Communication-efficient Distributed Learning for Large Batch Optimization

Rui Liu [1] **Barzan Mozafari** [1]

Abstract

Many communication-efficient methods have been proposed for distributed learning, whereby gradient compression is used to reduce the communication cost. However, given recent advances in large batch optimization (e.g., large batch SGD and its variant LARS with layerwise adaptive learning rates), the compute power of each machine is being fully utilized. This means, in modern distributed learning, the per-machine computation cost is no longer negligible compared to the communication cost. In this paper, we propose new gradient compression methods for large batch optimization, JOINTSPAR and its variant JOINTSPAR-LARS with layerwise adaptive learning rates, that jointly reduce both the computation and the communication cost. To achieve this, we take advantage of the redundancy in the gradient computation, unlike the existing methods compute all coordinates of the gradient vector, even if some coordinates are later dropped for communication efficiency. JOINTSPAR and its variant further reduce the training time by avoiding the wasted computation on dropped coordinates. While computationally more efficient, we prove that JOINTSPAR and its variant also maintain the same convergence rates as their respective baseline methods. Extensive experiments show that, by reducing the time per iteration, our methods converge faster than state-of-the-art compression methods in terms of wall-clock time.

1. Introduction

Distributed machine learning has drawn much attention as an important approach to scale up machine learning models. Since machine learning model training is notoriously time consuming, most distributed machine learning methods are focused on parallelizing *stochastic gradient descent* (SGD) algorithms (Shalev-Shwartz & Zhang, 2013; Schmidt et al., 2017; Johnson & Zhang, 2013). This is because SGD (and its variants) are commonly used to train machine learning models, including deep learning models (Sutskever et al., 2013; Duchi et al., 2011; Kingma & Ba, 2014). In SGD, let $f(x)$ be the loss function that we hope to minimize, where $x \in \mathbb{R}^D$ is the parameter vector of a machine learning model. At each iteration t, we have access to the unbiased gradient vector $g_t = g(x_t)$ such that $\mathbb{E}[g(x_t)] = \nabla f(x_t)$, where x_t is the parameter vector at current iteration. A standard SGD uses the following update rule to get x_{t+1} for the next iteraiton: $x_{t+1} = x_t - \alpha_t g_t$, where α_t is a scalar for step size.

A common method to parallelize SGD is to use several machines to collectively minimize f. Each machine has one partition of the whole training dataset, and maintains a local copy of the parameter vector x_t. At each iteration, it obtains a new gradient vector g_t according to its local parameter copy and data partition. After each iteration, all machines broadcast their local gradient vectors to their peers, so that gradient vectors can be aggregated to compute the new parameter vector x_{t+1}. This synchronization step of aggregating gradients can also be relaxed with asynchronous update methods (Leblond et al., 2018; Recht et al., 2011; Nguyen et al., 2018; Liu et al., 2015). It has been observed that gradient communication in each iteration is a significant performance bottleneck in practical applications (Chilimbi et al., 2014; Seide et al., 2014; Strom, 2015).

Existing Approaches. The main mechanism for alleviating the communication overhead is to compress the gradients before synchronously[1] broadcasting them to peer machines. There are two popular approaches to compressing the gradients: one is to quantize gradients (into lower precision representation) (De Sa et al., 2015; Alistarh et al., 2017; Zhou et al., 2016; Wen et al., 2017; De Sa et al., 2017; Zhang et al., 2017; Rastegari et al., 2016; De Sa et al., 2018; Bernstein et al., 2018), and the other is to sparsify gradients by dropping some of their coordinates (Mania et al., 2015; Leblond et al., 2016; Aji & Heafield, 2017; Lin et al., 2017; Chen et al., 2018a; Renggli et al., 2018; Tsuzuku et al., 2018;

[1]Computer Science and Engineering, University of Michigan, Ann Arbor. Correspondence to: Rui Liu <ruixliu@umich.edu>, Barzan Mozafari <mozafari@umich.edu>.

Proceedings of the 39th International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

[1]State-of-the-art gradient compression often uses synchronous updates. We will also discuss asynchronous updates in Section 2.

Stich et al., 2018). However, both approaches must compute the gradients first and then apply their compression. Specifically, assume a gradient vector $g = [g_1, \cdots, g_d, \cdots, g_D]$, where the d-th coordinate g_d is the gradient value with respect to the d-th model parameter and D is the number of model parameters. The compressed gradient $Q(g)$ is usually obtained by compressing g according to a probability vector $p = [p_1, \cdots, p_d, \cdots, p_D]$. The variance of $Q(g)$ is defined as $\mathbb{E}\sum_{d=1}^{D}[Q(g)_d^2]$, and the sparsity of $Q(g)$ is defined as $\mathbb{E}[\|Q(g)\|_0]$, where $Q(g)_d$ is the d-th coordinate of $Q(g)$. Different methods differ in the way they choose p. For example, a popular technique, called TERNGRAD (Wen et al., 2017), chooses each p_d to be proportional to the absolute value of g_d, i.e., $p_d = \frac{|g_d|}{\|g\|_\infty}$, and defines $Q(g)$ as $Q(g)_d = \|g\|_\infty \operatorname{sign}(g_d) Z_d$, where Z_d is a Bernoulli random variable with $\mathbb{P}(Z_d = 1) = p_d = |g_d|/\|g\|_\infty$. Since $\|g\|_\infty$ is the same for all coordinates, TERNGRAD effectively quantizes each coordinate into one of the $\{-1, 0, 1\}$ values. For TERNGRAD, the variance of $Q(g)$ is $\|g\|_1\|g\|_\infty$, and the sparsity of $Q(g)$ is $\frac{\|g\|_1}{\|g\|_\infty}$, both of which are fixed for any given g. Other techniques, such as GSPAR (Wangni et al., 2018), provide the flexibility to trade off between variance and sparsity by choosing p through solving a linear problem: $min_p \sum_{d=1}^{D} p_d \quad \text{s.t.} \sum_{d=1}^{D} \frac{g_d^2}{p_d} \leq (1+\epsilon)\sum_{d=1}^{D} g_d^2$, where ϵ is the parameter that allows the user to control this tradeoff. There are other techniques too, that apply SVD decomposition to g (Wang et al., 2018a), consider multiple-level quantization (Alistarh et al., 2017), use additional memory to compensate for the sparsification error (Stich et al., 2018), or extend to decentralized settings where machines communicate only with their neighbors (Tang et al., 2018; Vogels et al., 2020).

