\boldsymbol{g}_t^{(i)}+\left(\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{\delta}_t^{(i)}+\boldsymbol{\delta}_t\right)-\left(\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{\delta}_{t+1}^{(i)}+\boldsymbol{\delta}_{t+1}\right).\end{aligned}$$

Denote

$$\begin{aligned}\overline{\boldsymbol{g}}_t =&\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{g}_t^{(i)}\\ \overline{\boldsymbol{\delta}}_t =&\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{\delta}_t^{(i)}+\boldsymbol{\delta}_t,\end{aligned}$$

the update rule of $\boldsymbol{m}_t$ can be summarized as

$$\boldsymbol{m}_t=\beta\boldsymbol{m}_{t-1}+(1-\beta)\overline{\boldsymbol{g}}_t+\overline{\boldsymbol{\delta}}_{t-1}-\overline{\boldsymbol{\delta}}_t,$$

and

$$\boldsymbol{x}_{t+1}=\boldsymbol{x}_t-\gamma V\boldsymbol{m}_t,$$

where $V=\text{diag}(1/\sqrt{v_1},1/\sqrt{v_2},\cdots,1/\sqrt{v_d})$ is the a diagonal matrix that spanned with $\boldsymbol{v}_{T_w}$.

12. Proof to Theorem 1

Notice that in for 1-bit Adam, the learning rate for each coordinate is different. In order to simplify our analysis, we instead consider another function that is defined as

$$H(\boldsymbol{z})=F(V^{\frac{1}{2}}\boldsymbol{z}),$$

also

$$h(\boldsymbol{z})=f(V^{\frac{1}{2}}\boldsymbol{z}),$$

where V is a diagonal matrix.

In this case we have

$$V^{\frac{1}{2}}\nabla f(V^{\frac{1}{2}}\boldsymbol{z}) = \nabla h(\boldsymbol{z}).$$

Therefore the updating rule of 1-bit Adam in the view of $h(\cdot)$ is

$$V^{\frac{1}{2}}\boldsymbol{z}_{t+1} = V^{\frac{1}{2}}\boldsymbol{z}_t - \gamma V^{\frac{1}{2}}\left(V^{\frac{1}{2}}\boldsymbol{m}_t\right).$$

It can be easily verified that

$$\begin{aligned}\boldsymbol{m}_t =&(1-\beta)\sum_{s=0}^{t}\beta^{t-s}\overline{\boldsymbol{g}}_s + \sum_{s=0}^{t}\beta^{t-s}(\overline{\boldsymbol{\delta}}_{s-1} - \overline{\boldsymbol{\delta}}_s)\\ =&(1-\beta)\sum_{s=0}^{t}\beta^{t-s}\frac{1}{n}\sum_{i=1}^{n}\nabla F(V^{\frac{1}{2}}\boldsymbol{z}_t;\xi_t^{(i)}) + \sum_{s=0}^{t}\beta^{t-s}(\overline{\boldsymbol{\delta}}_{s-1} - \overline{\boldsymbol{\delta}}_s)\end{aligned}$$

which means

$$\begin{aligned}V^{\frac{1}{2}}\boldsymbol{m}_t =&(1-\beta)\sum_{s=0}^{t}\beta^{t-s}\frac{1}{n}\sum_{i=1}^{n}V^{\frac{1}{2}}\nabla F(V^{\frac{1}{2}}\boldsymbol{z}_t;\xi_t^{(i)}) + \sum_{s=0}^{t}\beta^{t-s}V^{\frac{1}{2}}(\overline{\boldsymbol{\delta}}_{s-1} - \overline{\boldsymbol{\delta}}_s)\\ =&(1-\beta)\sum_{s=0}^{t}\beta^{t-s}\frac{1}{n}\sum_{i=1}^{n}\nabla H(V^{\frac{1}{2}}\boldsymbol{z}_t;\xi_t^{(i)}) + \sum_{s=0}^{t}\beta^{t-s}V^{\frac{1}{2}}(\overline{\boldsymbol{\delta}}_{s-1} - \overline{\boldsymbol{\delta}}_s)\\ =&(1-\beta)\sum_{s=0}^{t}\beta^{t-s}\overline{\boldsymbol{g}}_s(\boldsymbol{z}) + \sum_{s=0}^{t}\beta^{t-s}V^{\frac{1}{2}}(\overline{\boldsymbol{\delta}}_{s-1} - \overline{\boldsymbol{\delta}}_s),\end{aligned}$$

where $\overline{\boldsymbol{g}}_s(\boldsymbol{z})$ is the corresponding averaged stochastic gradient computed in the view of loss function $h(\cdot)$.

