SLAMB: Accelerated Large Batch Training with Sparse Communication

Hang Xu [1] **Wenxuan Zhang** [1] **Jiawei Fei** [1] **Yuzhe Wu** [2] **TingWen Xie** [2] **Jun Huang** [2] **Yuchen Xie** [2]
Mohamed Elhoseiny [1] **Panos Kalnis** [1]

Abstract

Distributed training of large deep neural networks requires frequent exchange of massive data between machines, thus communication efficiency is a major concern. Existing compressed communication methods are either not compatible with large batch optimization algorithms, or do not provide sufficient speedup in large scale. In this paper, we combine sparsification-based gradient compression with the layer-wise adaptive moments optimizer for large batch training (LAMB). We propose SLAMB, a novel communication-efficient optimizer that supports large batch sizes and scales to thousands of GPUs. SLAMB employs momentum masking, local error compensation, and element-wise adaptive rescaling to achieve accurate layer-wise weight updates, which translates to fast convergence for very large batches. Our empirical results show that, compared to the state-of-the-art, SLAMB transmits half the amount of data in large-batch BERT pre-training, without sacrificing accuracy. Moreover, SLAMB achieves excellent scalability in large computing infrastructures. For instance, SLAMB with 128 GPUs reduces the training time of Swin Transformer pre-training on ImageNet to 5.35 hours, which is 2 hours faster than the state-of-the-art. At the extreme, we trained BERT-XL (2.8B parameters) on 1,024 NVIDIA A100 GPUs, where SLAMB achieved 90% scaling efficiency.

1. Introduction

Modern deep neural networks (DNNs) are becoming unprecedentedly large in various machine learning fields, including computer vision, natural language processing, and many others. For example, BERT (Devlin et al., 2018) has 340M parameters and GPT-3 (Brown et al., 2020) has up to 175B parameters. In order to train such models within reasonable time, large-scale distributed training is commonly utilized. Unfortunately, synchronous stochastic gradient descent (SGD) and its variants do not scale-up well due to two challenges: (*i*) using SGD with very large batch size typically degrades generalization accuracy; and (*ii*) scaling efficiency is severely compromised due to communication overhead.

The generalization issue of large-scale training can be addressed by layer-wise adaptive large batch training techniques, such as LARS (You et al., 2018) and LAMB (You et al., 2020). For instance, LAMB can scale the batch size of Adam (Kingma & Ba, 2015) from 512 to 64K for BERT pre-training, without affecting accuracy. However, communication is still a major performance bottleneck, due to the frequent gradient synchronization. We find that LAMB suffers from low scaling efficiency in large-scale distributed training. As illustrated in Table 1, by scaling LAMB to 1,024 GPUs, the scaling efficiency over the baseline (i.e., Adam on 8 GPUs) is only 53%; essentially, almost half of the 1,024 GPUs are heavily underutilized. To alleviate this, compressed communication has been proposed to accelerate the training speed by reducing the volume of the exchanged data. 1-bit LAMB (Li et al., 2022) combines 1-bit quantized compression with LAMB. However, due to its warm-up strategy, 1-bit LAMB reduces the volume of communicated data only by 4.6 times; in Table 1 we show that the resulting scaling efficiency on 1,024 GPUs is 80.9%.

In this paper, we address these challenges by combining large batch optimization (i.e., LAMB) with sparse communication and propose a novel communication-efficient large batch optimization algorithm, called *SLAMB*[1]. Our method scales efficiently to thousands of GPUs, without compromising accuracy. Our contributions are:

- We develop a novel communication efficient layer-wise adaptive algorithm SLAMB for large batch DNN optimization. SLAMB employs momentum masking, local

[1]King Abdullah University of Science and Technology, Thuwal, Kingdom of Saudi Arabia [2]Meituan, Beijing, China. Correspondence to: Hang Xu <hang.xu@kaust.edu.sa>, Panos Kalnis <panos.kalnis@kaust.edu.sa>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

[1]SLAMB: Sparse Layer-wise Adaptive Moments optimizer for large Batch training. Code is available at https://github.com/hangxu0304/SLAMB

Table 1. Comparison of scaling efficiency of different optimization algorithms on BERT-XLarge (2.7 billion parameters) pre-training task on wikipedia dataset (128 seqlen). We measure the throughput (samples per second) on a dedicated cluster with up to 1024 NVIDIA A100 GPUs, where each node is equipped with 8 GPUs and 100 Gbit/s inter-node network connection.

Algorithm	Batch size	#GPU	Throughput (samples/s)	Speedup	Scaling efficiency	Volume Reduction	Large batch	Compressed communication	Compression at all steps
Adam	512	8	435	1.0 ×	-	-			
LAMB	64K	1024	29520	67.8 ×	53.0%	1.0 ×	✓		
1-bit LAMB	64K	1024	45083	103.6 ×	80.9%	4.6 ×	✓	✓	
SLAMB	64K	1024	50412	115.8 ×	90.5%	9.1 ×	✓	✓	✓

error compensation and element-wise adaptive rescaling to ensure correct layer-wise scaling with sparse communication. To the best of our knowledge, our algorithm is the first to combine *sparsification*-based compression with large-batch training.

- We provide convergence analysis for SLAMB in non-convex settings; we show that SLAMB achieves the same convergence rate as LAMB.

- We evaluate the performance of SLAMB in popular language modeling and image classification tasks, namely BERT pre-training and Swin Transformer training. In both tasks, SLAMB with large batch training reduces the communication volume by up to 9.1 times, without compromising accuracy. For comparison, the existing state-of-the-art (i.e., 1-bit LAMB) reduces the volume of communicated data for BERT pre-training, only by up to 4.6 times.

- We demonstrate the superiority of SLAMB on large-scale GPU clusters. On a 128-GPU NVIDIA V100 cluster, SLAMB executes the Swin Transformer training task in 5.3 hours, which is around 2 hours faster than the state-of-the-art (i.e., 1-bit LAMB). Moreover, in Table 1 we show that SLAMB executes the pre-training task of BERT-XLarge (2.7B parameters) on a cluster with 1,024 NVIDIA A100 GPUs, with a scaling efficiency of 90.5%.

2. Related Work

Data Parallel DNN Training. The most common distributed deep learning is data parallel training via SGD, which enables each worker to process in parallel a different subset (mini-batch) of the training data. At each iteration, each worker runs forward and backward propagation independently to generate gradients. Then, all workers combine the local gradients to produce an averaged gradient that is applied to the model, prior to the next iteration. The gradient aggregation can be done synchronously or asynchronously. This paper focuses on the synchronous case, which is a widespread use due to its convergence guarantees (Zinkevich et al., 2010).

Large Batch Training. Parallelizing DNN training workloads on thousands of machines inevitably results to large batch sizes. (Hoffer et al., 2017; Keskar et al., 2017; Krizhevsky, 2014; Li et al., 2014) found that naïvely increasing the batch size, often results in performance degradation. Several works alleviate this issue by carefully hand-tuning training hyper-parameters, like learning rate (LR) and momentum (Goyal et al., 2017; Li, 2017; You et al., 2018; Shallue et al., 2019). Linear scaling (Bottou et al., 2018) and square root scaling (Jastrzebski et al., 2017) are two commonly used LR heuristics. Using LR warm-up and linear scaling, (Goyal et al., 2017) managed to train ResNet-50 with batch size 8192 without loss in generalization accuracy. To further increase the batch size, LARS (You et al., 2017) and LAMB (You et al., 2020) propose to adaptively scale the LR for each layer of the model. This adaptive strategy enables 32K batch size for ImageNet training and 64K batch size for BERT training. By combining LAMB with Nesterov Momentum and linear-constant LR scheduler, LANS (Zheng et al., 2020) managed to train BERT with 96K batch size. We focus on LAMB in this paper.

Compressed Communication. Compressed communication methods aim to reduce the number of transmitted bits by applying a carefully chosen gradient compression scheme, while achieving similar accuracy as uncompressed methods. For example, *quantization* methods (Alistarh et al., 2017; Wen et al., 2017; Strom, 2015; Seide et al., 2014; Xu et al., 2021a) reduce the amount of communication volume by reducing the per-element bit-width. *Sparsification* methods (Lin et al., 2018; Wangni et al., 2018; Alistarh et al., 2018; Stich et al., 2018; Xu et al., 2021b) instead send a subset of gradient elements, which can be selected by various selection algorithms such as hard-threshold, Top-k and Random-k. By properly setting training parameters, gradients can be quantized to 1-bit (Seide et al., 2014) or be sparsified by up to 0.1% (Lin et al., 2018) while not affecting the training accuracy. Recent works find that directly applying existing compression methods in more complex optimizers (e.g. Adam, LAMB) leads to slow convergence and low accuracy, thus requiring manual adaptation. 1-bit Adam (Tang et al., 2021) and 1-bit LAMB (Li et al., 2022) propose a novel two-stage compression strategy to ensure the same convergence speed as Adam and LAMB when using 1-bit

compression. Unlike previous work that uses quantization, in this paper we focus on combining sparsification-based compression with LAMB.

3. Motivation and Insights

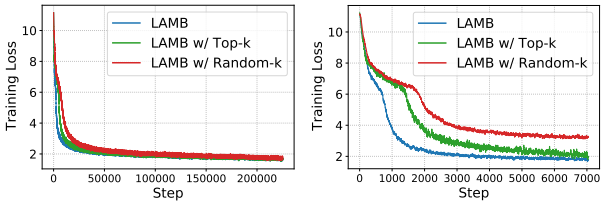
In this section, we investigate the limitations of existing methods and the challenges of applying gradient sparsification in LAMB. We start by revisiting the LAMB algorithm. The updating rule of LAMB can be summarized as:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2)(g_t)^2 \\ m_t &= m_t/(1-\beta_1^t), v_t = v_t/(1-\beta_2^t) \\ u_t &= \frac{m_t}{\sqrt{v_t}+\epsilon} \\ x_{t+1} &= x_t - \eta_t \frac{\phi(\|x_t\|)}{\|u_t\|} u_t \end{aligned} \tag{1}$$

Here, g_t, m_t, v_t, x_t denote the mini-batch stochastic gradient, momentum, variance, and model parameters for *each layer* of the neural network at step t. β_1, β_2 are the decaying parameters. η is the learning rate. ϵ is a small constant to avoid division by zero. ϕ is a scaling function which is usually configured as a clipping function $\phi(z) = \min\{\max\{z, a\}, b\}$. For simplicity, we omit the weight decay term. Intuitively, LAMB ensures the norm of the model update is of the same order as that of the parameter for each layer. This usually leads to faster convergence in large-batch training.

1-bit LAMB. 1-bit LAMB proposes a novel way to combine 1-bit gradient compression with LAMB, since naively combining 1-bit compression with LAMB or Adam leads to slower convergence (Li et al., 2022; Tang et al., 2021). The main idea is that, during training, the non-linear term in LAMB, e.g., $\sqrt{v_t}$, or $\frac{\phi(\|x_t^{(i)}\|)}{\|u_t^{(i)}\|}$, becomes stable after a certain number of steps. Hence, by freezing these terms, we can use LAMB in the same way as momentum SGD (Goyal et al., 2017). However, in order to find a stable value for these terms, we need to employ vanilla LAMB as a warm-up for a certain number of steps, before entering its compression stage. (Li et al., 2022) reports that the warm-up stage accounts for 16.7% - 19.2% of the total steps in BERT pre-training task. This motivates us to develop a new compression strategy that does not rely on freezing the non-linear terms of LAMB.

LAMB with naive sparsification. To explore compression strategy without freezing non-linear terms, we start by implementing the naive sparsification methods (Top-k, Random-k) within LAMB and conduct the experiment on BERT pre-training task (see details in Algorithm 2 in Appendix). The results are shown in Figure 1. We find that,



(a) 2K batch size (b) 64K batch size

Figure 1. Loss curve of BERT-Base pre-training using naive gradient sparsification in LAMB at different batch sizes (2K, 64K). We set compression ratio k=0.1% in (a) and k=10% in (b).

for small batch size 2K, by synchronizing only 0.1% of gradients, Top-k and Random-k convergences to the similar loss value of LAMB, though the convergence rate is slightly slower in the initial stage. However, when the batch size is increased to 64K, even if we increase the compression ratio k to 10%, both methods converge much slower than LAMB. Such convergence degradation indicates that gradient sparsification methods do not generalize well for extremely large batch-size settings.

To find out the fundamental reason behind the issue, we examine the updating rule of LAMB together with gradient sparsification. Notice that LAMB uses the momentum term m_t instead of the original gradient g_t to compute the model updates. Since $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$, where $\beta_1 = 0.9$ in most cases, the gradient g_t in current step only contributes to a fraction of m_t. This means the model update term u_t mainly comes from the history momentum m_{t-1}. Therefore, only synchronizing the compressed gradient is not sufficient for momentum-based optimizers. To overcome this, (Lin et al., 2018) proposed to compress and synchronize the momentum term in their method. Inspired by this, we also employ *masked momentum synchronization* in SLAMB to address this issue. See section 4.1.

LAMB with naive error compensation. We further investigate the compatibility between error compensation mechanism and LAMB. Error compensation has been widely adopted in various compressed communication methods (Seide et al., 2014; Karimireddy et al., 2019; Lin et al., 2018) to improve the convergence rate due to the existence of compression error. The basic idea is that, for vanilla SGD, the compression error in the current step can be added to the model updates in the next step, so that the history error terms get cancelled over iterations. The update rule can be written as:

$$x_{t+1} = x_t - \eta(\mathcal{C}[g_t + \delta_{t-1}]) = x_t - \eta(g_t - \underbrace{\delta_t + \delta_{t-1}}_{cancelled}) \tag{2}$$

where $\mathcal{C}[\cdot]$ denotes the compression operation and $\delta_{t-1} = (g_{t-1} + \delta_{t-2}) - \mathcal{C}[g_{t-1} + \delta_{t-2}]$ is the compression error in $t-1$ step. We can see that error compensation works for vanilla

SGD because the model update term $x_{t+1} - x_t$ depends linearly on compressed gradient $\mathcal{C}(g_t)$. However, this is not the case for LAMB, as the model update term $u_t = \frac{m_t}{\sqrt{v_t}+\epsilon}$ indicates that $\sqrt{v_t}$ varies over iterations. Therefore, using naive error compensation in LAMB would cause a larger error instead of getting the error canceled. Motivated by this, in SLAMB we use *local error compensation* to avoid the accumulation of history errors. See section 4.1.