These gradient compression methods are quite effective at compressing the size of the gradients, and thus reducing the *communication time* (i.e., the time spent on communicating the gradients with other machines over the network) in traditional distributed learning. Recently, there has been a surge in using large batch optimization to fully utilize the compute power, particularly for training deep neural networks on large scale datasets (Goyal et al., 2017; Golmant et al., 2018; Lin et al., 2020; Keskar et al., 2016; Smith et al., 2017). It has been shown that using larger batch sizes with synchronous SGD variants can significantly speed up the training (Goyal et al., 2017; Akiba et al., 2017; You et al., 2017; 2019). The *computation time* (i.e., time spent on computing the gradients that will be sent over the network) is no longer negligible due to large batch sizes. However, existing gradient compression methods cannot reduce the computation time. This is because the computation time is mainly determined by how the gradients are computed based on the batch of examples in each iteration. In fact, the computation time slightly increases due to the additional operations of compressing and uncompressing the gradients. This is why in many real life settings, the computation time is reported to be much larger than the communication time (Akiba et al., 2017). For example, as shown in Table 1, when training AlexNet (Krizhevsky et al., 2012) on the ImageNet dataset (Deng et al., 2009) with existing gradient compression methods (e.g., TERNGRAD and GSPAR), the computation time is at least twice the communication time. Therefore, to achieve the best of both worlds and reduce the total training time, we need to reduce the computation while retaining the benefits of gradient compression techniques.

Table 1. Breakdown of computation time (t_{comp}) versus communication time (t_{comm}) per epoch (in seconds) for vanilla SGD versus gradient compression methods in large batch optimization. The results are obtained by training AlexNet on ImageNet dataset using 8 machines with P100 GPUs connected with a 100Gb/s InfiniBand fabric (batch size is set as a large value, i.e., 1024 for each machine). While gradient compression methods (e.g., TERNGRAD and GSPAR) reduce communication time by 2-3x, the computation time becomes the dominant part. Note that the slight increase in computation time of the gradient compression methods is due to time spent compressing and uncompressing gradients.

Method	t_{comp}	t_{comm}	t_{comp}/t_{comm}
SGD	1021	1156	0.88
TERNGRAD	1205	446	2.70
GSPAR	1079	519	2.08

Our Approach. In this paper, we propose a new gradient compression method, JOINTSPAR, that aims to additionally reduce the computation time while keeping the goal of reducing the communication time. Our key insight is that our method can take advantage of the redundancy in gradient computation in existing methods. Specifically, all existing methods need to compute all coordinates in a gradient vector at each iteration before sparsifying it, even though some coordinates might be dropped by sparsification. The training time could be further reduced if only the coordinates that would be communicated were computed. JOINTSPAR integrates sparsity in the gradient computation, thereby avoiding redundant gradient computation and achieving joint sparsification of gradient computation and communication.

Specifically, rather than recomputing the probability vector at each iteration, we maintain a probability distribution p over all coordinates which is being updated across training iterations. The optimal solution to the probability distribution at each iteration requires us to compute the gradient value for each coordinate, which is exactly what we aim to avoid in the first place. We cast the problem of learning an optimal probability distribution as an adversarial multi-armed bandit problem. We use a multi-armed bandit method to update the distribution p, based on the partial information available at each iteration (i.e., the gradient values for coordinates that will not be dropped). The bandit method for learning p is embedded into the parallelized SGD that minimizes the loss function with respect to x, so that their iterations are synchronized. At each iteration, JOINTSPAR performs one update step for parallelized SGD with respect to x, and one update step for the bandit method with respect to p. The gradient coordinates are chosen according to the current value of p at the beginning of every iteration, and only the chosen coordinates are computed and communicated. This joint sparsification of gradient computation and communication greatly reduces the end-to-end training time of JOINTSPAR. In addition, JOINTSPAR also provides users with the flexibility to choose the sparsity budget. As with GSPAR, this can be used to trade off variance and sparsity. We prove that JOINTSPAR achieves the same convergence rate in iteration numbers as existing state-of-the-art methods for communication-efficient distributed learning (Bernstein et al., 2018; Tang et al., 2018). We empirically verify that, compared to state-of-the-art methods, JOINTSPAR converges faster in terms of wall-clock time, because it requires less time per iteration.

In summary, we make the following contributions:

- We observe that, for large batch optimization, the computation time is no longer negligible, responsible for a significant fraction of their total training time. We propose new gradient compression method for large batch optimization, JOINTSPAR and its variant JOINTSPAR-LARS, which jointly sparsify both computation and communication of the gradients. Similar to state-of-the-art (Wangni et al., 2018; Wang et al., 2018a), our methods provide users the flexibility to trade off variance and sparsity.
- We prove that JOINTSPAR has the same convergence rate in terms of *the number of iterations* as the state-of-the-art methods for SGD-based distributed learning (Bernstein et al., 2018; Tang et al., 2018), namely $O(\frac{1}{\sqrt{T}})$, where T is the number of iterations. We also prove that when applied to recently proposed large batch optimizer with layerwise adaptive learning rates LARS, the variant JOINTSPAR-LARS enjoys the same convergence rate as the original LARS.
- Our extensive experiments on several benchmark datasets (e.g., MNIST, CIAF10, CIFAR100 and ImageNet) using various models (e.g., convolutional neural networks, and residual neural networks) show that our methods spend significantly less time per iteration, and converge faster than state-of-the-art methods in terms of *wall-clock time*.

2. Related Work

Training Efficiency. For efficient model training, especially in the single machine setting, weight compression techniques whereby the precision of the weights are reduced to save compute and memory overheads have been shown very effective (Micikevicius et al., 2017; Sakr et al., 2019). This is because memory usage can be decreased by using fewer bits to store the same number of values, and compute time can also be reduced on processors that offer higher throughput for reduced precision arithmetic operations. For example, recent work has successfully trained deep neural networks using 8-bit floating point numbers without sacrificing accuracy (Wang et al., 2018b). Another line of work has shown that sampling techniques can be used to accelerate model training (Zhao & Zhang, 2015; Katharopoulos & Fleuret, 2018). For example, Adam with bandit sampling is proved to converge faster than the original Adam when the feature distribution in the training set is skewed (e.g., exponentially faster in terms of the size of the training set when the feature follows a doubly heavy-tailed distribution) (Liu et al., 2020). The key idea is that, by sampling those important examples more often, the training process can be accelerated due to improved sample efficiency. Recent works (Clark et al., 2020; Liu & Mozafari, 2022) have shown that improving sample efficiency is also useful for efficiently training transformer-based language models (Devlin et al., 2018). Our focus in this paper is on efficient model training in a distributed setting through reducing both computation and communication costs.