Then, if we define $\boldsymbol{m}_t(\boldsymbol{z}) = V^{\frac{1}{2}}\boldsymbol{m}_t$, the updating rule of $\boldsymbol{m}_t(z)$ admits

$$\boldsymbol{m}_t(\boldsymbol{z}) = \beta\boldsymbol{m}_{t-1}(\boldsymbol{z}) + (1-\beta)\overline{\boldsymbol{g}}_t(\boldsymbol{z}) + V^{\frac{1}{2}}\overline{\boldsymbol{\delta}}_{t-1} - V^{\frac{1}{2}}\overline{\boldsymbol{\delta}}_t, \tag{7}$$

and

$$\begin{aligned}V^{\frac{1}{2}}\boldsymbol{z}_{t+1} =&V^{\frac{1}{2}}\boldsymbol{z}_t - \gamma V^{\frac{1}{2}}\boldsymbol{m}_t(\boldsymbol{z})\\ \boldsymbol{z}_{t+1} =&\boldsymbol{z}_t - \gamma\boldsymbol{m}_t(\boldsymbol{z}).\end{aligned} \tag{8}$$

From (7) and (8) we shall see that using different learning rate for each coordinate is equivalent to optimizing a new loss function defined on scaling the original coordinate and using a uniform learning for all coordinates. Therefore below we first study the behavior of the error-compensated momentum SGD using a constant learning rate.

Below are some critical lemmas for the proof of Theorem 1.

Lemma 1. *Given two non-negative sequences $\{a_t\}_{t=1}^{\infty}$ and $\{b_t\}_{t=1}^{\infty}$ that satisfying*

$$a_t = \sum_{s=1}^{t}\rho^{t-s}b_s, \tag{9}$$

with $\rho \in [0,1)$, we have

$$D_k := \sum_{t=1}^{k}a_t^2 \leq \frac{1}{(1-\rho)^2}\sum_{s=1}^{k}b_s^2.$$

Proof. From the definition, we have

$$S_k = \sum_{t=1}^{k}\sum_{s=1}^{t}\rho^{t-s}b_s = \sum_{s=1}^{k}\sum_{t=s}^{k}\rho^{t-s}b_s = \sum_{s=1}^{k}\sum_{t=0}^{k-s}\rho^{t}b_s \leq \sum_{s=1}^{k}\frac{b_s}{1-\rho}, \tag{10}$$

$$\begin{aligned}
D_k =& \sum_{t=1}^{k}\sum_{s=1}^{t}\rho^{t-s}b_s\sum_{r=1}^{t}\rho^{t-r}b_r\\
=& \sum_{t=1}^{k}\sum_{s=1}^{t}\sum_{r=1}^{t}\rho^{2t-s-r}b_sb_r\\
\leq& \sum_{t=1}^{k}\sum_{s=1}^{t}\sum_{r=1}^{t}\rho^{2t-s-r}\frac{b_s^2+b_r^2}{2}\\
=& \sum_{t=1}^{k}\sum_{s=1}^{t}\sum_{r=1}^{t}\rho^{2t-s-r}b_s^2\\
\leq& \frac{1}{1-\rho}\sum_{t=1}^{k}\sum_{s=1}^{t}\rho^{t-s}b_s^2\\
\leq& \frac{1}{(1-\rho)^2}\sum_{s=1}^{k}b_s^2, \quad \text{(due to (10))}
\end{aligned}$$

which completes the proof. □

Lemma 2. *Under Assumption 1, for any sequence that follows the updating rule of*

$$\begin{aligned}
\boldsymbol{x}_{t+1} =& \boldsymbol{x}_t - \gamma\boldsymbol{m}_t\\
\boldsymbol{m}_t =& \beta\boldsymbol{m}_{t-1} + (1-\beta)\overline{\boldsymbol{g}}_t + \overline{\boldsymbol{\delta}}_{t-1} - \overline{\boldsymbol{\delta}}_t,
\end{aligned}$$