4. Algorithm

Problem setup. In this paper, we study non-convex stochastic optimization problems of the form:

$$\min_{x\in\mathbb{R}^d} \left\{ f(x) := \frac{1}{n}\sum_{i\in[n]} f^{(i)}(x) \right\}, \tag{3}$$

where $f^{(i)} : \mathbb{R}^d \to \mathbb{R}$ is a smooth non-convex function for all $i \in [n] := \{1, 2, \ldots, n\}$ representing the loss function on each node. d is the dimensionality of the input model x.

Notations and definitions. We use the following notations throughout the paper:

- $[x]_j$ is the j-th element of vector x.
- $\|\cdot\|$ denotes the l_2 norm of a given vector, unless otherwise specified.
- $\odot$ denotes the dot product between two vectors.
- $\neg$ denotes element-wise NOT operation for a given 0/1 vector.
- $\sqrt{x}$ and $(x)^2$ are element-wise square root and square, respectively, if x is a vector.
- $\frac{x}{y}$ or x/y is element-wise division if x, y are vectors.

4.1. SLAMB algorithm

Based on the investigation of naive sparsification in LAMB, we propose SLAMB, summarized in Algorithm 1. SLAMB differs from the original LAMB algorithm in three aspects: sparse communication, adaptive scaling function, and model synchronization. The differences are marked by comments in Algorithm 1. We explain each of them next.

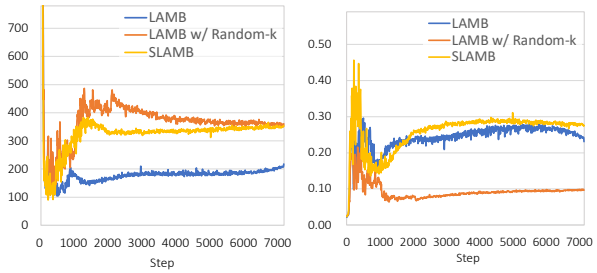
Sparse communication and local error compensation. Unlike LAMB performs full gradient synchronization, SLAMB only synchronizes a subset of the elements in momentum and gradient (line 8,9 in Algo. 1). We use a 0/1 mask vector M_t to select which elements to synchronize, where $[M_t]_j \sim \text{Bernoulli}(k)$. Note that synchronization requires an identical mask among all nodes. After communication, the synchronized part needs to be compensated by the local error, which is the unsynchronized part and

Algorithm 1 SLAMB

1: **Input:** $x_1 \in \mathbb{R}^d$, number of nodes n, learning rate $\{\eta_t\}_{t=1}^T$, weight decay λ, parameters $0 < \beta_1, \beta_2, \beta_3, k < 1$, $\epsilon > 0$, scaling function ϕ, synchronization interval H
2: **(On each node i)**
3: Set $m_0^{(i)} = \mathbf{0}$, $v_0^{(i)} = \mathbf{0}$, $c_0^{(i)} = \mathbf{1}$
4: **for** $t = 1$ **to** T **do**
5: Compute local mini-batch stochastic gradient $g_t^{(i)}$
6: Generate 0/1 mask vector M_t by Random-k selection
7: $m_t^{(i)} = \beta_1 m_{t-1}^{(i)} + (1-\beta_1) g_t^{(i)}$
8: $m_t^{(i)} = m_t^{(i)} \odot \neg M_t + \frac{1}{n}\sum_{i=1}^n [m_t^{(i)} \odot M_t]$ {**Sync** m}
9: $g_t^{(i)} = g_t^{(i)} \odot \neg M_t + \frac{1}{n}\sum_{i=1}^n [g_t^{(i)} \odot M_t]$ {**Sync** g}
10: $v_t^{(i)} = \beta_2 v_{t-1}^{(i)} + (1-\beta_2)(g_t^{(i)})^2$
11: $m_t^{(i)} = m_t^{(i)}/(1-\beta_1^t)$, $v_t^{(i)} = v_t^{(i)}/(1-\beta_2^t)$
12: $u_t^{(i)} = \frac{m_t^{(i)}}{\sqrt{v_t^{(i)}}+\epsilon} + \lambda x_t^{(i)}$
13: $c_t^{(i)} = \mathbf{1} \odot M_t + \beta_3 c_{t-1}^{(i)} \odot \neg M_t$ {**Update staleness**}
14: **for** each layer ℓ **do**
15: $\phi_{max} = \phi(\frac{\|x_t^{(\ell,i)} \odot M_t^{(\ell)}\| \|u_t^{(\ell,i)}\|}{\|u_t^{(\ell,i)} \odot M_t^{(\ell)}\|})$
16: $\phi_{min} = \phi(\frac{\|x_t^{(\ell,i)} \odot \neg M_t^{(\ell)}\| \|u_t^{(\ell,i)}\|}{\|u_t^{(\ell,i)} \odot \neg M_t^{(\ell)}\|})$
17: $\tilde{\phi} = \phi_{max} c_t^{(\ell,i)} + \phi_{min}(1 - c_t^{(\ell,i)})$ {**Rescale** ϕ}
18: $\tilde{\eta}_t = \eta_t c_t^{(\ell,i)} + \frac{\eta_t}{\sqrt{n}}(1 - c_t^{(\ell,i)})$ {**Rescale** η}
19: $x_{t+1}^{(\ell,i)} = x_t^{(\ell,i)} - \tilde{\eta}_t \frac{\tilde{\phi}}{\|u_t^{(\ell,i)}\|} u_t^{(\ell,i)}$
20: **end for**
21: **if** $t \in \{H, 2H, 3H, ..., T\}$ **then**
22: $x_{t+1} = \frac{1}{n}\sum_{i=1}^n x_{t+1}^{(i)}$ {**Sync** x}
23: **end if**
24: **end for**
25: **Output:** x

can be selected by $\neg M_t$. Now each node has a partially synchronized local momentum $m_t^{(i)}$ and gradient $g_t^{(i)}$. This leads to a partially synchronized model update $u_t^{(i)}$ (line 12 in Algo. 1.). However, if we continue the model updating by the same scaling rule as LAMB, our algorithm will suffer from slow convergence. We find that $\|u_t^{(i)}\|$ becomes larger than the fully synchronized one in LAMB, thus the scaling coefficients diminish a lot; see Figure 2. This is mainly due to the local error compensation, where the high variance of the unsynchronized part in $u_t^{(i)}$ results in a larger $\|u_t^{(i)}\|$. In addition, we find that synchronizing $m_t^{(i)}$ is more efficient than synchronizing $g_t^{(i)}$ in terms of variance reduction in $u_t^{(i)}$. Therefore, in practice we skip the synchronization of $g_t^{(i)}$ as it does not affect the accuracy. Please refer to Appendix H Table 13 for experiments validation and Appendix C for convergence analysis.

Adaptive scaling function. Since we know $\|u_t^{(i)}\|$ is biased, we can fix it by modifying the scaling function term $\phi(\|x_t^{(i)}\|)$ (lines 15-17 in Algo. 1.). Intuitively, if we use



(a) norm of model update (b) scaling coefficient

Figure 2. The norm of model update ($\|u_t^{(i)}\|$) and the scaling coefficients (LAMB: $\frac{\phi(\|x_t^{(i)}\|)}{\|u_t^{(i)}\|}$, SLAMB: $\frac{\tilde{\phi}}{\|u_t^{(i)}\|}$) of the first attention layer in BERT-Base pre-training (seqlen=128).

the synchronized part of $u_t^{(i)}$ and $\phi(\|x_t^{(i)}\|)$ to compute the scaling coefficient, it will be unbiased and can be a good approximation, as if $u_t^{(i)}$ was fully synchronized, assuming $x_t^{(i)}$ is synchronized. In this way, by using ϕ_{max}, we cancel out the bias in $\|u_t^{(i)}\|$ and obtain $\phi\left(\frac{\|x_t^{(\ell,i)} \odot M_t^{(\ell)}\|}{\|u_t^{(\ell,i)} \odot M_t^{(\ell)}\|}\right)$ as the scaling coefficient (see lines 15, 19 in Algo. 1). However, simply applying ϕ_{max} for the whole layer may cause divergence, because ϕ_{max} is too large for the non-synchronized part of $u_t^{(i)}$. Therefore, we need a more fine-grained way to adaptively rescale the learning rate for each element in $u_t^{(i)}$, according to their synchronization frequencies. We use the decaying factor β_3 and the staleness vector c_t to monitor the status of each element. Then, we use the weighted average between ϕ_{max} and ϕ_{min} to generate a new scaling function $\tilde{\phi}$, where ϕ_{min} is the lower bound. This is because we want to adjust the scaling coefficient according to the proportion of the synchronized parts in the momentum: when the momentum term is accumulated by a unsynchronized gradient, the proportion of the synchronized part is decreased by a factor of β_1. Hence we use a similar factor β_3 to capture this proportion. In this way, we manage to recover the scaling coefficient to the same magnitude as that of the original LAMB; see Figure 2 (b). Note that the learning rate also needs to be rescaled in the same way to avoid potential divergence (line 18 in Algo. 1.). We set the lower bound to $\eta/\sqrt{n}$ by the square root scaling rule, since each node has $1/n$ local mini-batch size.

Model synchronization. Since the model update $u_t^{(i)}$ is not fully synchronized, we end up with non-synchronized model parameters $x_t^{(i)}$ on each node. During non-synchronized steps, $x_t^{(i)}$ has larger variance than the fully synchronized x_t, which makes the scaling coefficient larger than LAMB's in some cases; see Figure 2 (b). Therefore, we perform the synchronization of model parameters every H steps, to bound the variance. Note that the last step T is always included for synchronization. In our experiment, we find that H is not a sensitive parameter; therefore, we set $H = 100$ as default.

4.2. Convergence analysis

In this section, we provide the convergence rate analysis for SLAMB in general non-convex settings. We follow (Gorbunov et al., 2021) to use virtual iterates x_t. It is defined as the mean of the local iterates $x_t = \frac{1}{n}\sum_{i\in[n]} x_t^{(i)}$, but only computed physically at synchronized steps, $i.e., t \in \{H, 2H, 3H, \ldots, T\}$. $g_t^{(i)}$ is the stochastic gradient computed batch-wise on each node with non-synchronized parameters $x_t^{(i)}$ satisfying $\mathbb{E}[g_t^{(i)}] = \nabla f^{(i)}(x_t^{(i)})$. We further define $\tilde{g}_t = \frac{1}{n}\sum_{i\in[n]} \tilde{g}_t^{(i)}$, where $\tilde{g}_t^{(i)}$ is computed on each node with synchronized parameters x_t in synchronized steps. For the non-synchronized steps, $\tilde{g}_t^{(i)}$ is computed virtually. It follows that $\mathbb{E}[\tilde{g}_t] = \nabla f(x_t)$. Our goal is to find an ϵ-solution of any (random) initial point x_1 such that for every synchronized parameters x_t, $\mathbb{E}\left[\|\nabla f(x_t)\|^2\right]$ is bounded.

Assumptions. We first introduce the assumptions following (You et al., 2020) and observations in the experiments. We assume coordinate-wise L-smoothness for every $f^{(i)}$ with respect to $[x_t^{(i)}]_j$, where $j \in [d]$. That is $|\nabla_j f^{(i)}(x) - \nabla_j f^{(i)}(y)| \le L_j^{(i)}\big|[x]_j - [y]_j\big|$We use L to denote $\max_{j\in[d],i\in[n]} L_j^{(i)}$.

For any stochastic gradient $g_t^{(i)}$, we assume its variance is coordinate-wise bounded. That is, there exists $\sigma_j^{(i)}$ for $j \in [d], i \in [n]$ such that: $\mathbb{E}\big\|[g_t^{(i)}]_j - [\nabla_j f^{(i)}(x_t)]\big\|^2 \le [\sigma_j^{(i)}]^2$. We use $\sigma^{(i)}$ to denote $\max_{j\in[d]} \sigma_j^{(i)}$.

We assume the coordinate-wisely bounded stochastic gradient $\|g_t^{(i)}\|_\infty \le G$, where $G \in \mathbb{R}$ and $G > 0$.

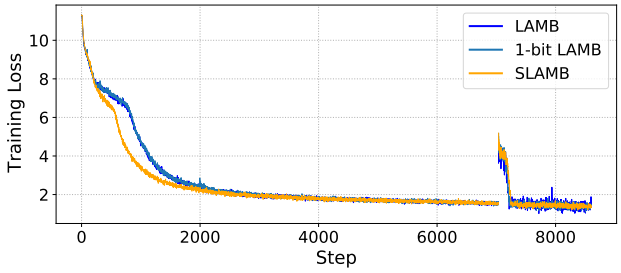
For every H non-synchronize steps with compression ratio k, we assume that the virtual gradient $\tilde{g}_t$ and averaged local gradients are element-wise bounded, that is $\mathbb{E}\big\|\frac{1}{n}\sum_{i\in[n]}[g_t^{(i)}]_j - [\tilde{g}_t]_j\|^2 \le \epsilon_{H,k}^2$.