Communication Efficiency. In the distributed setting where we use multiple machines to collectively train a model, frequent communication happens among these machines to exchange information that is local at each machine. The most studied type of communication is aggregating local gradients to get the global gradient which is needed for updating the model parameters at each training iteration, because distributed training in the data parallel fashion is widely adopted (Li et al., 2014). In this paper, we also focus on this type of communication (i.e., aggregating local gradients). There has been significant progress on using various compression techniques to reduce the communication cost. The convergence rate of these methods under different assumptions has been studied in the literature (Alistarh et al., 2018; Koloskova et al., 2019; Vogels et al., 2019; Lin et al., 2017; Zheng et al., 2019; Yu et al., 2018; Xu et al., 2020; Chen et al., 2018b; Liu et al., 2019; Wu et al., 2018; Tang et al., 2019; 2021). Some specialized systems have also been developed that can support efficient communication for gradient aggregation based on various compression methods (Fei et al., 2020; Renggli et al., 2019; Li et al., 2020). In addition to gradient compression methods, asynchronous updates provide an alternative means of alleviating the communication overhead (Leblond et al., 2018; Recht et al., 2011; Lian et al., 2015). Asynchronous updates are particularly useful in heterogeneous clusters, where some machines are considerably slower than others. These relatively slow machines are called stragglers. Asyn-

Table 2. Notation summary.

Symbol	Description	Symbol	Description
M	the total number of machines/processes	p_t^m	probability vector for machine m at iteration t
f	loss function	$p_{t,d}^m$	d-th coordinate of probability vector p_t^m
x	parameter vector	Z_t^m	selector vector for machine m at iteration t
$\mathcal{X}$	space of the parameter vector	S_t^m	active set for machine m at iteration t
D	number of blocks of the parameter vector	$Q(g_t^m)$	sparsified version of g_t^m
T	total number of training iterations	$Q(g_t^m)_d$	d-th block of $Q(g_t^m)$
g_t^m	full gradient vector for machine m at iteration t	s	sparsity budget, i.e., $\mathbb{E}\left[\|Q(g_t^m)\|_0\right]$
$g_{t,d}^m$	d-th block of gradient vector g_t^m	S_d	smoothness constant for x_d
$g_{t,[d]}^m$	zero vector with $g_{t,d}^m$ at d-th block	L	upper bound on $\|g_{t,[d]}^m\|, \forall t, d, m$

chronous updates can introduce additional noise if there are too many stragglers. To avoid asynchronous noise and mitigate stragglers, an alternative solution is to use synchronous updates with backup workers (Chen et al., 2016) proposed to use. Although asynchronous updates could in principle be combined with gradient compression, state-of-the-art gradient compression methods still use synchronous updates (Wang et al., 2018a; Alistarh et al., 2017; Bernstein et al., 2018; Wen et al., 2017). This is primarily to avoid the additional asynchronous noise, especially that stragglers are not a major concern. Thus, to be consistent with state-of-the-art gradient compression methods, we also focus on synchronous updates in the rest of this paper. It is worth noting that there are other types of communication, such as passing neural activation values across machines due to model parallelism (Shazeer et al., 2018; Liu et al., 2022; Gupta et al., 2021). Other techniques such as a dropout variant called Gating Dropout (Liu et al., 2022) have been proposed to reduce the cost of other communication types.

3. Joint Sparsification of Gradient Computation and Communication

Consider the following nonconvex [2] optimization problem: $\min_{x\in\mathcal{X}} f(x)$, where $\mathcal{X}$ is the parameter space. The goal in distributed learning is to solve this problem using M machines/processes[3], where each machine stores a partition of the entire training dataset. Without loss of generality, assume parameter vector x can be decomposed into D blocks, i.e., $x = [x_1, \cdots, x_d, \cdots, x_D]$ where D is the total number of parameter blocks. The reason that this is without loss of generality is because, in neural network models, their layerwise architecture provides a natural block decomposition, i.e., all the paramaters from one layer form a block. For the rest of this paper, we consider this layerwise formulation rather than the coordinate-based formulation, because (1) it is to be consistent with the recently proposed large batch optimizers with layerwise adaptive learning rates (You et al., 2017; 2019) (see Section 3.2), (2) it allows efficient implementation on top of modern deep learning frameworks such as PyTorch (see Section 3.3). Suppose the full gradient of f with respect to x for machine m at iteration t is g_t^m, and $g_{t,d}^m$ is the gradient of f with respect to x_d. Denote $g_{t,[d]}^m = [0, \cdots, g_{t,d}^m, \cdots, 0]$, where all entries are zero except $g_{t,d}^m$ at the d-th block. Thus, $g_t^m = \sum_{d=1}^D g_{t,[d]}^m$.

Our goal in this paper is to compress g_t^m by integrating sparsity in the gradient computation. To this end, we maintain a probability distribution $p_t^m = [p_{t,1}^m, \cdots, p_{t,d}^m, \cdots, p_{t,D}^m]$ over the gradient blocks. To sparsify (i.e., compress) the gradient, we keep $g_{t,d}^m$ with probability $p_{t,d}^m$, and drop it with probability $1 - p_{t,d}^m$. The sparsified gradient vector is then represented as $Q(g_t^m) = \sum_{d=1}^D \frac{g_{t,[d]}^m}{p_{t,d}^m} Z_{t,d}^m$, where $Z_{t,d}^m$ is a Bernoulli random variable with $\mathbb{P}(Z_{t,d}^m = 1) = p_{t,d}^m$, for all $1 \le d \le D$. We can verify that $Q(g_t^m)$ is unbiased: $\mathbb{E}[Q(g_t^m)_d] = \frac{g_{t,d}^m}{p_{t,d}^m}\mathbb{E}[Z_{t,d}^m] = \frac{g_{t,d}^m}{p_{t,d}^m} p_{t,d}^m = g_{t,d}^m$.

To avoid redundant gradient computations, at the beginning of each iteration, we instantiate the random variable $Z_{t,d}^m$ by setting $Z_{t,d}^m = 1$ with probability $p_{t,d}^m$ and $Z_{t,d}^m = 0$ with probability $1 - p_{t,d}^m$. Since we will drop any gradient block d for which $Z_{t,d}^m = 0$ during the communication, we also skip computing these dropped blocks. In other words, we only compute $g_{t,d}^m$ where $d \in S_t^m = \{d : Z_{t,d} = 1, \forall 1 \le d \le D\}$. We call the set S_t^m the active set as it indicates the gradient blocks that we will actively compute and communicate. The key notations used in this paper are listed in Table 2. Because gradient computation accounts for a significant portion of the time spent in each iteration, computing only the necessary gradient blocks will be hugely advantageous. For our analysis, we make the following common assumptions (Beck & Teboulle, 2003; Darzentas, 1984; You et al., 2019).

Assumption 1. (1) The objective function f is S_d-smooth with respect to x_d; (2) The gradient $g_{t,[d]}^m$ is upper bounded by L, i.e., $L \ge \|g_{t,[d]}^m\|, \forall t, d, m$; (3) For all $x, y \in \mathcal{X}$,

[2]Due to space constraint, here we focus on nonconvex objective functions, which are the case for neural network models. Please refer to the appendix for our results on a tigher bound on convex objective functions.

