

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

Hanlin Tang^{1,2}, Shaoduo Gan³, Ammar Ahmad Awan¹, Samyam Rajbhandari¹, Conglong Li¹, Xiangru Lian², Ji Liu², Ce Zhang³, and Yuxiong He¹

¹Microsoft

²Department of Computer Science, University of Rochester

³Department of Computer Science, ETH Zurich

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 (after a warmup phase) and can be used as a fixed precondition for 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

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 [Wangni et al., 2018, Zhang et al., 2017], decentralization [Koloskova* et al., 2020, Li et al., 2018, Lian et al., 2017], and asynchronous communication [Chaturapruek et al., 2015, Zheng et al., 2016].

One widely used strategy for alleviating the communication overhead is gradient compression. Before communication, the original gradient \mathbf{g} will be compressed into $\mathcal{C}_\omega[\mathbf{g}]$, where $\mathcal{C}_\omega[\cdot]$ ² is the compress operator. 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

¹This work was done during Hanlin Tang’s internship at Microsoft

² $\mathcal{C}_\omega[\cdot]$ could also include randomness.

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).

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 communication as well as the 1-bit Adam optimizer has been open sourced in a deep learning optimization library called DeepSpeed³.

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 [Ivkin et al., 2019, Yu et al., 2019]. Previous work has applied a range of techniques such as quantization, sparsification, and sketching [Agarwal et al., 2018, Alistarh et al., 2017, Spring et al., 2019, Ye and Abbe, 2018]. The compression is mostly assumed to be unbiased [Jiang and Agrawal, 2018, Shen et al., 2018, Wangni et al., 2018, Wen et al., 2017, Zhang et al., 2017]. 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.

³<https://github.com/microsoft/DeepSpeed>, <https://www.deepspeed.ai/>

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 [Basu et al., 2019, Ivkin et al., 2019, Phuong and Phong, 2020, Shi et al., 2019, Sun et al., 2019, Vogels et al., 2019, Yu et al., 2019, Zheng 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] (**RESprop**), 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 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,

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%

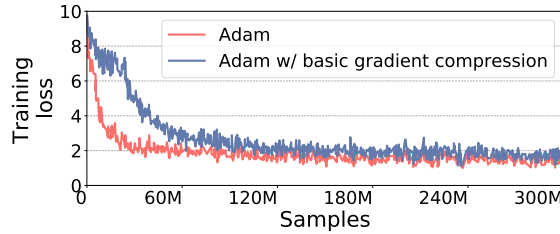


Figure 1: Training loss for BERT-Large pre-training using vanilla Adam and Adam with error compensated gradient compression.

and update both the momentum and variance based on the compressed gradient. We compare the BERT-Large pre-training (sequence 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 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 new 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 \mathbf{g} to update the model \mathbf{x} , **Adam** uses two auxiliary variables \mathbf{m} and \mathbf{v} for the update. The mathematical updating rule of original **Adam** can be summarized as:

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

Here \mathbf{x}_t is the model at t -iteration, $\mathbf{g}_t = \nabla F(\mathbf{x}_t; \zeta_t)$ 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 \mathbf{m}_t as the momentum term and \mathbf{v}_t as the variance term. Notice that when \mathbf{v}_t is changed into a constant \mathbf{v} , then **Adam** becomes equivalent to **Momentum SGD** under a coordinate-dependent learning rate $\frac{\gamma}{\sqrt{\mathbf{v} + \eta}}$.

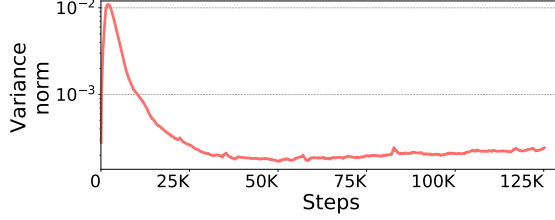


Figure 2: Norm of fused variance for BERT-Large pre-training using vanilla Adam. The y-axis is in log scale.

To investigate the non-linear gradient dependency challenge, we analyze **Adam**’s variance during BERT-Large pre-training (sequence 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.

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:

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

where d is the dimension of the input model \mathbf{x} , \mathcal{D}_i is the data distribution of individual data sample $\zeta^{(i)}$ on the i -th worker, $F(\mathbf{x}; \zeta)$ 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).
- $f_i(\mathbf{x}) := \mathbb{E}_{\zeta^{(i)} \sim \mathcal{D}_i} F(\mathbf{x}; \zeta^{(i)})$.
- $\|\cdot\|$ denotes the ℓ_2 norm for vectors and the spectral norm for matrices.
- $\|X\|_A := \text{Tr}(X^\top A X)$.
- $\mathbf{C}_\omega(\cdot)$ denotes the randomized compressing operator, where ω denotes the random variable. One example is the randomized quantization operator, for example, $\mathbf{C}_\omega(0.7) = 1$ with probability 0.7 and $\mathbf{C}_\omega(0.7) = 0$ with probability 0.3.
- $\sqrt{\cdot}$ 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.
- $(\mathbf{x})^2$ denotes the element-wise square operation if \mathbf{x} is a vector.
- $\frac{\mathbf{a}}{\mathbf{b}}$ or \mathbf{a}/\mathbf{b} denotes the element-wise division operation if both \mathbf{a} and \mathbf{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

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

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

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

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

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

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

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

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 \mathbf{v}_t with history error being cancelled.