We assume there exists at least one global minimum x^*, where $f(x) \le f(x^*)$ for all $x \in \mathbb{R}^d$.

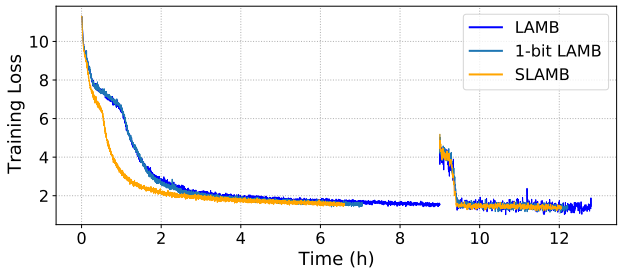
For all $i \in [n]$, we assume the local gradients are unbiased, that is $\frac{1}{n}\sum_{i\in[n]} \mathbb{E}[g_t^{(i)}] = \nabla f(x_t)$

Convergence of SLAMB. Similar as LAMB, we establish the convergence rate when $\beta_1 = 0, \lambda = 0, \beta_2 > 0, \beta_3 = 1$.

Theorem 4.1. *Let* $\eta_t = \eta = \sqrt{\frac{2(f(x_1)-f(x^*))}{\alpha_u^2 \|L\|_1 T}}$ *for all* $t \in [T]$, $b = T$, $d_i = d/h$ *for all* $i \in [h]$, *and* $\alpha_l \le \phi(v) \le \alpha_u$ *for all* $v > 0$ *where* $\alpha_l, \alpha_u > 0$. *Then for* x_t *generated using SLAMB (Algorithm 1), we have the following bounds:*



(a) Step-wise convergence



(b) Time-wise convergence

Figure 3. Training loss convergence for BERT-Large pre-training with LAMB and SLAMB. We use standard two-phase training strategy: phase1 with 128 seqlen for 7038 steps, and phase2 uses 512 seqlen for 1563 steps. The batch size for phase1/phase2 is 64K/32K.

$$\frac{1}{T}\sum_{t=1}^{T}\|\nabla f(x_t)\|^2 \leq O\Big(\sqrt{\frac{G^2 d}{h(1-\beta_2)}} \Big[\sqrt{\frac{8(f(x_1)-f(x^*))\|L\|_1}{T}} + \frac{1}{n}\sum_{i\in[n]}\frac{2\|\sigma^{(i)}\|_1}{\sqrt{T}} + d\frac{2\epsilon_{H,k}}{\sqrt{T}}\Big]\Big) \tag{4}$$

where k is the compression ratio, H is the synchronization frequency, and x^* is the optimal solution to the problem defined in Eq. 3, $x_t \in \{x_H, x_{2H}, x_{3H} \ldots, x_T\}$.

In general, both SLAMB and LAMB have convergence rates of the same order in terms of T. However, compressed communication has an impact on the convergence rate of SLAMB in two ways: Firstly, the convergence rate of SLAMB depends on the variance of the local gradients, using the distributed format $\frac{1}{n}\sum_{i\in[n]}\|\sigma^{(i)}\|_1$. This is in contrast to the convergence rate of LAMB, which is dependent on the synchronized gradient variance bound. Secondly, the convergence rate of SLAMB also depends on $\epsilon_{H,k}$, the bounded error between the virtual gradient and the average over all local gradients. Both terms are of the same order as LAMB's, as long as the compression ratio k and synchronization frequency H are properly set. If k is too small or H is too large, SLAMB converges slower than LAMB.

Table 2. BERT-Large pre-training final validation loss, and SQuAD average/max dev set F1 scores using the pre-trained models. We run this experiment on 128 V100 GPUs and repeat 3 times using different random seeds.

Algorithm	**LAMB**	**1-bit LAMB**	**SLAMB**
BERT validation loss	1.447	1.422	**1.419**
SQuAD Avg. F1	90.582	90.633	**90.646**
SQuAD Max F1	90.754	90.763	**90.790**
Volume Reduction	1.0 ×	4.6 ×	**9.1** ×
Time (h)	12.9	10.3	**9.7**

5. Experiments

We validate SLAMB on two DNN training tasks: language modeling and image classification, on up to 1,024 GPUs. We show that SLAMB converges at similar speed as LAMB but runs 1.3 to 2.3 times faster in end-to-end training than uncompressed LAMB.

5.1. BERT pre-training Task

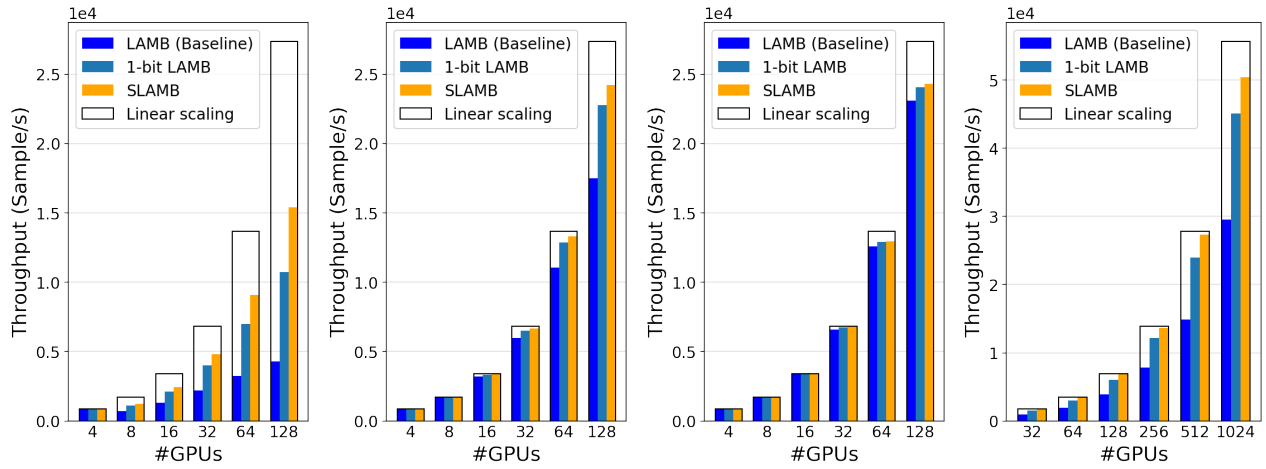
Model and Dataset. We run the BERT-Large pre-training task on the dataset introduced by (Devlin et al., 2018), which is a concatenation of Wikipedia and BooksCorpus with 2.5B and 800M words, respectively. We use NVIDIA's implementation[2] of BERT-Large (340M parameters) and the LAMB optimizer as the baseline. We further manually increase the size of BERT-Large by increasing the size of hidden layer and intermediate layer and obtain BERT-XLarge (2.8B parameters) for performance analysis. See detailed model configuration in Appendix E, Table 8.

Hardware. We use two different types of clusters in our experiment: (*i*) a V100 cluster on Amazon EC2 cloud, where each machine is equipped with 4 NVIDIA V100 GPUs (16GB memory each) and 10Gbps network (i.e., P3.16xlarge instance), with an optional upgrade to 8 GPUs per node and 100Gbps network; and (*ii*) an A100 cluster, where each machine has 8 NVIDIA A100 GPUs (80GB memory each) and 100Gbps Infiniband network. For both types of clusters, the GPUs within each node are connected with high speed NVLink.

Training setup. The pre-training of BERT consists of phase1 and phase2, with the input sequence length set to 128 and 512, respectively. We use the F1 score of SQuAD 1.1 [3] fine-tuning task as the accuracy metric to evaluate the pre-trained models. Note that the fine-tuning task uses the Adam optimizer and can be easily trained on a single machine.

[2]https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/LanguageModeling/BERT

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



(a) BERT-L, 1 Gbps (b) BERT-L, 10 Gbps (c) BERT-L, 100 Gbps (d) BERT-XL, 100 Gbps

Figure 4. Throughput scaling performance of BERT pre-training (seqlen128, phase1) using LAMB/1-bit LAMB/SLAMB at various bandwidth conditions (1/10/100 Gbps). In (a-c), we run BERT-Large on NVIDIA V100 GPU cluster, while in (d) we run BERT-XLarge on NVIDIA A100 GPU cluster. BERT-XLarge (2.8B params) is about 8× larger than BERT-Large (340M params).

Thus we only compare different methods in the pre-training stage. For SLAMB, we use the following hyper-parameter settings: $\beta_1 = 0.9, \beta_2 = 0.999, \beta_3 = 0.93$, compression ratio $k = 0.1$, model synchronization interval $H = 100$. Note that we keep β_1, β_2 identical to LAMB's setup. For 1-bit LAMB, we follow the setting from (Li et al., 2022): the warm-up ratio of phase1 and phase2 are set to 16.7% and 19.1%, respectively. The rest of the parameters, such as learning rate schedule and training steps, are identical across LAMB, 1-bit LAMB and SLAMB. We use mixed-precision training in this experiment, therefore the gradient is in float-16 format during communication. Detailed hyper-parameter setting are shown in Appendix G.1.

Convergence analysis. Figure 3a shows the step-wise convergence results of BERT-Large pre-training task. We use the largest batch size setting as (You et al., 2020), which is 64K in phase1 and 32K in phase2. We observe that in both phases, SLAMB provides comparable step-wise convergence speed as LAMB baseline. Note that in step 300-2000, SLAMB converges even faster in terms of training loss than other methods. This is mainly affected by β_3. We find that larger β_3 can speedup the convergence rate; however, it also makes the training unstable, especially when the learning rate is large.

After fine-tuning, we obtain the SQuAD F1 score as shown in Table 2. Our method, SLAMB, achieves the best validation loss and F1 score, compared to the original LAMB and 1-bit LAMB. SLAMB also reduces the communication volume by 9.1× compared to uncompressed LAMB, and by 2× compared to 1-bit LAMB. Unlike 1-bit LAMB, which requires a certain number of uncompressed warm-up steps before entering its compression stage, SLAMB uses a consistent compression ratio ($k = 0.1$) throughout the whole training process. Therefore, SLAMB achieves faster convergence speed in both phase1 and phase2, as illustrated in Figure 3b. We also show SLAMB with different compression ratios (k) for BERT-Large pre-training. As we can see from Table 3, although low compression ratio can greatly improve the scaling performance, e.g. 97% scaling efficiency when $k = 0.01$, the training quality (F1 score) is severely affected. Therefore, we fix the ratio at 0.1 to balance the performance and training quality.

Table 3. Training quality and scaling performance of SLAMB at different compression ratio for BERT-Large pre-training on 512 A100 GPUs. We use SQuAD Max F1 score to validate the training quality.

Algorithm	k	β_3	H	**Max F1 score**	**Scaling efficiency**
LAMB	-	-	-	90.754	52.0%
SLAMB	**0.5**	0.93	100	90.797	67.7%
	0.2	0.93	100	90.799	82.1%
	0.1	0.93	100	90.790	90.9%
	0.05	0.93	100	89.884	94.1%
	0.01	0.93	100	87.847	97.0%

Performance analysis. To demonstrate the practical applicability of SLAMB in realistic environment, we measure the throughput performance of BERT pre-training task by varying the model size, batch size, network bandwidth, as well as the number of GPUs. We configure a consistent compression ratio ($k = 0.1$) for SLAMB and 16.7% warm-up ratio for 1-bit LAMB. Throughput results are averaged for 400 training steps.

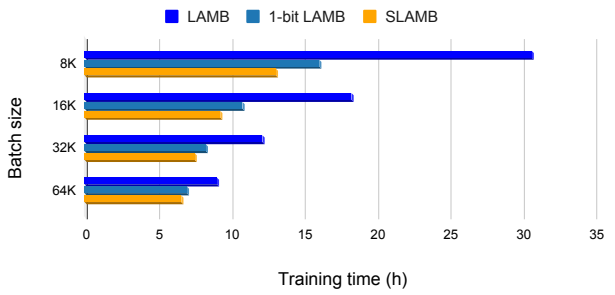
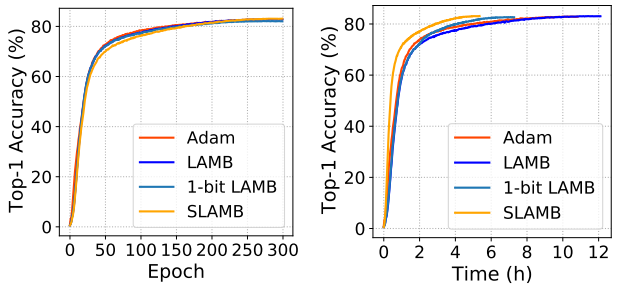


Figure 5. Training time of BERT-Large pre-training (seqlen128, phase1) on 128 V100 GPUs using LAMB/1-bit LAMB/SLAMB at different batch sizes. The network bandwidth is 10Gbps.



(a) (b)

Figure 6. Top-1 Test Accuracy of Swin Transformer training on ImageNet Dataset using 128 GPUs and 8K batch size. In (b) we compare the running time by setting 300 epochs for all methods.

Figure 4 (a-c) shows the strong scaling performance for BERT-Large pre-training in different bandwidth conditions (1/10/100 Gbps). We fix the total batch size to 64K, while increasing the number of GPUs from 4 to 128. Results show that training BERT-Large with LAMB suffers from low scalability when the bandwidth is limited (1/10 Gbps). In contrast, 1-bit LAMB and SLAMB both provide a significant speedup over LAMB. However, due to non-negligible uncompressed warm-up stage, 1-bit LAMB can not fully accelerate the training at all training steps. Once the network bandwidth is increased to 100Gbps, training is no longer constrained by communication for BERT-Large.

Figure 4 (d) shows the weak scaling performance for BERT-XLarge pre-training by fixing the batch size as 64 per GPU and increasing the number of GPUs up to 1,024. Due to the increased model size, communication now takes longer; thus, the baseline LAMB is suffering from low scalability even with high speed network. SLAMB can still achieve almost linear scaling on up to 512 GPUs and more than 90% scaling efficiency on 1,024 GPUs.