[3]We use machine and process interchangeably in this paper.

we have $B(x, y) \leq R^2$, where $B(x, y)$ is the Bregman divergence between x and y.

We use $S = [S_1, S_2, \cdots, S_D]$ to denote the D-dimensional vector of Lipschitz smoothness constants. Let $S_\infty = \max_d S_d$. For simplicity, we also assume the function value at the initial point is upper bounded by R^2, i.e., $f(x_1) \leq R^2$. The probability distribution p_t^m will govern how many gradient blocks are computed and communicated. A good choice of p_t^m should place high probability values on those blocks that have the largest impact on the ultimate convergence of the model. Before we discuss how to determine $p_{t,d}^m$, let us assume that there is an oracle that provides some arbitrary value for $p_{t,d}^m$. We call this ideal method as ORACLESPAR. Its pseudocode is the same as Algorithm 2 except that line 9 should be replaced by using an oracle to get $p_{t+1,d}^m$. Then, we have the following theorem[4] regarding its convergence rate.

Theorem 1. *Under Assumption 1,* ORACLESPAR *achieves*

$$\begin{aligned} &\mathbb{E}\sum_{t=1}^{T}\sum_{d=1}^{D}\|f'_d(x_t)\|^2 \\ \leq &\frac{R^2}{\alpha_x} + \frac{S_\infty \alpha_x}{2M^2}\sum_{m=1}^{M}\sum_{t=1}^{T}\mathbb{E}\left[\sum_{d=1}^{D}\frac{\|g_{t,[d]}^m\|^2}{p_{t,d}^m}\right] \end{aligned} \tag{1}$$

where $f'(x_t) = [f'_1(x_t), \cdots, f'_d(x_t), \cdots, f'_D(x_t)]$ is the gradient of f with respect to x_t and α_x is the step size.

According to Theorem 1, the second term on the right hand side of Equation 1 depends on the values of probabilities $p_{t,d}^m$. For a fast convergence rate, we wish to choose the $p_{t,d}^m$ values such that the second term is minimized. Interestingly, the second term is the same as the gradient variance in existing methods (Wangni et al., 2018; Wang et al., 2018a). It can be shown that, for every iteration t, the optimal distribution $p_{t,[d]}^m$ is proportional to the gradient value of individual coordinate (Needell et al., 2014; Alain et al., 2015). Formally speaking, for any t, the optimal solution to the problem

$$arg \min_{\sum_{d=1}^{D} p_{t,d}=1} \sum_{d=1}^{D}\frac{\|g_{t,[d]}\|^2}{p_{t,d}} \tag{2}$$

is $p_{t,d} = \frac{\|g_{t,[d]}\|}{\sum_{d=1}^{D}\|g_{t,[d]}\|}, \forall d$. However, it is computationally prohibitive to get the optimal solution, because we need to compute the gradient values for all blocks (thus all coordinates), which is exactly what we aim to avoid in the first place. Instead, we use a multi-armed bandit method (McMahan & Blum, 2004; Dani et al., 2008) to learn this distribution from the partial information that is available during training. Here, partial information is the gradient values for blocks in the active set S_t^m. The multi-armed bandit method maintains the distribution over all blocks, and keeps updating this distribution at every training iteration.

[4]All omitted proofs can be found in the appendix of this paper.

3.1. Bandit Method for Distribution Learning

Algorithm 1 Distribution update for p_t^m

```
1:  Function: update $(p_t^m, S_t^m, \{g_{t,[d]}^m\}_{d \in S_t^m})$
2:      for $d \leftarrow 1$ to $D$ do
3:          if $d \in S_t^m$ then
4:              $\tilde{l}_{t,d}^m = -\frac{\|g_{t,[d]}^m\|^2}{(p_{t,d}^m)^2} + \frac{L^2}{p_{min}^2}$;
5:          else
6:              $\tilde{l}_{t,d}^m = 0$;
7:          end if
8:          $w_{t,d}^m = p_{t,d}^m \exp(-\alpha_p \tilde{l}_{t,d}^m / p_{t,d}^m)$;
9:      end for
10:     $p_{t+1}^m = argmin_{q \in \mathcal{P}} D_{kl}(q \| w_t^m)$;
11:     Return: $p_{t+1}^m$
```

Our goal is to minimize $\sum_{t=1}^{T}\mathbb{E}\left[\sum_{d=1}^{D}\frac{\|g_{t,[d]}^m\|^2}{p_{t,d}^m}\right]$ by using a multi-armed bandit method on each machine m. We use a bandit method based on EXP3 (Auer et al., 2002) but extended to handle multiple actions at every iteration. The pseudocode is described in Algorithm 1. Note that L is assumed to be the upper bound on the gradient norm (see Assumption 1). In addition, we define the set $\mathcal{P}$ as $\mathcal{P} = \{p \in \mathbb{R}^D : \sum_{d=1}^{D} p_d = s, p_d \geq p_{min}, \forall 1 \leq d \leq D\}$, where s is the *sparsity budget* and p_{min} is a constant lower bound. Both s and p_{min} are parameters directly controlled by the user, where $0 < s \leq D$ and $0 < p_{min} \leq 1$. The sparsity budget s serves as a knob for the user to gradually transition from no gradient computation (i.e., when $s = 0$) to full gradient computation (i.e., when $s = D$). In line 10, $D_{kl}(q\|w)$ is the Kullback-Leibler divergence between q and w.

To further illustrate the distribution update algorithm from the perspective of the bandit setting, the number of arms is D, where each arm corresponds to a parameter block. Selecting an active set S_t^m of blocks for which gradient computation and communication will be performed, is the same as choosing $|S_t^m|$ arms to pull. At iteration t, the loss of pulling the arm d is $\tilde{l}_{t,d}^m$, which is defined in line 4 of Algorithm 1. Each time we update the distribution, we only pull these arms from the set S_t^m, which are blocks for which gradient computation and communication will be performed. From the definition of $\tilde{l}_{t,d}^m$ as in line 4 of Algorithm 1, we can see that the loss $\tilde{l}_{t,d}^m$ is always nonnegative, and is inversely correlated with $\|g_{t,[d]}^m\|$. This implies that a block with small gradient norm will receive large loss value, resulting in its weight getting decreased. In other words, blocks with larger gradient norms are more likely to be selected in the active set. This aligns with existing gradient compression methods in the sense that blocks with small gradient norms tend to get dropped to reduce the communication cost. Using a sum tree structure to store these weights, we can efficiently update

p_t^m in $O(|S_t^m|\log D)$ time (Namkoong et al., 2017). With the logarithm dependence on D, the overhead of the bandit method is small even for a large neural network model.