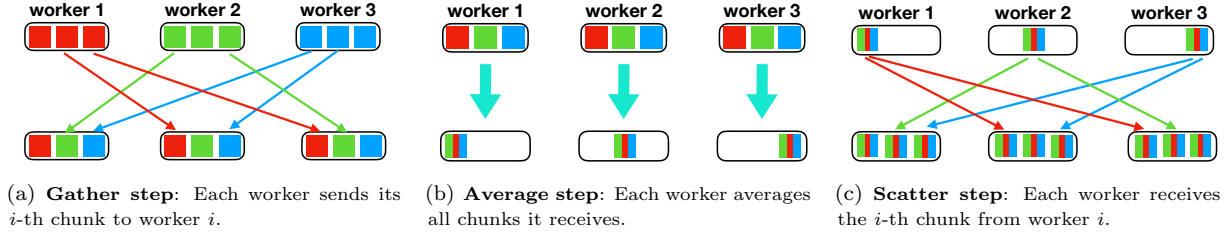


Figure 3: Efficient system design for communication (compressed_allreduce)

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

$$C \left[\mathbf{g}_t + \frac{\gamma_t}{\gamma_{t-1}} \delta_{t-1} \right],$$

instead of adding back δ_{t-1} directly. In this case, if we view $\frac{\gamma_t}{\sqrt{\mathbf{v}_t} + \eta}$ 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

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

The problem is that we cannot get the value of \mathbf{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 \mathbf{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 $\frac{\gamma}{\sqrt{\mathbf{v}_{T_w}}}$. Therefore here we consider the **Adam**-based warm-up phase as a way to find a good precondition variance term \mathbf{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

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

Algorithm 1 1-bit Adam

- 1: **Initialize:** \mathbf{x}_0 , learning rate γ , initial error $\bar{\delta} = \mathbf{0}$, $\mathbf{m}_0 = \mathbf{0}$, $\mathbf{v}_0 = \mathbf{0}$, number of total iterations T , warm-up steps T_w , two decaying factor β_1 and β_2 for **Adam**.
 - 2: Running the original **Adam** for T_w steps, then store the variance term (defined as \mathbf{v}_t in (1)) \mathbf{v}_{T_w} .
 - 3: **for** $t = T_w, \dots, T$ **do**
 - 4: **(On i -th node)**
 - 5: Randomly sample $\xi_t^{(i)}$ and compute local stochastic gradient $\mathbf{g}_t^{(i)} := \nabla F_i(\mathbf{x}_t^{(i)}, \xi_t^{(i)})$.
 - 6: Update the local momentum variable \mathbf{m}_{t-1} according to $\mathbf{m}_t^{(i)} = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t^{(i)}$.
 - 7: Compress $\mathbf{m}_t^{(i)}$ into $\hat{\mathbf{m}}_t^{(i)} = \mathcal{C}_\omega [\mathbf{m}_t^{(i)} + \delta_{t-1}^{(i)}]$, and update the compression error by $\delta_t^{(i)} = \mathbf{m}_t^{(i)} + \delta_{t-1}^{(i)} - \hat{\mathbf{m}}_t^{(i)}$.
 - 8: Send the $\hat{\mathbf{m}}_t^{(i)}$ to the server.
 - 9: **(On server)**
 - 10: Take the average over all $\hat{\mathbf{m}}_t^{(i)}$ it receives and compress it into $\bar{\mathbf{m}}_t = \mathcal{C}_\omega [\frac{1}{n} \sum_{i=1}^n \hat{\mathbf{m}}_t^{(i)} + \bar{\delta}_{t-1}]$, and update the compression error accordingly by $\bar{\delta}_t = \frac{1}{n} \sum_{j=1}^n \mathcal{C}_\omega [\mathbf{m}_t^{(j)}] + \bar{\delta}_{t-1} - \bar{\mathbf{m}}_t$.
 - 11: Send $\bar{\mathbf{m}}_t$ to all the workers.
 - 12: **(On i -th node)**
 - 13: Set $\mathbf{m}_t = \bar{\mathbf{m}}_t$, and update local model $\mathbf{x}_{t+1} = \mathbf{x}_t - \gamma \mathbf{m}_t / \sqrt{\mathbf{v}_{T_w}}$.
 - 14: **end for**
 - 15: **Output:** \mathbf{x} .
-

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

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

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

$$\sum_{k=1}^n \mathbb{E}_\omega \|\delta_t^{(i)}\| \leq \frac{\epsilon}{2}, \quad \sum_{i=1}^n \mathbb{E}_\omega \|\bar{\delta}_t\| \leq \frac{\epsilon}{2}, \quad \forall t, \forall i.$$