Furthermore, we study the acceleration of 1-bit LAMB and SLAMB for BERT-Large pre-training at different bath sizes. As illustrated in Figure 5, increasing the batch size can significantly reduce the training time. This is due to the fact that the total number of steps is halved when the batch size is doubled. As a result, the number of communication rounds is also reduced. Larger batch sizes causes longer computation time in each training step. However, the communication time in each step is not affected by batch size. Consequently, large batch training can improve the overall computation efficiency, as it is less affected by communication. Therefore, one should always use the largest possible batch size to maximize the computation ratio as well as minimize the communication rounds.

5.2. Swin Transformer on ImageNet

Setup. We run image classification task on ImageNet (Deng et al., 2009) dataset by using Swin Transformer Base model (Swin-B) (Liu et al., 2021) (88M parameters). Swin-B was originally trained with batch size 1024, which takes more than 70 hours on a single node with 8 V100 GPUs. To reduce the time, we scale the training to 128 GPUs and increase batch size to 8K in our experiment. For SLAMB, we use the default parameter setting, i.e., $\beta_3 = 0.95$, $k = 0.1$, $H = 100$. For 1-bit LAMB, we use 16.7% of total epochs as its warm-up stage (50 epochs). We use grid search to find the best learning rate (LR) for Adam and LAMB, respectively, and then keep the LR of SLAMB and 1-bit LAMB the same as LAMB's. We use a cosine decay LR scheduler and the first 20 epochs for linear LR warm-up. More detailed hyper-parameter tuning results can be found in Appendix G.2, Table 11.

Table 4. Swin-Base Transformer on ImageNet, using 128 V100 GPUs and 8K batch size. The network bandwidth is 10 Gbps. The original Swin-B is trained with 1024 batch size, which takes more than 70 hours on a single node with 8 V100 GPUs.

Algorithm	**Adam**	**LAMB**	**1-bit LAMB**	**SLAMB**
Top-1 Acc. (%)	83.08	83.09	82.91	**83.10**
Volume Reduction	1.0×	1.0×	4.6 ×	**9.1** ×
Time (h)	10.82	12.10	7.28	**5.35**

Results. Table 4 shows the final Top-1 test accuracy of trained models by four optimizers. Adam and LAMB achieve similar accuracy, while LAMB is 2 hours slower than Adam due to the computational overhead in its layer-wise updating strategy, especially when there are many training steps (46,800 steps in this task). In our experiment, we find that Adam is more likely to diverge when we increase the LR for large batch training. SLAMB also achieves the same accuracy; however, SLAMB transmits 9.1 × fewer data than LAMB during communication; therefore the training time is reduced from 12.1 hours (LAMB) to 5.35 hours (SLAMB). Our profiling results show that communication takes up 58% of the total training time in Swin Tranformer training (see Appendix H, Table 12).

Table 5. Per-step training time breakdown for LAMB and SLAMB. Experiment settings follow Table 2 and Table 4.

Task	Algorithm	Computation (s)	Communication (s)	Total (s)	Comm. / Total	Total time speedup
BERT-Large	LAMB	2.592	1.246	3.846	32%	-
	SLAMB	2.638	0.172	2.813	-	1.36 ×
Swin-Base	LAMB	0.418	0.647	1.101	58%	-
	SLAMB	0.422	0.075	0.499	-	2.20 ×

1-bit LAMB achieves a slightly lower accuracy than the rest, mainly because 1-bit LAMB cannot represent exact zero when compressing values into ± 1. It requires a mask to skip the parameters that have constant exact zeros in their gradients. Such mask is only configured for BERT training task and we did not find an easy way to adapt it for the Swin Transformer. Compared with 1-bit LAMB, SLAMB requires less effort to set up the parameters and provides better accuracy and speedup in large-scale Swin Transformer training. We illustrate the epoch-wise and time-wise convergence curve in Figure 6.

5.3. Time breakdown and speedup analysis

Here we provide detailed per-step time breakdown comparison between LAMB and SLAMB for BERT and Swin Transformer training tasks. The results are shown in Table 5. The ratio between communication time and total time denotes the communication overhead of the training task. Typically, tasks with higher communication overhead will benefit more from SLAMB acceleration. We can see from the results that Swin transformer task has larger communication overhead, thus the end-to-end speedup is also higher. Note that even for the same training task, the communication overhead can be also affected by network bandwidth, GPU numbers, batch size, and many other factors. We have profiled per-iteration time breakdown of different training workloads for BERT and Swin transformer tasks, see Appendix H, Table 12.

5.4. Ablation Study

In this part, we show the ablation study of SLAMB by removing the adaptive scaling function and model synchronization, respectively. In addition, we try to replace the adaptive scaling function with another min-max scaling strategy, i.e. using ϕ_{max} for the synchronized parts and ϕ_{min} for the unsynchronized parts. The results are shown is Table 6. We can see that both adaptive scaling and model synchronization have significant impact on the training quality, as the accuracy in both tasks have dropped a lot if we remove either of them. The min-max scaling strategy has little improvement compared to no adaptive scaling. This demonstrates that a more fine-grained scaling strategy is necessary for SLAMB to achieve lossless training quality.

Table 6. SQuAD performance of BERT-Large after pre-training using different variants of SLAMB

Algorithm	Avg. F1 Score
LAMB	90.582
SLAMB	90.622
SLAMB w/o adaptive scaling	89.177
SLAMB w/o model synchronization	90.176
SLAMB w/ min-max scaling	89.442

6. Conclusions

In this paper, we propose SLAMB, a communication efficient large batch optimizer. It is the first work that combines sparsification-based gradient compression with large batch training. SLAMB achieves similar convergence characteristics as LAMB, but with significantly lower communication overhead. We validate the performance of SLAMB on BERT language model and Swin Transformer vision model on up to 1,024 A100 GPUs. Our results show that SLAMB achieves virtually the same model accuracy as LAMB, while reducing the actual training time by up to 2.2 times. In addition, SLAMB achieves more than 90% scaling efficiency in large-scale distributed training with up to 1,024 GPUs. Currently SLAMB only supports the Random-k sparsifier. Other sparsification methods, like Top-k, may provide better performance but require algorithm adaptation, which we leave for future work.

Acknowledgements

We want to thank Yuzhe Wu, TingWen Xie, Jun Huang and Yuchen Xie for the GPU cluster support. We also want to thank the reviewers for their helpful suggestions.

References

Alistarh, D., Grubic, D., Li, J., Tomioka, R., and Vojnovic, M. Qsgd: Communication-efficient sgd via gradient quantization and encoding. *Advances in neural information processing systems*, 30, 2017.

Alistarh, D., Hoefler, T., Johansson, M., Konstantinov, N., Khirirat, S., and Renggli, C. The convergence of sparsified gradient methods. *Advances in Neural Information Processing Systems*, 31, 2018.

Bernstein, J., Wang, Y.-X., Azizzadenesheli, K., and Anandkumar, A. signsgd: Compressed optimisation for non-convex problems. In *International Conference on Machine Learning*, pp. 560–569. PMLR, 2018.

Bottou, L., Curtis, F. E., and Nocedal, J. Optimization methods for large-scale machine learning. *Siam Review*, 60(2), 2018.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Gorbunov, E., Hanzely, F., and Richtárik, P. Local sgd: Unified theory and new efficient methods. In *International Conference on Artificial Intelligence and Statistics*, pp. 3556–3564. PMLR, 2021.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Hoffer, E., Hubara, I., and Soudry, D. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. *Advances in neural information processing systems*, 30, 2017.

Jastrzebski, S., Kenton, Z., Arpit, D., Ballas, N., Fischer, A., Bengio, Y., and Storkey, A. Three factors influencing minima in sgd. *arXiv preprint arXiv:1711.04623*, 2017.

Karimireddy, S. P., Rebjock, Q., Stich, S., and Jaggi, M. Error feedback fixes signsgd and other gradient compression schemes. In *International Conference on Machine Learning*, pp. 3252–3261. PMLR, 2019.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations*, 2017.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations*, 2015.

Krizhevsky, A. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

Li, C., Awan, A. A., Tang, H., Rajbhandari, S., and He, Y. 1-bit lamb: communication efficient large-scale large-batch training with lamb's convergence speed. In *2022 IEEE 29th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, pp. 272–281. IEEE, 2022.

Li, M. *Scaling distributed machine learning with system and algorithm co-design*. PhD thesis, Intel, 2017.

Li, M., Zhang, T., Chen, Y., and Smola, A. J. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 661–670, 2014.

Lin, Y., Han, S., Mao, H., Wang, Y., and Dally, W. J. Deep Gradient Compression: Reducing the communication bandwidth for distributed training. In *The International Conference on Learning Representations*, 2018.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10012–10022, 2021.

Seide, F., Fu, H., Droppo, J., Li, G., and Yu, D. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns. In *Fifteenth annual conference of the international speech communication association*, 2014.

Shallue, C. J., Lee, J., Antognini, J., Sohl-Dickstein, J., Frostig, R., and Dahl, G. E. Measuring the effects of data parallelism on neural network training. *Journal of Machine Learning Research*, 20(112):1–49, 2019.

Stich, S. U., Cordonnier, J.-B., and Jaggi, M. Sparsified sgd with memory. *Advances in Neural Information Processing Systems*, 31, 2018.

Strom, N. Scalable distributed DNN training using commodity GPU cloud computing. In *16th Annual Conference of the International Speech Communication Association*, 2015.

Tang, H., Gan, S., Awan, A. A., Rajbhandari, S., Li, C., Lian, X., Liu, J., Zhang, C., and He, Y. 1-bit Adam: Communication Efficient Large-Scale Training with Adam's Convergence Speed. In *International Conference on Machine Learning*, 2021.

Wangni, J., Wang, J., Liu, J., and Zhang, T. Gradient sparsification for communication-efficient distributed optimization. *Advances in Neural Information Processing Systems*, 31, 2018.

Wen, W., Xu, C., Yan, F., Wu, C., Wang, Y., Chen, Y., and Li, H. Terngrad: Ternary gradients to reduce communication in distributed deep learning. *Advances in neural information processing systems*, 30, 2017.

Xu, H., Ho, C.-Y., Abdelmoniem, A. M., Dutta, A., Bergou, E. H., Karatsenidis, K., Canini, M., and Kalnis, P. Grace: A compressed communication framework for distributed machine learning. In *2021 IEEE 41st international conference on distributed computing systems (ICDCS)*, pp. 561–572. IEEE, 2021a.

Xu, H., Kostopoulou, K., Dutta, A., Li, X., Ntoulas, A., and Kalnis, P. Deepreduce: A sparse-tensor communication framework for federated deep learning. *Advances in Neural Information Processing Systems*, 34:21150–21163, 2021b.

You, Y., Gitman, I., and Ginsburg, B. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017.

You, Y., Zhang, Z., Hsieh, C.-J., Demmel, J., and Keutzer, K. Imagenet training in minutes. In *Proceedings of the 47th International Conference on Parallel Processing*, pp. 1–10, 2018.

You, Y., Li, J., Reddi, S. J., Hseu, J., Kumar, S., Bhojanapalli, S., Song, X., Demmel, J., Keutzer, K., and Hsieh, C. Large batch optimization for deep learning: Training BERT in 76 minutes. In *8th International Conference on Learning Representations*, 2020.

Zheng, S., Lin, H., Zha, S., and Li, M. Accelerated large batch optimization of bert pretraining in 54 minutes. *arXiv preprint arXiv:2006.13484*, 2020.

Zinkevich, M., Weimer, M., Li, L., and Smola, A. Parallelized stochastic gradient descent. *Advances in neural information processing systems*, 23, 2010.

A. Problem formulation and assumption

A.1. Formulation

In this paper, we study non-convex stochastic optimization problems of the form

$$\min_{x\in\mathbb{R}^d}\left\{f(x):=\frac{1}{n}\sum_{i\in[n]}f^{(i)}(x)\right\}\ , \tag{5}$$

where $f^{(i)}:\mathbb{R}^d\to\mathbb{R}$ is a smooth non-convex function for all $i\in[n]:=\{1,2,\ldots,n\}$ representing the loss function on each node. d is the dimension of the input model x. Our goal is to find a ϵ-solution of any (random) initial point x_1 such that $\mathbb{E}\left[\|\nabla f(x_t)\|^2\right]$ is bounded.

In order to simplify our notations, we follow (Gorbunov et al., 2021) to use virtual iterates x_t defined as a mean of the local iterates $x_t=\frac{1}{n}\sum_{i\in[n]}x_t^{(i)}$. Note that only when the local parameters $x_t^{(i)}$ are synchronized, virtual iterates are physically computed.

$g_t^{(i)}$ is the stochastic gradient computed batch-wise on each node with non-synchronized parameters $x_t^{(i)}$ satisfying $\mathbb{E}[g_t^{(i)}]=\nabla f^{(i)}(x_t^{(i)})$.

We further define $\tilde{g}_t=\frac{1}{n}\sum_{i\in[n]}\tilde{g}_t^{(i)}$, where $\tilde{g}_t^{(i)}$ is computed on each node with synchronized parameters. For the non-synchronized steps, $\tilde{g}_t^{(i)}$ is virtually computed. It follows that $\mathbb{E}[\tilde{g}_t]=\nabla f(x_t)$.

Since LAMB is a layer-wise optimization algorithm, we write $x^{(i)}=\sum_\ell x^{(i,\ell)}$, which is the summation of the layer-wise parameters $x^{(\ell)}\in\mathbb{R}^{d_\ell}$, where $d=\sum_{\ell\in[h]}d_\ell$

A.2. Notation

The notations in the appendix are used as follows,

- Upper-scripted index denotes layer-wise vector on a specific node. For example, $x^{(i,\ell)}$ denotes ℓ-th layer model parameters on node i.
- Sub-scripted index denotes the update step t.
- $[x]_j$ is j-th element of a vector x .
- All the operations are element-wised.