if

$$\begin{gathered}
\mathbb{E}\overline{\boldsymbol{g}}_t = \nabla f(\boldsymbol{x}_t), \quad \mathbb{E}\|\overline{\boldsymbol{g}}_t - \nabla f(\boldsymbol{x}_t)\|^2 \leq \frac{\sigma^2}{n}, \quad \mathbb{E}\|\overline{\boldsymbol{\delta}}_t\|^2 \leq \epsilon^2, \quad \forall t,\\
\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\| \leq L\|\boldsymbol{x}-\boldsymbol{y}\|, \quad \forall \boldsymbol{x}, \forall \boldsymbol{y},
\end{gathered}$$

then we can guarantee that

$$\begin{aligned}
&\left(1-\gamma L - \frac{2\gamma^2L^2}{(1-\beta)^2}\right)\sum_{t=0}^{T}\mathbb{E}\|\nabla f(\boldsymbol{x}_t)\|^2\\
\leq& \frac{2\mathbb{E}f(\boldsymbol{x}_1) - 2\mathbb{E}f(\boldsymbol{x}^*)}{\gamma} + \frac{6\gamma^2L^2\epsilon^2T}{(1-\beta)^2} + \frac{L\gamma\sigma^2T}{n} + \frac{2\gamma^2L^2\sigma^2T}{n(1-\beta)^2}
\end{aligned}$$

Proof. Instead of investigating $\boldsymbol{x}_t$ directly, we introduce the following sequence

$$\boldsymbol{y}_t = \boldsymbol{x}_t - \frac{\gamma}{1-\beta}(\boldsymbol{m}_t + \overline{\boldsymbol{\delta}}_{t-1}).$$

The updating rule of $\boldsymbol{y}_t$ admits

$$\begin{aligned}
\boldsymbol{y}_{t+1} - \boldsymbol{y}_t =& \boldsymbol{x}_{t+1} - \boldsymbol{x}_t - \frac{\gamma}{1-\beta}(\boldsymbol{m}_{t+1} - \boldsymbol{m}_t - \overline{\boldsymbol{\delta}}_{t+1} + \overline{\boldsymbol{\delta}}_t)\\
=& -\gamma\boldsymbol{m}_t - \frac{\gamma}{1-\beta}(\beta\boldsymbol{m}_t + (1-\beta)\boldsymbol{g}_t + \overline{\boldsymbol{\delta}}_{t-1} - \overline{\boldsymbol{\delta}}_t - \boldsymbol{m}_t + \overline{\boldsymbol{\delta}}_t - \overline{\boldsymbol{\delta}}_{t-1})\\
=& -\gamma\boldsymbol{g}_t.
\end{aligned}$$