Theorem 2. *Under Assumption 1, if we set the step size* $\alpha_p = \sqrt{\frac{2R^2 p_{min}^4}{DTL^4}}$, *Algorithm 1 achieves the following convergence rate:*

$$\mathbb{E}\sum_{t=1}^{T} l(p_t^m) - \min_{p\in\mathcal{P}} \mathbb{E}\sum_{t=1}^{T} l(p) \le \frac{RL^2}{p_{min}^2}\sqrt{2DT} \tag{3}$$

where $l(p_t^m) = \sum_{d=1}^{D} \frac{\|g_{t,[d]}^m\|^2}{p_{t,d}^m}$.

Using a bandit method to keep updating the distribution p_t^m (which is initialized uniformly, i.e., $p_1^m = \frac{s}{D}$) as in Algorithm 1, we can now introduce our distributed learning algorithm that jointly sparsifies gradient computation and communication. We call our algorithm JOINTSPAR, which is presented in Algorithm 2.

Algorithm 2 Our distributed learning method JOINTSPAR (for each worker m)

```
1:  for $t = 1, \cdots, T$ do
2:      Instantiate the value for $Z_{t,d}^m$, where $Z_{t,d}^m$ is a Bernoulli random variable with $\mathbb{P}(Z_{t,d}^m = 1) = p_{t,d}^m$, for all $1 \le d \le D$;
3:      Determine active set $S_t^m = \{d : Z_{t,d} = 1, \forall 1 \le d \le D\}$;
4:      Compute gradient $g_{t,d}^m$ for all $d \in S_t^m$;
5:      Correct the gradients to make them unbiased with $\hat{g}_{t,d}^m = \frac{g_{t,d}^m}{p_{t,d}^m}$;
6:      Construct sparsified gradient $Q(g_t^m) = [q_1, \cdots, q_D]$, where $q_d = \hat{g}_{t,d}^m$ if $d \in S_t^m$, otherwise $q_d = 0$;
7:      Encode $Q(g_t^m)$ in sparse format and broadcast it to other peer machines;
8:      Receive $Q(g_t^{m'})$ from peer machines $1 \le m' \le M$, and update the parameter vector $x_{t+1} = x_t - \alpha_x \frac{1}{M}\sum_{m'=1}^{M} Q(g_t^{m'})$;
9:      $p_{t+1,d}^m = \text{update}(p_t^m, S_t^m, \{g_{t,[d]}^m\}_{d\in S_t^m})$;
10: end for
```

Combining both Theorem 1 and 2, we can prove the convergence rate of JOINTSPAR, as stated in Theorem 3.

Theorem 3. *Under Assumption 1,* JOINTSPAR *achieves*

$$\sum_{t=1}^{T}\sum_{d=1}^{D} \mathbb{E}\|f_d'(x_t)\|^2 \le RLD\sqrt{\frac{S_\infty T}{2Ms}} + \frac{R}{LD}\sqrt{\frac{S_\infty s}{2MT}}\bar{R}^* + \frac{R^2 L}{p_{min}^2}\sqrt{\frac{S_\infty s}{MD}} \tag{4}$$

where

$$\bar{R}^* = \max_m \min_{p\in\mathcal{P}} \mathbb{E}\left[\sum_{t=1}^{T}\sum_{d=1}^{D} \frac{\|g_{t,[d]}^m\|^2}{p_d}\right]. \tag{5}$$

We now further bound Equation 5 to show that our convergence rate is $O(\frac{1}{\sqrt{T}})$. Since we have assumed the upper bound on gradient $L \ge \|g_{t,[d]}^m\|, \forall t, d, m$, we could get the following upper bound on $\bar{R}^*$.

Lemma 1. *Under Assumption 1, then we have*

$$\bar{R}^* = \max_m \min_{p\in\mathcal{P}} \mathbb{E}\left[\sum_{t=1}^{T}\sum_{d=1}^{D} \frac{\|g_{t,[d]}^m\|^2}{p_d}\right] \le O(\frac{TL^2D^2}{s}). \tag{6}$$

Theorem 4. *Under Assumption 1, our distributed learning method* JOINTSPAR *has the following convergence rate*

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{d=1}^{D} \mathbb{E}\|f_d'(x_t)\|^2 \le O\left(RLD\sqrt{\frac{2S_\infty}{MsT}}\right) \tag{7}$$

where f is the convex objective function and s is the sparsity budget .

This theorem implies that the average convergence rate of JOINTSPAR is $O(\frac{1}{\sqrt{T}})$, which is the same as the state-of-the-art methods for distributed learning (Bernstein et al., 2018; Tang et al., 2018). As we increase the sparsity budget s, the right hand side of Equation 7 becomes smaller, implying a faster convergence rate. This is expected, as a larger value of s allows our method to communicate gradient vectors that are less sparsified, hence carrying more information at each iteration of the gradient communication. On the other hand, computing and communicating less sparsified gradient vectors will be more expensive in practice in terms of wall-clock time. This tradeoff can be directly controlled by the user, via setting the s value.

3.2. Large Batch Optimizer Variant with Layerwise Adaptive Learning Rates

Our method JOINTSPAR as shown in Algorithm 2 is based on synchronous SGD. Combined with some tricks such as linear scaling and warmup, synchronous SGD was able to drastically reduce the training time of ImageNet training with ResNet-50 from 29 hours to 1 hour using a large batch size (up to 8192) (Goyal et al., 2017) . To further increase the batch size without compromising the test accuracy, layerwise adaptive learning rates have been recently introduced for large batch optimizers. The most prominent optimizers are LARS and LAMB (You et al., 2017; 2019). In this section, we dicuss a variant of JOINTSPAR by applying our bandit-based compression method to LARS and show that the same convergence rate as with the original LARS can still be achieved [5]. We call this variant JOINTSPAR-LARS. Its pseudocode is the same with Algorithm 2, except that the parameter update rule in line 8 becomes $x_{t+1,d} = x_{t,d} - \alpha_x \frac{\phi\|x_{t,d}\|}{\|m_{t,d}\|} m_{t,d}$ for $1 \le d \le D$.

[5] Due to space limit, we omit discussion on LAMB. But similar analysis can be extended to LAMB.

Here, momentum $m_{t,d}$ is updated by $m_{t,d} = \beta_1 m_{t-1,d} + (1-\beta_1)\left(\frac{1}{M}\sum_{m'=1}^{M} Q(g_t^{m'})\right)$, where initial momentum $m_0 = 0$ and β_1 is a hyperparameter taking value in $(0,1)$. $\phi(\cdot)$ is a rescaling function. As with (You et al., 2019), we assume $\phi(z) = \min\{\max\{z, \alpha_l\}, \alpha_u\}$, where α_l and α_u are constant values. We have the following theorem regarding the convergence rate of JOINTSPAR-LARS[6].