A.3. Assumption

Assumptions adopted from (You et al., 2020)

1. Smoothness.We assume coordinate-wise L-smoothness for every $f^{(i)}$ with respect to $[x_t^{(i)}]_j$, where $j\in[d]$. That is

$$|\nabla_j f^{(i)}(x)-\nabla_j f^{(i)}(y)|\le L_j^{(i)}\left|[x]_j-[y]_j\right|$$

   We use L to denote $\max_{j\in[d],i\in[n]}L_j^{(i)}$.

2. Bounded variance. For any stochastic gradient $g_t^{(i)}$, we assume it coordinate-wise bounded. That is, there exists $\sigma_j^{(i)}$ for $j\in[d],i\in[n]$

$$\mathbb{E}\left\|[g_t^{(i)}]_j-[\nabla_j f^{(i)}(x_t)]\right\|^2\le[\sigma_j^{(i)}]^2\ .$$

   We use $\sigma^{(i)}$ to denote $\max_{j\in[d]}\sigma_j^{(i)}$.

3. Bounded gradients. We assume coordinate-wisely bounded stochastic gradient $\|g_t^{(i)}\|_\infty\le G$

4. For every H non-synchronize steps with k synchronize bits, we assume that the virtual gradient $\tilde{g}_t$ and averaged local gradients are element-wise bounded, *i.e.*,

$$\mathbb{E}\|\frac{1}{n}\sum_{i\in[n]}[g_t^{(i)}]_j - [\tilde{g}_t]_j\|^2 \leq \epsilon_{H,k}^2 \ .$$

5. Bounded rescaling factor. We assume that $0 < \alpha_l \leq \phi(v) \leq \alpha_u$ for all $v \in \mathbb{R}_+ \cup \{0\}$. It follows that $< \alpha_l \leq \tilde{\phi}(v) \leq \alpha_u$ for all $v \in \mathbb{R}_+ \cup \{0\}$.

6. Existence of global minimum. We assume there exists at least one global minimum x^*, where $f(x) \leq f(x^*)$ for all $x \in \mathbb{R}^d$

7. Unbiased local gradient. For all $i \in [n]$ we have

$$\frac{1}{n}\sum_{i\in[n]}\mathbb{E}[g_t^{(i)}] = \nabla f(x_t)$$

Prove guideline 1. Develop new bounds for $p_{t,i}$ for assumption 2,3. 2. Assemble all the $f^{(i)}$ and use $f(x^*) = \sum_{i\in[n]} f^{(i)}(x^*)$. Note that x^* is not necessarily to be the optimal for $f^{(i)}(x)$

B. Proof

Similar as (You et al., 2020), we focus on the setting where $\beta_1 = 0$, $\lambda = 0$, and $\tilde{\eta}_t = \eta$ for all t and all layers.

We rewrite our update scheme here

$$p_t^{(i)} = \frac{1}{n}\Big(\sum_{m\in[n]}\mathcal{C}(g_t^{(m)})\Big) + g_t^{(i)} - \mathcal{C}(g_t^{(i)}) \tag{6}$$

$$v_t^{(i)} = \frac{\beta_2 v_{t-1}^{(i)} + (1-\beta_2){p_t^{(i)}}^2}{1-\beta_2^t} \tag{7}$$

$$u_t^{(i)} = \frac{p_t^{(i)}}{\sqrt{v_t^{(i)}}} \tag{8}$$

$$x_{t+1}^{(\ell)} = x_t^{(\ell)} - \eta\Big(\frac{1}{n}\sum_{m\in[n]}\tilde{\phi}\cdot\frac{u_t^{(m,\ell)}}{\|u_t^{(m,\ell)}\|}\Big) \tag{9}$$

$x_{t+1}^{(\ell)}, x_t^{(\ell)}$ in (9) are the virtual iterates. In non-synchronize steps, it is physically computed as $x_{t+1}^{(i,\ell)} = x_t^{(i,\ell)} - \eta\tilde{\phi}\frac{u_t^{(i,\ell)}}{\|u_t^{(i,\ell)}\|}$ for any $i \in [n]$.

Proposition B.1 (Facts related to the algorithm)**.** *1. f is a linear summation of $f^{(i)}$, then f is L-smoothness.*

Proof.

$$\begin{aligned}\big|[\nabla_\ell f(x) - \nabla_\ell f(y)]_j\big| &= \big|\frac{1}{n}\sum_i[\nabla_\ell f^{(i)}(x) - \nabla_\ell f^{(i)}(y)]_j\big| \\ &\leq \frac{1}{n}\sum_i\big|[\nabla_\ell f^{(i)}(x) - \nabla_\ell f^{(i)}(y)]_j\big| \\ &\leq L\big|[x]_j - [y]_j\big|\end{aligned}$$

The first inequality is by the inequality of the absolute value, and the second inequality is by the smoothness of each $f^{(i)}$. □

2. *Triangle inequality for norms. For any feasible norm $\|\cdot\|$ and any $a_i, \ldots, a_n \in \mathbb{R}^d$*

$$\|\sum_{i\in[n]} a_i\| \le \sum_{i\in[n]} \|a_i\| \tag{10}$$

3. *Inequality for L-smoothness function. If f is L-smoothness, for any $x, y \in \mathbb{R}^d$ we have*

$$f(y) \le f(x) + \langle \nabla f(x), y - x\rangle + \frac{L}{2}\|x - y\|^2 \tag{11}$$

Proposition B.2. *Several properties for the variables x_t*

1. $\|x_{t+1}^{(\ell)} - x_t^{(\ell)}\| \le \alpha_u \eta$
2. $\|u_t^{(i,\ell)}\| \le \sqrt{\frac{d_\ell}{1-\beta_2}}$
3. $\frac{1}{n}\sum_{i\in[n]} p_t^{(i)} = \frac{1}{n}\sum_{i\in[n]} g_t^{(i)}$
4. $[p_t^{(i)}]_j \le G$
5. $\sqrt{v_t^{(i)}} \le G$

Proof. 1. We bound the term as following,

$$\begin{aligned}
\|x_{t+1}^{(\ell)} - x_t^{(\ell)}\| &= \Big\|\eta\Big(\frac{1}{n}\sum_{m\in[n]} \tilde{\phi}\cdot\frac{u_t^{(m,\ell)}}{\|u_t^{(m,\ell)}\|}\Big)\Big\| \\
&\le \alpha_u\eta\Big\|\frac{1}{n}\sum_{m\in[n]}\frac{u_t^{(m,\ell)}}{\|u_t^{(m,\ell)}\|}\Big\| \\
&\le \alpha_u\eta \ ,
\end{aligned}$$

where the first inequality holds by Assumption 4, and the second inequality holds by the triangle inequality (10).

2. We bound the term as the following

$$\begin{aligned}
\|u_t^{(i,\ell)}\| &= \Big\|\frac{\sqrt{1-\beta_2^t}\, p_t^{(i,\ell)}}{\sqrt{\beta_2 v_{t-1}^{(i,\ell)} + (1-\beta_2){p_t^{(i,\ell)}}^2}}\Big\| \\
&\le \Big\|\frac{p_t^{(i,\ell)}}{\sqrt{(1-\beta_2){p_t^{(i,\ell)}}^2}}\Big\| \\
&\le \sqrt{\frac{d_\ell}{1-\beta_2}}
\end{aligned}$$

3.

$$\frac{1}{n}\sum_{i\in[n]} p_t^{(i)} = \frac{1}{n}\sum_{i\in[n]}\Big(\frac{1}{n}\Big(\sum_{m\in[n]}\mathcal{C}(g_t^{(m)})\Big) + g_t^{(i)} - \mathcal{C}(g_t^{(i)})\Big) = \frac{1}{n}\sum_{i\in[n]} g_t^{(i)}$$

4. For the non-synchronized elements in $p_t^{(i)}$, $[p_t^{(i)}]_j = [g_t^{(i)}]_j \le G$. For the synchronized elements in $p_t^{(i)}$,

$$[p_t^{(i)}]_j = \frac{1}{n}\sum_{i\in[n]} [g_t^{(i)}]_j \le G$$

5. This can be easily seen by the induction method with $v_0^{(i)} \leq G$

□

Lemma B.3. *The probability that the sign of* $[\nabla f(x_t)]_j$ *and* $[\frac{1}{n}\sum_{m\in[n]} p_t^{(m)}]_j$ *is different is bounded, that is*

$$\mathbb{P}\big(\operatorname{sign}([\nabla_\ell f(x_t)]_j) \neq \operatorname{sign}([\frac{1}{n}\sum_{m\in[n]} p_t^{(m,\ell)}]_j\big) \leq \frac{\frac{\epsilon_{H,k}}{\sqrt{b}} + \frac{1}{n}\sum_{m\in[n]}\frac{[\sigma^{(m)}]_j}{\sqrt{b}}}{\big|[\nabla_\ell f(x_t)]_j\big|}$$

Proof. We develop this lemma mainly following the Equation † in Theorem 2 in (Bernstein et al., 2018). By the Markov's inequality and Jensen's inequality, we have

$$\begin{aligned}
&\mathbb{P}\bigg(\operatorname{sign}([\nabla_\ell f(x_t)]_j) \neq \operatorname{sign}([\frac{1}{n}\sum_{m\in[n]} p_t^{(m,\ell)}]_j\bigg)\\
&\leq \mathbb{P}\bigg(\big|[\nabla_\ell f(x_t)]_j - [\frac{1}{n}\sum_{m\in[n]} p_t^{(m,\ell)}]_j\big| \geq \big|[\nabla_\ell f(x_t)]_j\big|\bigg)\\
&\leq \frac{\mathbb{E}\big|[\nabla_\ell f(x_t)]_j - [\frac{1}{n}\sum_{m\in[n]} p_t^{(m,\ell)}]_j\big|}{\big|[\nabla_\ell f(x_t)]_j\big|}\\
&\leq \frac{\mathbb{E}\big|[\nabla_\ell f(x_t)]_j - [\frac{1}{n}\sum_{m\in[n]} g_t^{(m,\ell)}]_j\big|}{\big|[\nabla_\ell f(x_t)]_j\big|}\\
&\leq \frac{\mathbb{E}\big|[\nabla_\ell f(x_t)]_j - \frac{1}{n}\sum_{m\in[n]} \nabla_\ell f(x_t^{(i)})\big| + \frac{1}{n}\sum_{m\in[n]}\mathbb{E}\big|[\nabla_\ell f_m(x_t^{(i)})]_j - [g_t^{(m,\ell)}]_j\big|}{\big|[\nabla_\ell f(x_t)]_j\big|}\\
&\leq \frac{\mathbb{E}\big|[\tilde{g}_t]_j - \frac{1}{n}\sum_{m\in[n]} \nabla_\ell g_t^{(i)}\big| + \frac{1}{n}\sum_{m\in[n]}\mathbb{E}\big|[\nabla_\ell f_m(x_t^{(i)})]_j - [g_t^{(m,\ell)}]_j\big|}{\big|[\nabla_\ell f(x_t)]_j\big|}\\
&\leq \frac{\sqrt{\mathbb{E}\big|[\tilde{g}_t]_j - \frac{1}{n}\sum_{m\in[n]} \nabla_\ell g_t^{(i)}\big|^2} + \frac{1}{n}\sum_{m\in[n]}\sqrt{\mathbb{E}\big|[\nabla_\ell f_m(x_t)]_j - [g_t^{(m,\ell)}]_j\big|^2}}{\big|[\nabla_\ell f(x_t)]_j\big|}\\
&\leq \frac{\epsilon_{H,k} + \frac{1}{n}\sum_{m\in[n]}[\sigma^{(m)}]_j}{\big|[\nabla_\ell f(x_t)]_j\big|}
\end{aligned}$$

In practice, the upper bound of the σ, ϵ is computed by a mini-batch with size b. Then by assumption 2, we have

$$\mathbb{P}\bigg(\operatorname{sign}([\nabla_\ell f(x_t)]_j) \neq \operatorname{sign}([\frac{1}{n}\sum_{m\in[n]} p_t^{(m,\ell)}]_j\bigg) \leq \frac{\frac{\epsilon_{H,k}}{\sqrt{b}} + \frac{1}{n}\sum_{m\in[n]}\frac{[\sigma^{(m)}]_j}{\sqrt{b}}}{\big|[\nabla_\ell f(x_t)]_j\big|} \tag{12}$$

□

Theorem B.4. *After* T *iterations for our algorithm with the general gradient estimator* $p_t^{(i)}$ *with* $[p_t^{(i)}]_j \leq V$*, we have the following error bound*

$$\mathbb{E}\|\nabla f(x_t)\|^2 \leq O\Big(\sqrt{\frac{G^2 d}{h(1-\beta_2)}}\Big[\sqrt{\frac{2(f(x_1)-f(x^*))\|L\|_1}{T}} + \frac{1}{n}\sum_{i\in[n]}\frac{\|\sigma^{(i)}\|_1}{2\sqrt{T}} + d\frac{\epsilon_{H,k}}{2\sqrt{T}}\Big]\Big) \ . \tag{13}$$

Proof. Using the L-smoothness inequality, we first have the following

$$
\begin{aligned}
f(x_{t+1}) &\leq f(x_t) + \sum_{\ell\in[h]} \langle \nabla_\ell f(x_t), x_{t+1}^{(\ell)} - x_t^{(\ell)} \rangle + \sum_{\ell\in[h]} \frac{L}{2} \|x_{t+1}^{(\ell)} - x_t^{(\ell)}\|^2 \\
&\leq f(x_t) - \eta \sum_{\ell\in[h]} \sum_{j\in[d_\ell]} \left(\big[\nabla_\ell f(x_t)\big]_j \cdot \Big(\frac{1}{n} \sum_{m\in[n]} \tilde{\phi} \frac{\big[u_t^{(m,\ell)}\big]_j}{\|u_t^{(m,\ell)}\|} \Big) \right) + \sum_{\ell\in[h]} \frac{L\alpha_u^2\eta^2}{2} \\
&= f(x_t) \underbrace{ -\eta \sum_{\ell\in[h]} \sum_{j\in[d_\ell]} \Bigg(\underbrace{ \frac{1}{n} \sum_{m\in[n]} \tilde{\phi} \cdot \big[\nabla_\ell f(x_t)\big]_j \cdot \frac{\big[u_t^{(m,\ell)}\big]_j}{\|u_t^{(m,\ell)}\|} }_{T_2} \Bigg) }_{T_1} + \sum_{\ell\in[h]} \frac{L\alpha_u^2\eta^2}{2} \quad ,
\end{aligned}
\tag{14}
$$

where the first inequality holds by the L-smoothness of f in Fact 1 and Equation (11), and the second inequality holds by the Proposition B.2.1.