Since $f(\cdot)$ is with L-Lipschitzian, we have

$$\begin{aligned}
\mathbb{E}f(\boldsymbol{y}_{t+1}) - \mathbb{E}f(\boldsymbol{y}_t) \leq & \mathbb{E}\left\langle \nabla f(\boldsymbol{y}_t), \boldsymbol{y}_{t+1} - \boldsymbol{y}_t \right\rangle + \frac{L}{2}\mathbb{E}\left\|\boldsymbol{y}_{t+1} - \boldsymbol{y}_t\right\|^2 \\
= & -\gamma\mathbb{E}\left\langle \nabla f(\boldsymbol{y}_t), \boldsymbol{g}_t \right\rangle + \frac{L\gamma^2}{2}\mathbb{E}\|\boldsymbol{g}_t\|^2 \\
= & -\gamma\mathbb{E}\left\langle \nabla f(\boldsymbol{y}_t), \nabla f(\boldsymbol{x}_t) \right\rangle + \frac{L\gamma^2}{2}\mathbb{E}\|\boldsymbol{g}_t\|^2 \\
= & -\frac{\gamma}{2}\mathbb{E}\|\nabla f(\boldsymbol{x}_t)\|^2 - \frac{\gamma}{2}\mathbb{E}\|\nabla f(\boldsymbol{y}_t)\|^2 + \frac{\gamma}{2}\mathbb{E}\|\nabla f(\boldsymbol{x}_t) - \nabla f(\boldsymbol{y}_t)\|^2 + \frac{L\gamma^2}{2}\mathbb{E}\|\boldsymbol{g}_t\|^2 \\
\leq & -\frac{\gamma}{2}\mathbb{E}\|\nabla f(\boldsymbol{x}_t)\|^2 + \frac{\gamma L^2}{2}\mathbb{E}\|\boldsymbol{x}_t - \boldsymbol{y}_t\|^2 + \frac{L\gamma^2}{2}\mathbb{E}\|\boldsymbol{g}_t\|^2 \\
= & -\frac{\gamma}{2}\mathbb{E}\|\nabla f(\boldsymbol{x}_t)\|^2 + \frac{\gamma^3 L^2}{2}\mathbb{E}\left\|\frac{\boldsymbol{m}_t}{1-\beta} + \frac{\overline{\boldsymbol{\delta}}_{t-1}}{1-\beta}\right\|^2 + \frac{L\gamma^2}{2}\mathbb{E}\|\boldsymbol{g}_t\|^2 \\
\leq & -\frac{\gamma}{2}\mathbb{E}\|\nabla f(\boldsymbol{x}_t)\|^2 + \frac{\gamma^3 L^2}{(1-\beta)^2}\mathbb{E}\|\boldsymbol{m}_t\|^2 + \frac{\gamma^3 L^2}{(1-\beta)^2}\mathbb{E}\|\overline{\boldsymbol{\delta}}_{t-1}\|^2 + \frac{L\gamma^2}{2}\mathbb{E}\|\boldsymbol{g}_t\|^2 \\
\leq & -\frac{\gamma}{2}\mathbb{E}\|\nabla f(\boldsymbol{x}_t)\|^2 + \frac{\gamma^3 L^2}{(1-\beta)^2}\mathbb{E}\|\boldsymbol{m}_t\|^2 + \frac{\gamma^3 L^2\epsilon^2}{(1-\beta)^2} + \frac{L\gamma^2}{2}\mathbb{E}\|\boldsymbol{g}_t\|^2 \\
\leq & -\frac{\gamma}{2}\mathbb{E}\|\nabla f(\boldsymbol{x}_t)\|^2 + \frac{\gamma^3 L^2}{(1-\beta)^2}\mathbb{E}\|\boldsymbol{m}_t\|^2 + \frac{\gamma^3 L^2\epsilon^2}{(1-\beta)^2} + \frac{L\gamma^2}{2}\mathbb{E}\|\nabla f(\boldsymbol{x}_t)\|^2 + \frac{L\gamma^2\sigma^2}{2n}.
\end{aligned}$$

Summing up the equation above from $t = 0$ to $t = T$ we get

$$\mathbb{E}f(\boldsymbol{y}_{T+1}) - \mathbb{E}f(\boldsymbol{y}_0) \leq -\frac{(1-\gamma L)\gamma}{2}\sum_{t=0}^{T}\mathbb{E}\|\nabla f(\boldsymbol{x}_t)\|^2 + \frac{\gamma^3 L^2}{(1-\beta)^2}\sum_{t=0}^{T}\mathbb{E}\|\boldsymbol{m}_t\|^2 + \frac{\gamma^3 L^2\epsilon^2 T}{(1-\beta)^2} + \frac{L\gamma^2\sigma^2 T}{2n},$$

which can be rewritten into

$$(1-\gamma L)\sum_{t=0}^{T}\mathbb{E}\|\nabla f(\boldsymbol{x}_t)\|^2 \leq \frac{2\mathbb{E}f(\boldsymbol{y}_0) - 2\mathbb{E}f(\boldsymbol{y}_{T+1})}{\gamma} + \frac{2\gamma^2 L^2}{(1-\beta)^2}\sum_{t=0}^{T}\mathbb{E}\|\boldsymbol{m}_t\|^2 + \frac{2\gamma^2 L^2\epsilon^2 T}{(1-\beta)^2} + \frac{L\gamma\sigma^2 T}{n}. \tag{11}$$

Notice that we have

$$\boldsymbol{m}_t = (1-\beta)\sum_{s=0}^{t}\beta^{t-s}\overline{\boldsymbol{g}}_s + \sum_{s=0}^{t}\beta^{t-s}(\overline{\boldsymbol{\delta}}_{s-1} - \overline{\boldsymbol{\delta}}_s)$$

which by using Lemma 1, we have

$$\sum_{t=0}^{T}\|\boldsymbol{m}_t\|^2 \leq \sum_{t=0}^{T}\|\boldsymbol{g}_t\|^2 + \frac{2}{(1-\beta)^2}\sum_{t=0}^{T}\|\overline{\boldsymbol{\delta}}_t\|^2 \leq \sum_{t=0}^{T}\|\nabla f(\boldsymbol{x}_t)\|^2 + \frac{\sigma^2 T}{n} + \frac{2\epsilon^2 T}{(1-\beta)^2}. \tag{12}$$