Theorem 5. *Under Assumption 1, let* $\alpha_x = \sqrt{\frac{2(f(x_1)-f(x^*))}{\alpha_u^2\|S\|_1 T}}$, *batch size* $B = T$. *Then for* x_t *generated using* JOINTSPAR-LARS, *we have the following bound*

$$\left(\mathbb{E}\left[\frac{1}{\sqrt{D}}\sum_{d=1}^{D}\|f_d'(x_a)\|\right]\right)^2 \leq O\left(\frac{(f(x_1)-f(x^*))\|S\|_1}{TD} + \frac{L^2D^2}{MsT}\right), \tag{8}$$

where $x^* = argmin_{x\in\mathcal{X}} f(x)$ *and* x_a *is an iterate uniformly randomly chosen from* $\{x_1, \cdots, x_T\}$.

This theorem implies that the convergence rate of JOINTSPAR-LARS is $O(\frac{1}{T})$ in term of *the number of iterations*, which is the same as the original LARS (see Theorem 2 in (You et al., 2019)). We empirically show that JOINTSPAR-LARS has a faster convergence rate than existing methods in terms of *wall-clock time*.

3.3. Implementation Details

We discuss how our methods are efficiently implemented on top of PyTorch. We treat each parameter tensor from the PyTorch framework as a parameter block in our implementation. This could also help us achieve highly efficient execution on GPU. Specifically, to skip gradient computation for a parameter block, we use the PyTorch API: `param_tensor.requires_grad_(False)` to turn off the gradient computation for its corresponding tensor `param_tensor`. For example, the `param_tensor` could be the kernel matrix of some convolutional layer, the bias vector at some linear layer and so on. This eliminates the need to break the matrix computation on GPUs while we still benefit from skipping some gradient computations. Note that the gradient is turned off (i.e., `requires_grad_(False)`) before *both* the forward pass as well as the backward pass. Because of the way auto-differentiation works in PyTorch (Paszke et al., 2017), backpropagation will stop at some intermediate layer if the gradients for all previous layers have been turned off. This has often been the case across all our experiments, which certainly does save a lot more than half of the computation during backpropagation. However, turning off the gradient compuation on the whole weight matrix on linear layers could greatly slow down the training process. To gain finer-grained control over gradient computation for linear layers, we implemented a new `nn.Linear` layer that splits each column of its original weight matrix into an independent submatrix. In this way, the gradient computation of each column can be independently turned off.

[6]As with (You et al., 2019), we analyze the case where $\beta_1 = 0$.

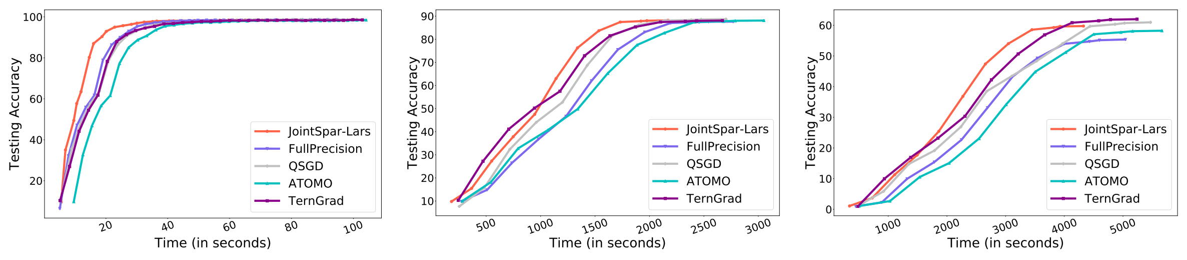
Modern deep learning frameworks, such as PyTorch, support overlapping communication and computation by sending gradients as soon as they are available (Li et al., 2020). This can accelerate training by avoiding the additional delay of waiting for all the gradients. To take advantage of the overlapping characteristic, many gradient compression methods are implemented in the layer-wise way, i.e., the compression is independently applied to the gradient vector of each layer (Dutta et al., 2020). Although our compression methods are not layer-wise, they can still enjoy the acceleration benefits provided by the overlapping characteristic. Simply speaking, this is because our methods still allow gradient vector of each layer to be sent as soon as it is computed. More specifically, at each iteration, our methods decide which layers should not send their gradient vectors even before the forward pass starts. We instantiate this decision in PyTorch via setting `param_tensor.requires_grad_(False)`. Then the procedure of the forward pass and backward pass starts with no involvement from our methods. In other words, each gradient vector is sent as soon as it is ready if overlapping communication and computation is supported by the framework.

4. Experiments

We conduct extensive experiments to evaluate the effectiveness and efficiency of our method. Due to space limit, we focus on JOINTSPAR-LARS because optimizers with layer-wise adaptive learning rates are more effective in the large batch setting (You et al., 2017; 2019). More experiments on JOINTSPAR and other SGD-based compression methods can be found in the appendix.

4.1. Setup

Baselines. We compare our method, JOINTSPAR-LARS against the full precision version of LARS (no compression used) and the following state-of-the-art gradient compression methods for distributed learning: QSGD (Alistarh et al., 2017), TERNGRAD (Wen et al., 2017), and ATOMO (Wang et al., 2018a). All the gradient compression baseline methods are applied to LARS in our experiments. Note that all of these baseline compression methods must compute all coordinates in a gradient vector before compressing it to reduce communication cost. To the best of our knowledge, our method JOINTSPAR-LARS is the first to jointly sparsify gradient computation and communication.



(a) MNIST (b) CIFAR10 (c) CIFAR100

Figure 1. Convergence in terms of wall-clock time, confirming our method JOINTSPAR-LARS's faster convergence than baselines.

Datasets and Implementations. We use several benchmark datasets in our experiments: MNIST, Fashion-MNIST, SVHN, CIFAR10, CIFAR100 and ImageNet. Different neural network models are trained on these datasets. We use PyTorch (Paszke et al., 2019) to implement models and learning methods, and use mpi4py (Dalcin et al., 2011) as the communication framework in the distributed setting. In traditional distribution learning, the global batch size is usually fixed and evenly divided among multiple machines (Wang et al., 2018a; Alistarh et al., 2017). In our experiments, however, we fix the local batch size at each machine, which is the common practice in large batch optimization (Goyal et al., 2017). All experiments are run on a computer cluster with up to 16 nodes. Each node has 20 physical CPU cores with clock speed up to 4 GHz, and 4 NVIDIA P100 GPUs. Nodes are connected via a 100Gb/s InfiniBand fabric.