T_2 can be bounded by

$$
\begin{aligned}
T_2 =& \frac{1}{n} \sum_{m\in[n]} \tilde{\phi} \cdot \big[\nabla_\ell f(x_t)\big]_j \cdot \frac{\big[u_t^{(m,\ell)}\big]_j}{\|u_t^{(m,\ell)}\|} \\
=& \big[\nabla_\ell f(x_t)\big]_j \cdot \frac{1}{n} \sum_{m\in[n]} \Big[\frac{\tilde{\phi} p_t^{(m,\ell)}}{\sqrt{v_t^{(m,\ell)}}\|u_t^{(m,\ell)}\|} \Big]_j \cdot \mathbf{1}_{\{\mathrm{sign}([\nabla_\ell f(x_t)]_j) = \mathrm{sign}([\frac{1}{n}\sum_{m\in[n]} p_t^{(m,\ell)}]_j)\}} \\
&+ \big[\nabla_\ell f(x_t)\big]_j \cdot \frac{1}{n} \sum_{m\in[n]} \Big[\frac{\tilde{\phi} p_t^{(m,\ell)}}{\sqrt{v_t^{(m,\ell)}}\|u_t^{(m,\ell)}\|} \Big]_j \cdot \mathbf{1}_{\{\mathrm{sign}([\nabla_\ell f(x_t)]_j) \neq \mathrm{sign}([\frac{1}{n}\sum_{m\in[n]} p_t^{(m,\ell)}]_j)\}} \\
\geq& \Big| \big[\nabla_\ell f(x_t)\big]_j \cdot \frac{1}{n} \sum_{m\in[n]} \big[\tilde{\phi} p_t^{(m,\ell)}\big]_j \cdot \sqrt{\frac{1-\beta_2}{G^2 d_\ell}} \Big| \\
&+ 2\big[\nabla_\ell f(x_t)\big]_j \cdot \frac{1}{n} \sum_{m\in[n]} \Big[\frac{\tilde{\phi} p_t^{(m,\ell)}}{\sqrt{v_t^{(m,\ell)}}\|u_t^{(m,\ell)}\|} \Big]_j \cdot \mathbf{1}_{\{\mathrm{sign}([\nabla_\ell f(x_t)]_j) \neq \mathrm{sign}([\frac{1}{n}\sum_{m\in[n]} p_t^{(m,\ell)}]_j)\}}
\end{aligned}
\tag{15}
$$

The inequality holds by Proposition B.2.2 $\|u_t^{(i,\ell)}\| \leq \sqrt{\frac{d_\ell}{1-\beta_2}}$ and $\big[\sqrt{v_t^{(i)}}\big]_j \leq G$

Taking the expectation of T_1 over $p_t^{(i)}$, we have

$$
\begin{aligned}
\mathbb{E}\big[T_1\big] \leq& -\eta \sum_{\ell\in[h]} \sum_{j\in[d_\ell]} \sqrt{\frac{1-\beta_2}{G^2 d_\ell}} \mathbb{E}\Big[\big[\nabla_\ell f(x_t)\big]_j \cdot \frac{1}{n} \sum_{m\in[n]} \tilde{\phi} \cdot \big[p_t^{(m,\ell)}\big]_j \Big] \\
&- \eta \sum_{\ell\in[h]} \sum_{j\in[d_\ell]} \mathbb{E}\Big[\big[\nabla_\ell f(x_t)\big]_j \cdot \frac{1}{n} \sum_{m\in[n]} \Big[\frac{\tilde{\phi} p_t^{(m,\ell)}}{\sqrt{v_t^{(m,\ell)}}\|u_t^{(m,\ell)}\|} \Big]_j \cdot \mathbf{1}_{\{\mathrm{sign}([\nabla_\ell f(x_t)]_j) \neq \mathrm{sign}([\frac{1}{n}\sum_{m\in[n]} p_t^{(m,\ell)}]_j)\}} \Big]
\end{aligned}
\tag{16}
$$

Taking the absolute value of the internal part of the second expectation of Equation (16), we have

$$
\begin{aligned}
(16) \leq & -\eta \sum_{\ell\in[h]}\sum_{j\in[d_\ell]} \sqrt{\frac{1-\beta_2}{G^2 d_\ell}}\mathbb{E}\Big[\big[\nabla_\ell f(x_t)\big]_j \cdot \frac{1}{n}\sum_{m\in[n]} \tilde{\phi}\cdot \big[p_t^{(m,\ell)}\big]_j\Big] \\
& + \eta \sum_{\ell\in[h]}\sum_{j\in[d_\ell]} \mathbb{E}\Big[\big|\big[\nabla_\ell f(x_t)\big]_j \cdot \frac{1}{n}\sum_{m\in[n]} [\frac{\tilde{\phi} p_t^{(m,\ell)}}{\sqrt{v_t^{(m,\ell)}}\|u_t^{(m,\ell)}\|}]_j\big| \cdot \mathbf{1}_{\{\mathrm{sign}([\nabla_\ell f(x_t)]_j)\neq \mathrm{sign}([\frac{1}{n}\sum_{m\in[n]} p_t^{(m,\ell)}]_j)\}}\Big] \\
\leq & -\eta \sum_{\ell\in[h]}\sum_{j\in[d_\ell]} \sqrt{\frac{1-\beta_2}{G^2 d_\ell}}\mathbb{E}\Big[\big[\nabla_\ell f(x_t)\big]_j \cdot \frac{1}{n}\sum_{m\in[n]} \tilde{\phi}\cdot \big[p_t^{(m,\ell)}\big]_j\Big] \\
& + \eta \sum_{\ell\in[h]}\sum_{j\in[d_\ell]} \mathbb{E}\Big[\frac{1}{n}\sum_{m\in[n]} \alpha_u \big|\big[\nabla_\ell f(x_t)\big]_j\big| \cdot \mathbf{1}_{\{\mathrm{sign}([\nabla_\ell f(x_t)]_j)\neq \mathrm{sign}([\frac{1}{n}\sum_{m\in[n]} p_t^{(m,\ell)}]_j)\}}\Big] \\
= & -\eta \sum_{\ell\in[h]}\sum_{j\in[d_\ell]} \sqrt{\frac{1-\beta_2}{G^2 d_\ell}}\mathbb{E}\Big[\big[\nabla_\ell f(x_t)\big]_j \cdot \frac{1}{n}\sum_{m\in[n]} \tilde{\phi}\cdot \big[p_t^{(m,\ell)}\big]_j\Big] \\
& + \eta \sum_{\ell\in[h]}\sum_{j\in[d_\ell]} \Big[\frac{1}{n}\sum_{m\in[n]} \alpha_u \big|\big[\nabla_\ell f(x_t)\big]_j\big| \cdot \mathbb{P}\big(\mathrm{sign}([\nabla_\ell f(x_t)]_j) \neq \mathrm{sign}([\frac{1}{n}\sum_{m\in[n]} p_t^{(m,\ell)}]_j)\big)\Big]
\end{aligned}
\tag{17}
$$

where the second inequality is by $\left|\frac{\left[u_t^{(m,\ell)}\right]_j}{\|u_t^{(m,\ell)}\|}\right| \leq 1$ and $\tilde{\phi} \leq \alpha_u$

For the first part of the equation (17), we have

$$
\begin{aligned}
& \mathbb{E}\Big[\big[\nabla_\ell f(x_t)\big]_j \cdot \frac{1}{n}\sum_{m\in[n]} \tilde{\phi}\cdot \big[p_t^{(m,\ell)}\big]_j\Big] \\
& \geq \alpha_l \mathbb{E}\Big[\big[\nabla_\ell f(x_t)\big]_j \cdot \frac{1}{n}\sum_{m\in[n]} \big[p_t^{(m,\ell)}\big]_j\Big] \\
& = \alpha_l \big[\nabla_\ell f(x_t)\big]_j \mathbb{E}\Big[\frac{1}{n}\sum_{m\in[n]} \big[p_t^{(m,\ell)}\big]_j\Big] \\
& = \alpha_l \big[\nabla_\ell f(x_t)\big]_j \mathbb{E}\Big[\frac{1}{n}\sum_{m\in[n]} \big[g_t^{(m,\ell)}\big]_j\Big] \\
& = \alpha_l \big[\nabla_\ell f(x_t)\big]_j \Big(\mathbb{E}[\tilde{g}_t]_j - \mathbb{E}\big([\tilde{g}_t]_j - \frac{1}{n}\sum_{m\in[n]} \big[g_t^{(m,\ell)}\big]_j\big)\Big) \\
& = \alpha_l \Big(\big[\nabla_\ell f(x_t)\big]_j^2 - \mathbb{E}[\tilde{g}_t]_j \mathbb{E}\big([\tilde{g}_t]_j - \frac{1}{n}\sum_{m\in[n]} \big[g_t^{(m,\ell)}\big]_j\big)\Big) \\
& \geq \alpha_l \Big(\big[\nabla_\ell f(x_t)\big]_j^2 - \frac{(\mathbb{E}[\tilde{g}_t]_j)^2 + \big(\mathbb{E}\big[[\tilde{g}_t]_j - \frac{1}{n}\sum_{m\in[n]} \big[g_t^{(m,\ell)}\big]_j\big]\big)^2)}{2}\Big) \\
& \geq \alpha_l \Big(\big[\nabla_\ell f(x_t)\big]_j^2 - \frac{(\mathbb{E}[\tilde{g}_t]_j)^2 + \mathbb{E}\big[[\tilde{g}_t]_j - \frac{1}{n}\sum_{m\in[n]} \big[g_t^{(m,\ell)}\big]_j\big]^2)}{2}\Big) \\
& \geq \alpha_l (\frac{1}{2}\big[\nabla_\ell f(x_t)\big]_j^2 - \frac{\epsilon_{H,k}^2}{2b}) \ ,
\end{aligned}
\tag{18}
$$

where the equality in the third line is from Proposition B.2.3.

For the second part of the equation (17), we have

$$\begin{aligned}
&\frac{1}{n}\sum_{m\in[n]}\alpha_u\big|\big[\nabla_\ell f(x_t)\big]_j\big|\cdot\mathbb{P}\big(\operatorname{sign}([\nabla_\ell f(x_t)]_j)\neq\operatorname{sign}([\frac{1}{n}\sum_{m\in[n]}p_t^{(m,\ell)}]_j)\big)\\
&\leq\frac{1}{n}\sum_{m\in[n]}\alpha_u\big(\frac{1}{n}\sum_{i\in[n]}\frac{[\sigma^{(i)}]_j}{\sqrt{b}}+\frac{\epsilon_{H,k}}{\sqrt{b}}\big)\\
&=\alpha_u\big(\frac{1}{n}\sum_{i\in[n]}\frac{[\sigma^{(i)}]_j}{\sqrt{b}}+\frac{\epsilon_{H,k}}{\sqrt{b}}\big)
\end{aligned}\tag{19}$$

Combining Equation (17), (18), (19), we have

$$\begin{aligned}
\mathbb{E}[T_1]&\leq-\eta\sqrt{\frac{1-\beta_2}{G^2d_\ell}}\sum_{\ell\in[h]}\sum_{j\in[d_\ell]}\alpha_l\big(\frac{1}{2}\big[\nabla_\ell f(x_t)\big]_j^2+\frac{\epsilon_{H,k}^2}{2b}\big)+\eta\sum_{\ell\in[h]}\sum_{j\in[d_\ell]}\alpha_u\big(\frac{1}{n}\sum_{i\in[n]}\frac{[\sigma^{(i)}]_j}{\sqrt{b}}+\frac{\epsilon_{H,k}}{\sqrt{b}}\big)\\
&=-\eta\alpha_l\sqrt{\frac{h(1-\beta_2)}{G^2d_\ell}}\big(\frac{1}{2}\|\nabla f(x_t)\|^2+\frac{d\epsilon_{H,k}^2}{2b}\big)+\eta d\alpha_u\frac{\epsilon_{H,k}}{\sqrt{b}}+\frac{\eta\alpha_u}{n}\sum_{i\in[n]}\frac{\|\sigma^{(i)}\|_1}{\sqrt{b}}
\end{aligned}\tag{20}$$