Combing (11) and (12) together we get

$$\begin{aligned}
& \left(1 - \gamma L - \frac{2\gamma^2 L^2}{(1-\beta)^2}\right)\sum_{t=0}^{T}\mathbb{E}\|\nabla f(\boldsymbol{x}_t)\|^2 \\
\leq & \frac{2\mathbb{E}f(\boldsymbol{y}_0) - 2\mathbb{E}f(\boldsymbol{y}_{T+1})}{\gamma} + \frac{6\gamma^2 L^2\epsilon^2 T}{(1-\beta)^2} + \frac{L\gamma\sigma^2 T}{n} + \frac{2\gamma^2 L^2\sigma^2 T}{n(1-\beta)^2} \\
\leq & \frac{2\mathbb{E}f(\boldsymbol{x}_1) - 2\mathbb{E}f(\boldsymbol{x}^*)}{\gamma} + \frac{6\gamma^2 L^2\epsilon^2 T}{(1-\beta)^2} + \frac{L\gamma\sigma^2 T}{n} + \frac{2\gamma^2 L^2\sigma^2 T}{n(1-\beta)^2}.
\end{aligned}$$

□

Proof to Theorem 1 Since using a per-coordinate learning rate for loss function $f(\cdot)$ is equivalent to use a constant learning for all coordinates but for loss function $h(\cdot)$, the only two thing that change are

- **Different L-Lipschitzian coefficient**: the L-Lipschitzian coefficient for $h(\cdot)$ is

$$\begin{aligned}
\|\nabla h(\boldsymbol{x}) - \nabla h(\boldsymbol{y})\|^2 =& \left\| V^{\frac{1}{2}} \nabla f(V^{\frac{1}{2}}\boldsymbol{x}) - V^{\frac{1}{2}} \nabla f(V^{\frac{1}{2}}\boldsymbol{y}) \right\|^2 \\
=& \left\| \nabla f(V^{\frac{1}{2}}\boldsymbol{x}) - \nabla f(V^{\frac{1}{2}}\boldsymbol{y}) \right\|_V^2 \\
\leq& L^2 \left\| V^{\frac{1}{2}}\boldsymbol{x} - V^{\frac{1}{2}}\boldsymbol{y} \right\|_V^2 \\
=& L^2 \|\boldsymbol{x} - \boldsymbol{y}\|_{V^2}^2 \\
\leq& L^2 V_{\max}^2 \|\boldsymbol{x} - \boldsymbol{y}\|^2.
\end{aligned}$$

  Therefore the effective L-Lipschitzian coefficient of $h(\boldsymbol{x})$ is $LV_{\max}$

- **Different definition of $\overline{\boldsymbol{\delta}}_t$**: from (7) we shall see that actually the compression error in the view of $h(\cdot)$ is $V^{\frac{1}{2}}\overline{\boldsymbol{\delta}}_t$, so in this case we have

$$\mathbb{E}\|V^{\frac{1}{2}}\overline{\boldsymbol{\delta}}_t\|^2 \leq V_{\max}\epsilon^2$$

Proof. From Lemma 2, we have

$$\begin{aligned}
&\left(1 - \gamma L - \frac{2\gamma^2 L^2 V_{\max}^2}{(1-\beta)^2}\right) \sum_{t=0}^{T} \mathbb{E}\|\nabla h(\boldsymbol{z}_t)\|^2 \\
\leq& \frac{2\mathbb{E}f(\boldsymbol{x}_0) - 2\mathbb{E}f(\boldsymbol{x}^*)}{\gamma} + \frac{6\gamma^2 L^2 \epsilon^2 V_{\max}^3 T}{(1-\beta)^2} + \frac{L\gamma V_{\max}\sigma^2 T}{n} + \frac{2\gamma^2 L^2 \sigma^2 V_{\max}^2 T}{n(1-\beta)^2},
\end{aligned}$$