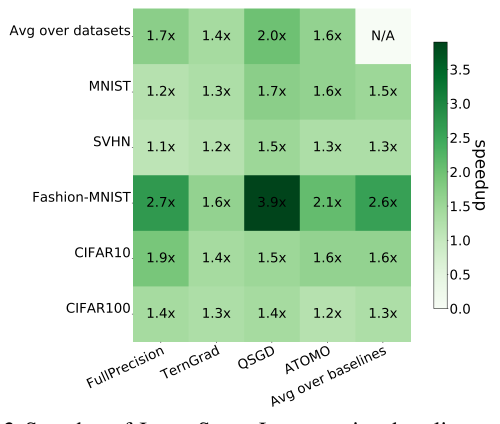


Figure 2. Speedup of JOINTSPAR-LARS against baseline methods, computed as the ratio of the wall-clock time spent by each baseline to that by JOINTSPAR when achieving the same testing loss.

4.2. Convergence Comparison

We compare the end-to-end convergence performance on different datasets and models trained with JOINTSPAR-LARS and all baselines, with respect to the number of epochs and wall-clock time. We use LeNet (Wen et al., 2017; LeCun et al., 1998) on MNIST, and ResNet-18 (He et al., 2016) on CIFAR10 and CIFAF100. We use 8 processes to collectively train each model and the local batch size is set to 1024. As with Wang et al. (2018a), for those methods that allow users to control the sparsity budget, we tune the sparsity budget to choose among $10\%, 30\%, 50\%, 80\%$ of the full gradient size. The convergence rates in terms of the wall-clock time are shown in Figure 1, where x-axis and y-axis are the wall-clock time and testing accuracy, respectively. In terms of wall-clock time, Figure 1 demonstrates that JOINTSPAR-LARS speeds up the model training significantly, thanks to spending less time per iteration than baselines.

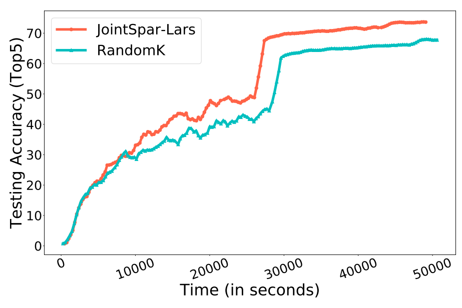


Figure 3. Convergence comparison for training AlexNet (with 62 million parameters) on ImageNet dataset. JOINTSPAR-LARS converges faster than baseline methods.

Speedup. To get a better understanding of our speedup against baselines, we compute the ratio of time spent by a baseline method vs. by our method when achieving the same target testing accuracy. We use the testing accuracy achieved by the full precision method at convergence as the target accuracy, as shown in Table 3. We report this ratio as speedup of our method against the corresponding baseline method. We use two more datasets: Fashion-MNIST and SVHN, and train the same convolutional neural network model as on MNIST. Figure 2 gives a detailed account of speedups of our method against different baseline methods on different datasets. Our method can achieve close to 2x speedup in most cases, and as high as 4x. Note that in

Table 3. Target accuracy when measuring the training time for speedup comparison

Dataset	MNIST	SVHN	Fashion-MNIST	CIFAR10	CIFAR100
Target Accuracy	90%	95%	91%	88%	55%

some cases such as FullPrecision on SVHN, the speedup is as low as 1.1x. To further reduce the training time, our method could be combined with skipping the forward pass for the layers for which the gradient will be dropped during the communication. The idea of skipping certain layers of neural network models has been explored in prior papers, which can speed up model training without compromising the accuracy (Huang et al., 2016; Fan et al., 2019). We leave this idea to the future work.

4.3. Convergence Comparison on ImageNet

To verify the effectiveness of our method on large models, we also train AlexNet (Krizhevsky et al., 2012) on the ImagetNet dataset (Deng et al., 2009). As demonstrated in (Goyal et al., 2017), using large batch size (up to 8k) is very effective at reducing the overall training time on the ImageNet dataset without loss of accuracy. We set the local batch size to 1024 for each machine, and uses the same tricks (i.e., linear scaling and warmup) as suggested in (Goyal et al., 2017). Other experimental settings are kept the same as in the previous subsection. From Figure 3, we observe that JOINTSPAR-LARS converges faster than the existing compression method RandomK, which randomly remove $K\%$ of the gradient coordinates. We still tune $K\%$ among $10\%, 30\%, 50\%, 80\%$ of the full gradient size. It demonstrates that our method maintains its benefits for large models.

5. Conclusion

We have observed that, in large batch optimization (e.g., large batch SGD and its variant LARS with layerwise adaptive learning rates), the per-machine computation cost is no longer negligible compared to the communication cost. We proposed new gradient compression methods for large batch optimization, JOINTSPAR (corresponding to large batch SGD) and its variant JOINTSPAR-LARS (corresponding to LARS), that only compute gradient coordinates that will be communicated. We formally prove that JOINTSPAR and JOINTSPAR-LARS still enjoy the same convergence rates (in terms of number of iterations) as their respective baseline methods. Extensive experiments demonstrate that our methods converge faster than existing methods.

Acknowledgements

Special thanks go to the ConFlux team at the University of Michigan for providing support to our experiments. The authors would also like to thank anonymous reviewers for their insightful feedback and suggestions.

References

Aji, A. F. and Heafield, K. Sparse communication for distributed gradient descent. *arXiv preprint arXiv:1704.05021*, 2017.

Akiba, T., Suzuki, S., and Fukuda, K. Extremely large minibatch sgd: Training resnet-50 on imagenet in 15 minutes. *arXiv preprint arXiv:1711.04325*, 2017.

Alain, G., Lamb, A., Sankar, C., Courville, A., and Bengio, Y. Variance reduction in sgd by distributed importance sampling. *arXiv preprint arXiv:1511.06481*, 2015.

Alistarh, D., Grubic, D., Li, J., Tomioka, R., and Vojnovic, M. Qsgd: Communication-efficient sgd via gradient quantization and encoding. In *Advances in Neural Information Processing Systems*, pp. 1709–1720, 2017.

Alistarh, D., Hoefler, T., Johansson, M., Konstantinov, N., Khirirat, S., and Renggli, C. The convergence of sparsified gradient methods. In *Advances in Neural Information Processing Systems*, pp. 5973–5983, 2018.

Auer, P., Cesa-Bianchi, N., and Fischer, P. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

Beck, A. and Teboulle, M. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 31(3):167–175, 2003.

Bernstein, J., Wang, Y.-X., Azizzadenesheli, K., and Anandkumar, A. signsgd: Compressed optimisation for nonconvex problems. *arXiv preprint arXiv:1802.04434*, 2018.

Bubeck, S., Cesa-Bianchi, N., et al. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends® in Machine Learning*, 5 (1):1–122, 2012.

Chen, C.-Y., Choi, J., Brand, D., Agrawal, A., Zhang, W., and Gopalakrishnan, K. Adacomp: Adaptive residual gradient compression for data-parallel distributed training. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018a.

Chen, J., Pan, X., Monga, R., Bengio, S., and Jozefowicz, R. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, 2016.