Here b is the batch size. Substituting the above bound on T_1 in Equation (14), we have the following bound

$$\begin{aligned}
&\mathbb{E}[f(x_{t+1})]\leq\\
&\qquad\mathbb{E}[f(x_t)]-\eta\alpha_l\sqrt{\frac{h(1-\beta_2)}{G^2d}}\big(\frac{1}{2}\|\nabla f(x_t)\|^2+\frac{d\epsilon_{H,k}^2}{2b}\big)+\frac{\eta\alpha_u}{n}\sum_{i\in[n]}\frac{\|\sigma^{(i)}\|_1}{\sqrt{b}}+\eta d\alpha_u\frac{\epsilon_{H,k}}{\sqrt{b}}+\sum_{\ell\in[h]}\frac{L\alpha_u^2\eta^2}{2}
\end{aligned}\tag{21}$$

Summing the above inequality for $t=1$ to T and using telescoping sum, we have the following inequality

$$\begin{aligned}
\mathbb{E}[f(x_{T+1})]\leq\mathbb{E}[f(x_1)]-\frac{1}{2}\eta\alpha_l\sqrt{\frac{h(1-\beta_2)}{G^2d}}\sum_{t=1}^{T}\|\nabla f(x_t)\|^2-\eta T\alpha_l\sqrt{\frac{h(1-\beta_2)}{G^2d}}\frac{d\epsilon_{H,k}^2}{2b}\\
+\frac{T\eta\alpha_u}{n}\sum_{i\in[n]}\frac{\|\sigma^{(i)}\|_1}{\sqrt{b}}+\sum_{\ell\in[h]}\frac{TL\alpha_u^2\eta^2}{2}+\eta Td\alpha_u\frac{\epsilon_{H,k}}{\sqrt{b}}\ .
\end{aligned}\tag{22}$$

Rearranging the terms of the above inequality, and dividing by $\frac{1}{2}\eta T\alpha_l$. In practice, $\epsilon_{H,k}$ is computed by a mini-batch with size b. Then we have

$$\begin{aligned}
\sqrt{\frac{h(1-\beta_2)}{G^2d}}\frac{1}{T}\sum_{t=1}^{T}\|\nabla f(x_t)\|^2\leq\frac{\mathbb{E}[f(x_1)]-\mathbb{E}[f(x_{T+1})]}{\frac{1}{2}\eta T\alpha_l}-\sqrt{\frac{h(1-\beta_2)}{G^2d}}\frac{d\epsilon_{H,k}^2}{b}\\
+\frac{2\alpha_u}{\alpha_l n}\sum_{i\in[n]}\frac{\|\sigma^{(i)}\|_1}{\sqrt{b}}+\sum_{\ell\in[h]}\frac{L\alpha_u^2\eta}{\alpha_l}+d\frac{2\alpha_u}{\alpha_l}\frac{\epsilon_{H,k}}{\sqrt{b}}\ .
\end{aligned}\tag{23}$$

Combining with the fact that $\mathbb{E}[f(x_1)]-\mathbb{E}[f(x_{T+1})]\leq f(x_1)-f(x^*)$ where x^* is the optimal parameter, we have

$$\begin{aligned}
\sqrt{\frac{h(1-\beta_2)}{G^2d}}\frac{1}{T}\sum_{t=1}^{T}\|\nabla f(x_t)\|^2\leq\frac{f(x_1)-f(x^*)}{\frac{1}{2}\eta T\alpha_l}-\sqrt{\frac{h(1-\beta_2)}{G^2d}}\frac{d\epsilon_{H,k}^2}{b}\\
+\frac{2\alpha_u}{\alpha_l n}\sum_{i\in[n]}\frac{\|\sigma^{(i)}\|_1}{\sqrt{b}}+\sum_{\ell\in[h]}\frac{L\alpha_u^2\eta}{\alpha_l}+d\frac{2\alpha_u}{\alpha_l}\frac{\epsilon_{H,k}}{\sqrt{b}}\ .
\end{aligned}\tag{24}$$

Here b is the batch size, and H is the synchronization interval.

Finally, let $\eta = \sqrt{\frac{2(f(x_1)-f(x^*))}{\alpha_u^2 \|L\|_1 T}}$, $b = T$ we have

$$\frac{1}{T}\sum_{t=1}^{T}\|\nabla f(x_t)\|^2 \leq O\Big(\sqrt{\frac{G^2 d}{h(1-\beta_2)}}\Big[\sqrt{\frac{8(f(x_1)-f(x^*))\|L\|_1}{T}} + \frac{1}{n}\sum_{i\in[n]}\frac{2\|\sigma^{(i)}\|_1}{\sqrt{T}} + d\frac{2\epsilon_{H,k}}{\sqrt{T}}\Big]\Big) \ . \tag{25}$$

□

C. Convergence analysis of SLAMB without gradient synchronization.

In practice, we have the following two minor changes due to the efficiency issue.

1. We skip the synchronization of $g_t^{(i)}$. In this case, Equation 8 should be modified as

$$v_t^{(i)} = \frac{\beta_2 v_{t-1}^{(i)} + (1-\beta_2){g_t^{(i)}}^2}{1-\beta_2^t} \tag{26}$$

2. As mentioned in Appendix E, we simply the scaling function into $\phi_{max} = \phi(\frac{\|x_t^{(\ell,i)} \odot M_t^{(\ell)}\|}{\|u_t^{(\ell,i)} \odot M_t^{(\ell)}\|})$ and $\phi_{min} = \phi(\frac{\|x_t^{(\ell,i)} \odot \neg M_t^{(\ell)}\|}{\|u_t^{(\ell,i)} \odot \neg M_t^{(\ell)}\|})$ and update the model by $x_{t+1}^{(\ell,i)} = x_t^{(\ell,i)} - \tilde{\eta}_t \tilde{\phi} u_t^{(\ell,i)}$.

Under this update scheme, Equation 14 becomes

$$f(x_{t+1}) \leq f(x_t) \underbrace{-\eta \sum_{\ell\in[h]}\sum_{j\in[d_\ell]}\Bigg(\underbrace{\frac{1}{n}\sum_{m\in[n]}\tilde{\phi}\cdot\big[\nabla_\ell f(x_t)\big]_j\cdot\big[u_t^{(m,\ell)}\big]_j}_{T_2}\Bigg)}_{T_1} + \sum_{\ell\in[h]}\frac{L\alpha_u^2\eta^2}{2} \ , \tag{27}$$

we rewrite the bound of T_2 by

$$\begin{aligned}
T_2 =& \frac{1}{n}\sum_{m\in[n]}\tilde{\phi}\cdot\big[\nabla_\ell f(x_t)\big]_j\cdot\big[u_t^{(m,\ell)}\big]_j \\
=& \Big|\big[\nabla_\ell f(x_t)\big]_j\cdot\frac{1}{n}\sum_{m\in[n]}\big[\frac{\tilde{\phi}p_t^{(m,\ell)}}{\sqrt{v_t^{(m,\ell)}}}\big]_j\Big| \\
&+ 2\frac{1}{n}\sum_{m\in[n]}\big[\nabla_\ell f(x_t)\big]_j\cdot\big[\frac{\tilde{\phi}p_t^{(m,\ell)}}{\sqrt{v_t^{(m,\ell)}}}\big]_j\cdot\mathbf{1}_{\{\mathrm{sign}([\nabla_\ell f(x_t)]_j)\neq\mathrm{sign}(\frac{1}{n}\sum_{m\in[n]}[p_t^{(m,\ell)}]_j)\}} \\
\geq& \Big|\frac{1}{G}\big[\nabla_\ell f(x_t)\big]_j\cdot\frac{1}{n}\sum_{m\in[n]}\big[\tilde{\phi}p_t^{(m,\ell)}\big]_j\Big| \\
&+ 2\frac{1}{n}\sum_{m\in[n]}\big[\nabla_\ell f(x_t)\big]_j\cdot\big[\frac{\tilde{\phi}p_t^{(m,\ell)}}{\sqrt{v_t^{(m,\ell)}}}\big]_j\cdot\mathbf{1}_{\{\mathrm{sign}([\nabla_\ell f(x_t)]_j)\neq\mathrm{sign}([\frac{1}{n}\sum_{m\in[n]}p_t^{(m,\ell)}]_j)\}}
\end{aligned} \tag{28}$$

The difference between the above equation and Equation 15 is that the coefficient $\sqrt{\frac{1-\beta_2}{G^2 d_\ell}}$ is changed to $\frac{1}{G}$ since we do not have a $\|u\|$ in denominator. To follow the remaining proof pipeline, we only need to discuss the second inequality in

Equation 17, that is, give an upper bound for $\big|\frac{1}{n}\sum_{m\in[n]}\big[u_t^{(m,\ell)}\big]_j\big|$

$$\begin{aligned}\big|\frac{1}{n}\sum_{m\in[n]}\big[u_t^{(m,\ell)}\big]_j\big| &= \big|\frac{1}{n}\sum_{m\in[n]}\big[\frac{p_t^{(m,\ell)}}{\sqrt{v_t^{(m,\ell)}}+\epsilon}\big]_j\big| \\ &\le \frac{1}{n}\sum_{m\in[n]}\big|\big[\frac{p_t^{(m,\ell)}}{\sqrt{v_t^{(m,\ell)}}+\epsilon}\big]_j\big| \\ &\le \frac{G}{V_{min}+\epsilon}\end{aligned} \tag{29}$$

Then Equation 19 becomes

$$\begin{aligned}&\frac{1}{n}\sum_{m\in[n]}\alpha_u\big|\big[\nabla_\ell f(x_t)\big]_j\big|\cdot\mathbb{P}\big(\operatorname{sign}([\nabla_\ell f(x_t)]_j)\neq\operatorname{sign}([\frac{1}{n}\sum_{m\in[n]}p_t^{(m,\ell)}]_j)\big) \\ &\le \alpha_u\frac{G}{V_{min}+\epsilon}\big(\frac{1}{n}\sum_{i\in[n]}\frac{[\sigma^{(i)}]_j}{\sqrt{b}}+\frac{\epsilon_{H,k}}{\sqrt{b}}\big)\end{aligned} \tag{30}$$

Then Equation 20 becomes

$$\begin{aligned}\mathbb{E}[T_1] &\le -\eta\frac{1}{G}\sum_{\ell\in[h]}\sum_{j\in[d_\ell]}\alpha_l\big(\frac{1}{2}\big[\nabla_\ell f(x_t)\big]_j^2+\frac{\epsilon_{H,k}^2}{2b}\big)+\eta\sum_{\ell\in[h]}\sum_{j\in[d_\ell]}\alpha_u\frac{G}{V_{min}+\epsilon}\big(\frac{1}{n}\sum_{i\in[n]}\frac{[\sigma^{(i)}]_j}{\sqrt{b}}+\frac{\epsilon_{H,k}}{d\sqrt{b}}\big) \\ &= -\eta\alpha_l\frac{d}{G}\big(\frac{1}{2}\|\nabla f(x_t)\|^2+\frac{d\epsilon_{H,k}^2}{2b}\big)+\eta\alpha_u\frac{dG}{V_{min}+\epsilon}\big(\frac{\epsilon_{H,k}}{\sqrt{b}}+\frac{1}{n}\sum_{i\in[n]}\frac{\|\sigma^{(i)}\|_1}{d\sqrt{b}}\big)\end{aligned} \tag{31}$$

Then the Equation 24 is rewritten as

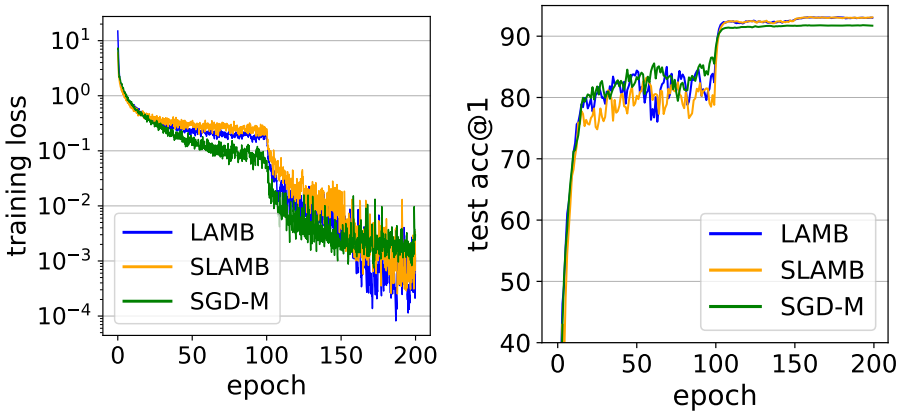
$$\begin{aligned}\frac{d}{G}\frac{1}{T}\sum_{t=1}^{T}\|\nabla f(x_t)\|^2 \le \frac{f(x_1)-f(x^*)}{\frac{1}{2}\eta T\alpha_l}-\frac{d}{G}\frac{d\epsilon_{H,k}^2}{b} \\ +\frac{2\alpha_u}{\alpha_l n}\frac{G}{V_{min}+\epsilon}\sum_{i\in[n]}\frac{\|\sigma^{(i)}\|_1}{\sqrt{b}}+\sum_{\ell\in[h]}\frac{L\alpha_u^2\eta}{\alpha_l}+\frac{dG}{V_{min}+\epsilon}\frac{2\alpha_u}{\alpha_l}\frac{\epsilon_{H,k}}{\sqrt{b}}\ .\end{aligned} \tag{32}$$

And the final convergence is

$$\frac{1}{T}\sum_{t=1}^{T}\|\nabla f(x_t)\|^2 \le O\Big(\frac{G}{d}\Big[\sqrt{\frac{8(f(x_1)-f(x^*))\|L\|_1}{T}}+\frac{G}{V_{min}+\epsilon}\frac{1}{n}\sum_{i\in[n]}\frac{2\|\sigma^{(i)}\|_1}{\sqrt{T}}+\frac{dG}{V_{min}+\epsilon}\frac{2\epsilon_{H,k}}{\sqrt{T}}\Big]\Big)\ . \tag{33}$$

The differences lie on the two coefficients, $\frac{G}{d}$, $\frac{dG}{V_{min}+\epsilon}$. Non-synchronized gradient causes a change of coefficient $\frac{dG}{V_{min}+\epsilon}$, and the simplified implementation changes the first coefficient to $\frac{G}{d}$. Here V_{min} is the minimum value of $\sqrt{v_t^{(m,\ell)}}$ for all t, m, ℓ. For experiments validation, please refer to Table 13 in Appendix H.