which by using $\nabla h(\boldsymbol{z}_t) = V^{\frac{1}{2}}\nabla f(\boldsymbol{x}_t)$, it becomes

$$\begin{aligned}
&\left(1 - \gamma L V_{\max} - \frac{2\gamma^2 L^2 V_{\max}^2}{(1-\beta)^2}\right) \sum_{t=0}^{T} \mathbb{E}\|\nabla f(\boldsymbol{x}_t)\|_V^2 \\
\leq& \frac{2\mathbb{E}f(\boldsymbol{x}_0) - 2\mathbb{E}f(\boldsymbol{x}^*)}{\gamma} + \frac{6\gamma^2 L^2 \epsilon^2 V_{\max}^3 T}{(1-\beta)^2} + \frac{L\gamma V_{\max}\sigma^2 T}{n} + \frac{2\gamma^2 L^2 \sigma^2 V_{\max}^2 T}{n(1-\beta)^2},
\end{aligned}$$

Since $V_{\max} = \frac{1}{\sqrt{v_{\min}}}$, therefore the equation above becomes

$$\begin{aligned}
&\left(1 - \frac{\gamma L}{v_{\min}} - \frac{2\gamma^2 L^2}{(1-\beta)^2 v_{\min}^2}\right) \sum_{t=0}^{T} \mathbb{E}\|\nabla f(\boldsymbol{x}_t)\|_V^2 \\
\leq& \frac{2\mathbb{E}f(\boldsymbol{x}_0) - 2\mathbb{E}f(\boldsymbol{x}^*)}{\gamma} + \frac{6\gamma^2 L^2 \epsilon^2 T}{(1-\beta)^2 v_{\min}^3} + \frac{L\gamma\sigma^2 T}{n v_{\min}} + \frac{2\gamma^2 L^2 \sigma^2 T}{n(1-\beta)^2 v_{\min}^2},
\end{aligned}$$

□

13. Proof to Corollary 1

Proof. By choosing $\gamma = \frac{1-\beta}{4LV_{\max} + \sigma\sqrt{\frac{T}{n}} + T^{\frac{1}{3}}\epsilon^{\frac{2}{3}}}$, we can guarantee that

$$1 - \gamma L - \frac{2\gamma^2 L^2 V_{\max}^2}{(1-\beta)^2} \geq \frac{1}{2}.$$

So (6) leads to

$$\begin{aligned}\sum_{t=0}^{T}\mathbb{E}\|\nabla f(\boldsymbol{x}_t)\|_V^2 \leq & \frac{2\left(\mathbb{E}f(\boldsymbol{y}_0)-f(\boldsymbol{y}^*)\right)}{(1-\beta)}\left(4LV_{\max}+\sigma\sqrt{\frac{T}{n}}+T^{\frac{1}{3}}\epsilon^{\frac{2}{3}}\right) \\ & + \left((1-\beta)L\sqrt{T}+2L^2V_{\max}^2\right)\frac{\sigma}{\sqrt{n}}+6L^2\epsilon^{\frac{2}{3}}T^{\frac{1}{3}}V_{\max}^3 \\ \frac{1}{T}\sum_{t=0}^{T}\mathbb{E}\|\nabla f(\boldsymbol{x}_t)\|_V^2 \leq & \frac{2\left(\mathbb{E}f(\boldsymbol{y}_0)-f(\boldsymbol{y}^*)\right)}{(1-\beta)}\left(\frac{4LV_{\max}}{T}+\frac{\sigma}{\sqrt{nT}}+T^{-\frac{2}{3}}\epsilon^{\frac{2}{3}}\right) \\ & + \left((1-\beta)L+\frac{2L^2V_{\max}^2}{\sqrt{T}}\right)\frac{\sigma}{\sqrt{nT}}+6L^2\epsilon^{\frac{2}{3}}T^{-\frac{2}{3}}V_{\max}^3.\end{aligned}$$

Treating $f(\boldsymbol{y}_1)-f^*$, β and L as constants, from the inequality above we get

$$\frac{1}{T}\sum_{t=0}^{T}\mathbb{E}\|\nabla f(\boldsymbol{x}_t)\|^2 \lesssim \frac{\sigma}{\sqrt{nT}}+\frac{\epsilon^{\frac{2}{3}}}{T^{\frac{2}{3}}}+\frac{1}{T}.$$

It completes the proof. □