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

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

where $V = \text{diag}(1/v_{T_w}^{(1)}, 1/v_{T_w}^{(2)}, \dots, 1/v_{T_w}^{(d)})$ is a diagonal matrix spanned by \mathbf{v}_{T_w} and $v_{\min} = \min\{v_{T_w}^{(1)}, v_{T_w}^{(2)}, \dots, v_{T_w}^{(d)}\}$ is the minimum value in \mathbf{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 $\gamma = \frac{1}{4L(v_{\min})^{-1} + \sigma \sqrt{\frac{1}{n}} + \epsilon^{\frac{2}{3}} T^{\frac{1}{3}} (v_{\min})^{-1}}$, we have the following convergence rate

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

where we treat $f(\mathbf{x}_1) - f^*$, 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-18 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 task⁴ using a pre-trained BERT model checkpoint from HuggingFace⁵.

Hardware 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**.

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.

⁴<https://rajpurkar.github.io/SQuAD-explorer/>

⁵<https://github.com/huggingface/transformers>

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)

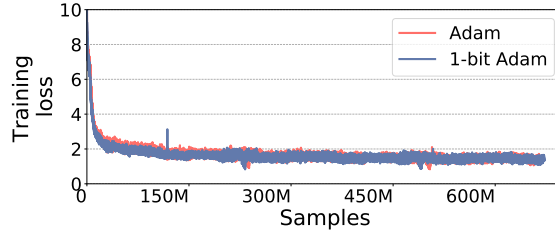


Figure 4: Epoch-wise convergence speed for BERT-Large pre-training sequence length 128. **1-bit Adam** and **Adam** also achieve the same convergence speed for BERT-Base pre-training.

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 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 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 training parameters.

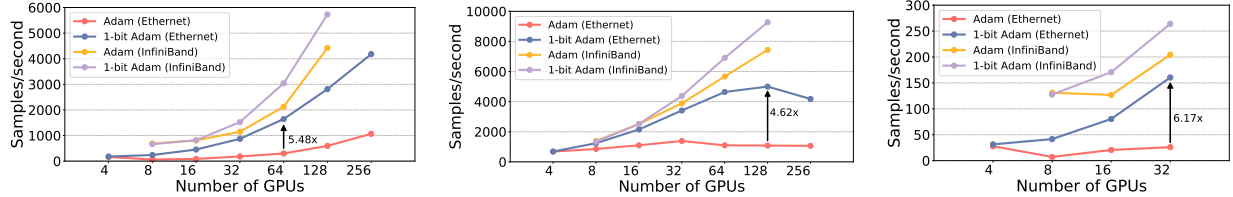
Performance results Computed as $1/(\text{warmup ratio} + (1 - \text{warmup ratio})/16)$ for FP16 training, **1-bit Adam** offers up to 5x less end-to-end communication volume for BERT-Base and BERT-Large. This leads to 3.3x higher throughput for BERT-Large sequence length 128 pre-training and up to 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 Ehternet, 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.

7.2 ResNet on CIFAR10

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.

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) Bert-Large pre-training, batch size = number of GPUs \times 16 (b) Bert-Large pre-training, batch size = 4K (c) SQuAD fine-tuning, batch size = number of GPUs \times 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.

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.

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.

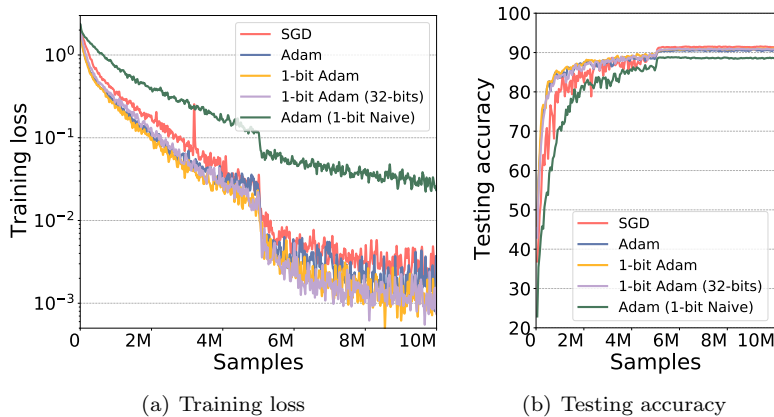


Figure 6: Epoch-wise convergence speed for ResNet-18.

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.
- 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. URL <http://jmlr.org/papers/v12/duchi11a.html>.
- 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. Iykin, 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. URL <https://openreview.net/forum?id=SkgGCKrKvH>.
- 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.
- L. Luo, Y. Xiong, and Y. Liu. Adaptive gradient methods with dynamic bound of learning rate. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=Bkg3g2R9FX>.
- T. T. Phuong and L. T. Phong. Distributed sgd with flexible gradient compression. *IEEE Access*, 8: 64707–64717, 2020.
- 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.
- 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, 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.

- 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. URL <https://www.aclweb.org/anthology/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. URL <http://arxiv.org/abs/1212.5701>.
- 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.