D. Comparison of SLAMB and SGD



(a) (b)

Figure 7. ResNet110 on CIFAR10

Table 7. Top-1 test accuracy of training ResNet110 on CIFAR10.

	batch size	**learning rate**	**acc@top-1**
SGD-M	1024	0.03	91.70%
LAMB	1024	0.01	93.15%
SLAMB (k=0.1)	1024	0.01	93.21%
SGD-M	128	0.01	92.92%
LAMB	128	0.003	93.42%
SLAMB (k=0.01)	128	0.003	93.50%

To further demonstrate the wide applicability of SLAMB on different models and datasets, we report the convergence result of training ResNet110 on CIFAR-10. ResNet110 is a relatively small model that can be easily trained on a single machine. We use 4 GPUs and 1024 total batch size to run this experiment. SLAMB is configured as: $\beta_3 = 0.99$, $k = 0.1$, $H = 50$. Here we use a smaller H because CIFAR-10 is a relatively small dataset and using 1024 batch size makes the number of steps per epoch down to 50. Unlike BERT, ResNet110 training is more stable w.r.t large β_3. We gradually decrease the compression ratio k in the first the 3 epochs to stabilize the training. The result is shown in Figure 7. We can see that SLAMB converges similarly to LAMB, and both of them achieve higher test accuracy than the basic Momentum SGD. Note that, if the batch size is smaller, SLAMB can reach 1% compression ratio without losing accuracy.

E. Implementation details.

There are two synchronization steps in Algorithm 1, gradient and momentum. As we explained in section 4.1, we skip the synchronization of g_t in all our experiments. The staleness vector c_t is implemented as a optimizer state variable for each layer's parameter. Note that like other optimizer states such as m_t and v_t, c_t also consumes additional memory depending on the model size. To further improve SLAMB's convergence speed, we average the local gradients among the GPUs within each node before compressing them for inter-node communication. The overhead of such local averaging is negligible since the GPUs are connected via high-speed NVLink within the same node. We simply the scaling function into $\phi_{max} = \phi(\frac{\|x_t^{(\ell,i)} \odot M_t^{(\ell)}\|}{\|u_t^{(\ell,i)} \odot M_t^{(\ell)}\|})$ and $\phi_{min} = \phi(\frac{\|x_t^{(\ell,i)} \odot \neg M_t^{(\ell)}\|}{\|u_t^{(\ell,i)} \odot \neg M_t^{(\ell)}\|})$ and update the model by $x_{t+1}^{(\ell,i)} = x_t^{(\ell,i)} - \tilde{\eta}_t \tilde{\phi} u_t^{(\ell,i)}$. As in practice, we find that ratio $\frac{\|x_t^{(\ell,i)}\|}{\|u_t^{(\ell,i)}\|}$ is more stable among different layers in BERT. The scaling function is implemented as a clipping function: $\phi(z) = \min\{\max\{z, a\}, b\}$.

Table 8. Configuration of BERT models

	BERT-base	**BERT-Large**	**BERT-XLarge**
hidden layer size	768	1024	3072
intermediate layer size	3072	4096	12288
num of attention heads	12	16	16
num of hidden layers	12	24	24
Total number of params	110M	340M	2.8B

F. Memory consumption profiling.

We have profiled the GPU memory consumption and execution time of different optimizers with BERT-Large on a Nvidia V100 GPU. The results are displayed in Table 9. BERT-Large has 336M parameters and it takes around 1.3GB memory to store the model parameters or the gradients. After initialization, the GPU memory consumption is 2.5GB. We can see that SLAMB consumes 2.3GB more memory than LAMB, and the computation overhead is around 20 milliseconds. The additional memory is used for storing the new optimizer state (i.e. c_t) and the intermediate results during computation. Therefore, when using SLAMB to training super large models, one may need to split the input into several micro-batches to decrease the memory consumption and accumulate the gradients from all micro-batches. Note that SLAMB is more memory efficient than the state-of-the-art, 1-bit LAMB. We conjecture that it is because 1-bit LAMB requires additional memory to support its custom communication backend, while SLAMB uses the same NCCL AllReduce primitive as LAMB. All optimizers tested here use the same level of implementation (native PyTorch API).

Table 9. GPU memory consumption and execution time profiling

	Momentum-SGD	**Adam**	**LAMB**	**1-bit LAMB**	**SLAMB**
Time (ms)	20.1	46.8	142.3	168.9	162.7
GPU Mem. Consumption(GB)	3.9	5.5	5.6	10.9	7.9

G. Hyper-parameter tuning.

G.1. BERT

There are several hyper-parameters in SLAMB, including $\beta_1, \beta_2, \beta_3$, learning rate η, compression ratio k, synchronization interval H, scaling function ϕ. β_1, β_2 are originated from LAMB and we keep them as default $\beta_1 = 0.9, \beta_2 = 0.999$. Learning rate η and learning rate scheduler often need to be tuned on specific tasks. We can always use the same learning rate setting for LAMB and SLAMB. β_3 is the only parameter that needs careful tuning. The value range of β_3 should be $(\beta_1, 1)$, and we use 0.95 as the default value. If β_3 is too small then convergence is slow. If it is too large, training may become unstable and diverge. The compression ratio k and synchronization interval H are fixed in most cases. We use $k = 0.1$ and $H = 100$ as default settings. The lower bound and upper bound of the scaling function $\phi(z) = \min\{\max\{z, a\}, b\}$ is specifically configured as $a = 0.01, b = 0.4$ for all layers' updates in BERT training tasks. We find that a, b are not sensitive in other training tasks such as ImageNet training and CIFAR-10 training. We provide the results of hyper-parameter tuning of SLAMB on BERT-Large pre-training task in Table 10.

We summarize other hyper-parameters for BERT training here. For phase1 with seqlen 128, we set learning rate (LR) to 6e-3, and use linear warm-up and polynomial decay (degree=0.5) LR scheduler. The LR warm up steps and total steps are 2000 and 7038, respectively. For phase2 with seqlen 512, we set LR to 4e-3, and use the same scheduler as phase1. The LR warm up steps and total steps are 200 and 1563, respectively. For SQuAD fine-tuning task, we use Adam optimizer with 3e-5 LR and 32 total batch size for 2 epochs.

G.2. Swin Transformer

Table 11 shows the learning rate fine-tuning results of Swin-Base Transformer on ImageNet. We found that as we increase the batch size from 1K to 32K, LAMB scales fairly well when using square root scaling rule to adjust the learning rate, while Adam is unstable with large learning rate. We observe a slight accuracy drop by increasing the batch size. (You et al., 2020) also reports similar accuracy drop when increasing the batch size from 1K to 32K for ImageNet training using

Table 10. Hyper-parameter tuning of SLAMB on BERT-Large pre-training task.

Algorithm	k	β_3	H	**Max F1 score**
LAMB	-	-	-	90.754
SLAMB	**0.5**	0.93	100	90.797
SLAMB	**0.2**	0.93	100	90.799
SLAMB	**0.1**	0.93	100	90.790
SLAMB	**0.05**	0.93	100	89.884
SLAMB	**0.01**	0.93	100	87.847
SLAMB	0.1	**0.9**	100	90.516
SLAMB	0.1	**0.93**	100	90.790
SLAMB	0.1	**0.95**	100	90.623
SLAMB	0.1	**0.97**	100	diverge
SLAMB	0.1	**1**	100	diverge
SLAMB	0.1	0.93	**10**	90.798
SLAMB	0.1	0.93	**100**	90.790
SLAMB	0.1	0.93	**200**	90.776
SLAMB	0.1	0.93	**500**	90.756
SLAMB	0.1	0.93	**1000**	90.201

ResNet50 (77.06% for 1K vs. 76.42% for 32K).

H. Additional results

Algorithm 2 Distributed LAMB with naive gradient compression and error feedback

Input: $x_1 \in \mathbb{R}^d$, learning rate $\{\eta_t\}_{t=1}^T$, parameters $0 < \beta_1, \beta_2 < 1$, $\epsilon > 0$, scaling function ϕ, compression operator C, number of nodes n
(On each node i)
Set $m_0^{(i)} = 0$, $v_0^{(i)} = 0$, $\delta_0^{(i)} = 0$
for $t = 1$ **to** T **do**
  Compute local mini-batch stochastic gradient $g_t^{(i)}$
  $g_t^{(i)} = g_t^{(i)} + \delta_{t-1}^{(i)}$ {**Compensate current gradient with the compression error from last step**}
  $\delta_t^{(i)} = g_t^{(i)} - C(g_t^{(i)})$ {**Compute new compression error in current step**}
  $g_t = \frac{1}{n}\sum_{i=1}^n C(g_t^{(i)})$
  $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$
  $v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$
  $m_t = m_t / (1-\beta_1^t)$
  $v_t = v_t / (1-\beta_2^t)$
  Compute ratio $r_t = \frac{m_t}{\sqrt{v_t}+\epsilon}$
  Compute update $u_t = r_t + \lambda x_t$
  for each layer ℓ **do**
    $x_{t+1}^{(\ell)} = x_t^{(\ell)} - \eta_t \frac{\phi(\|x_t^{(\ell)}\|)}{\|u_t^{(\ell)}\|} u_t^{(\ell)}$
  end for
end for
Output: x

Table 11. Learning rate fine-tuning of Swin Transformer pre-training task at different batch sizes.

Algorithm	batch size	learning rate	warm-up epochs	acc@top-1
Adam	1K	0.001	20	83.20%
Adam	8K	0.001*1	20	82.75%
Adam	8K	0.001*2	20	83.08%
Adam	8K	0.001*3	20	diverge
Adam	8K	0.001*4	20	diverge
Adam	32K	0.001*1	20	80.99%
Adam	32K	0.001*2	20	81.86%
Adam	32K	0.001*3	20	diverge
Adam	32K	0.001*4	20	diverge
LAMB	1K	0.001	20	82.57%
LAMB	8K	0.001*2	20	81.49%
LAMB	8K	0.001*2.8	20	82.54%
LAMB	8K	0.001*3.5	20	82.75%
LAMB	8K	0.001*4	20	83.09%
LAMB	8K	0.001*6	20	diverge
LAMB	8K	0.001*8	20	diverge
LAMB	32K	0.001*6	20	81.98%
LAMB	32K	0.001*8	20	82.46%
LAMB	32K	0.001*10	20	82.74%
LAMB	32K	0.001*12	20	82.74%
LAMB	32K	0.001*14	20	diverge
LAMB	32K	0.001*16	20	diverge
LAMB	32K	0.001*10	5	82.62%
LAMB	32K	0.001*10	20	82.74%
LAMB	32K	0.001*10	40	82.21%

Table 12. Per-step training time breakdown for BERT pre-training (seqlen 128) and Swin-Base transformer training. We profile the computation time (forward and backward), communication time, and the ratio of communication time to total time per step.

Model	GPU	Bandwidth (Gbps)	#node	#GPU	batch size per GPU	Grad. accum.	batch size	Comp.(s)	Comm.(s)	Total (s)	Comm. / Total
BERT-Large	V100	**1**	32	128	64	8	64K	2.584	12.690	15.290	**83%**
		10	32	128	64	8	64K	2.592	1.246	3.846	32%
		100	32	128	64	8	64K	2.588	0.238	2.838	8%
BERT-Large	V100	10	32	128	64	**4**	**32K**	1.292	1.218	2.518	48%
		10	32	128	64	**2**	**16K**	0.644	1.238	1.888	65%
		10	32	128	64	**1**	**8K**	0.318	1.255	1.580	**79%**
BERT-XLarge	A100	100	128	**1024**	64	1	**64K**	1.165	1.047	2.220	47%
		100	64	**512**	64	1	**32K**	1.147	1.043	2.201	47%
		100	32	**256**	64	1	**16K**	1.113	0.894	2.019	42%
Swin-Base	V100	10	32	128	64	4	32K	1.401	0.524	1.933	27%
		10	32	128	64	2	16K	0.779	0.533	1.325	40%
		10	32	128	64	1	8K	0.412	0.587	1.012	**58%**

Table 13. Accuracy validation for SLAMB with and without gradient synchronization. We can see that skipping gradient synchronization in SLAMB does not affect the training quality of various tasks.

Task	**LAMB**	**SLAMB (Algorithm 1)**	**SLAMB w/o gradient synchronization**
Top-1 acc % (ResNet110 on Cifar10)	93.15	93.14	93.21
Top-1 acc % (Swin Transformer on ImageNet)	83.09	83.02	83.10
SQuAD Avg. F1 Score (BERT-Large Pre-training)	90.582	90.680	90.646

Table 14. SLAMB scaling performance at different compression ratio (k) in BERT-XL Pre-training task. We use 512 A100 GPUs in this experiment. Warmup is a critical step for 1-bit LAMB to achieve good convergence in practice, however, it significantly affects the scaling performance.

Algorithm	k	**Volume Reduction ($\times$ 100%)**	**Throughput - samples / s**	**Scaling efficiency %**
LAMB	-	-	14813	52.0
SLAMB	0.5	2.0	19300	67.7
SLAMB	0.25	3.8	22951	80.5
SLAMB	0.1	9.1	25915	90.9
SLAMB	0.05	16.6	26845	94.1
SLAMB	0.01	50.0	27660	97.0
SLAMB	0.001	90.9	27798	97.5
1-bit LAMB	-	4.6	22930	80.4
1-bit LAMB w/o warmup	-	16.0	26552	93.1