Chen, T., Giannakis, G., Sun, T., and Yin, W. Lag: Lazily aggregated gradient for communication-efficient distributed learning. *Advances in Neural Information Processing Systems*, 31, 2018b.

Chilimbi, T., Suzue, Y., Apacible, J., and Kalyanaraman, K. Project adam: Building an efficient and scalable deep learning training system. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pp. 571–582, 2014.

Clark, K., Luong, M.-T., Le, Q. V., and Manning, C. D. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020.

Dalcin, L. D., Paz, R. R., Kler, P. A., and Cosimo, A. Parallel distributed computing using python. *Advances in Water Resources*, 34(9):1124–1139, 2011.

Dani, V., Kakade, S. M., and Hayes, T. P. The price of bandit information for online optimization. In *Advances in Neural Information Processing Systems*, pp. 345–352, 2008.

Darzentas, J. Problem complexity and method efficiency in optimization. *Journal of the Operational Research Society*, 35(5):455–455, 1984.

De Sa, C., Feldman, M., Ré, C., and Olukotun, K. Understanding and optimizing asynchronous low-precision stochastic gradient descent. In *ACM SIGARCH Computer Architecture News*, volume 45, pp. 561–574. ACM, 2017.

De Sa, C., Leszczynski, M., Zhang, J., Marzoev, A., Aberger, C. R., Olukotun, K., and Ré, C. High-accuracy low-precision training. *arXiv preprint arXiv:1803.03383*, 2018.

De Sa, C. M., Zhang, C., Olukotun, K., and Ré, C. Taming the wild: A unified analysis of hogwild-style algorithms. In *Advances in neural information processing systems*, pp. 2674–2682, 2015.

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.

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

Dutta, A., Bergou, E. H., Abdelmoniem, A. M., Ho, C.-Y., Sahu, A. N., Canini, M., and Kalnis, P. On the discrepancy between the theoretical analysis and practical implementations of compressed communication for distributed deep learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 3817–3824, 2020.

Fan, A., Grave, E., and Joulin, A. Reducing transformer depth on demand with structured dropout. *arXiv preprint arXiv:1909.11556*, 2019.

Fei, J., Ho, C.-Y., Sahu, A. N., Canini, M., and Sapio, A. Efficient sparse collective communication and its application to accelerate distributed deep learning. Technical report, 2020.

Golmant, N., Vemuri, N., Yao, Z., Feinberg, V., Gholami, A., Rothauge, K., Mahoney, M. W., and Gonzalez, J. On the computational inefficiency of large batch sizes for stochastic gradient descent. *arXiv preprint arXiv:1811.12941*, 2018.

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.

Gupta, V., Choudhary, D., Tang, P., Wei, X., Wang, X., Huang, Y., Kejariwal, A., Ramchandran, K., and Mahoney, M. W. Training recommender systems at scale: Communication-efficient model and data parallelism. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 2928–2936, 2021.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. Q. Deep networks with stochastic depth. In *European conference on computer vision*, pp. 646–661. Springer, 2016.

Johnson, R. and Zhang, T. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, pp. 315–323, 2013.

Katharopoulos, A. and Fleuret, F. Not all samples are created equal: Deep learning with importance sampling. In *International conference on machine learning*, pp. 2525–2534. PMLR, 2018.

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. *arXiv preprint arXiv:1609.04836*, 2016.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Koloskova, A., Lin, T., Stich, S. U., and Jaggi, M. Decentralized deep learning with arbitrary communication compression. *arXiv preprint arXiv:1907.09356*, 2019.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Leblond, R., Pedregosa, F., and Lacoste-Julien, S. Asaga: asynchronous parallel saga. *arXiv preprint arXiv:1606.04809*, 2016.

Leblond, R., Pedregosa, F., and Lacoste-Julien, S. Improved asynchronous parallel optimization analysis for stochastic incremental methods. *The Journal of Machine Learning Research*, 19(1):3140–3207, 2018.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., Long, J., Shekita, E. J., and Su, B.-Y. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pp. 583–598, 2014.

Li, S., Zhao, Y., Varma, R., Salpekar, O., Noordhuis, P., Li, T., Paszke, A., Smith, J., Vaughan, B., Damania, P., et al. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704*, 2020.

Lian, X., Huang, Y., Li, Y., and Liu, J. Asynchronous parallel stochastic gradient for nonconvex optimization. *Advances in Neural Information Processing Systems*, 28, 2015.

Lin, T., Kong, L., Stich, S., and Jaggi, M. Extrapolation for large-batch training in deep learning. In *International Conference on Machine Learning*, pp. 6094–6104. PMLR, 2020.

Lin, Y., Han, S., Mao, H., Wang, Y., and Dally, W. J. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*, 2017.

Liu, J., Wright, S. J., Ré, C., Bittorf, V., and Sridhar, S. An asynchronous parallel stochastic coordinate descent algorithm. *The Journal of Machine Learning Research*, 16(1):285–322, 2015.

Liu, R. and Mozafari, B. Transformer with memory replay. *arXiv preprint arXiv:2205.09869*, 2022.

Liu, R., Wu, T., and Mozafari, B. Adam with bandit sampling for deep learning. *Advances in Neural Information Processing Systems*, 33:5393–5404, 2020.

Liu, R., Kim, Y. J., Muzio, A., and Awadalla, H. H. Gating dropout: Communication-efficient regularization for sparsely activated transformers. *arXiv preprint arXiv:2205.14336*, 2022.

Liu, Y., Xu, W., Wu, G., Tian, Z., and Ling, Q. Communication-censored admm for decentralized consensus optimization. *IEEE Transactions on Signal Processing*, 67(10):2565–2579, 2019.

Mania, H., Pan, X., Papailiopoulos, D., Recht, B., Ramchandran, K., and Jordan, M. I. Perturbed iterate analysis for asynchronous stochastic optimization. *arXiv preprint arXiv:1507.06970*, 2015.

McMahan, H. B. and Blum, A. Online geometric optimization in the bandit setting against an adaptive adversary. In *International Conference on Computational Learning Theory*, pp. 109–123. Springer, 2004.

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

Namkoong, H., Sinha, A., Yadlowsky, S., and Duchi, J. C. Adaptive sampling probabilities for non-smooth optimization. In *International Conference on Machine Learning*, pp. 2574–2583, 2017.

Needell, D., Ward, R., and Srebro, N. Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm. *Advances in Neural Information Processing Systems*, 27:1017–1025, 2014.

Nguyen, L. M., Nguyen, P. H., van Dijk, M., Richtárik, P., Scheinberg, K., and Takáč, M. Sgd and hogwild! convergence without the bounded gradients assumption. *arXiv preprint arXiv:1802.03801*, 2018.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. 2017.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pp. 8024–8035, 2019.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pp. 525–542. Springer, 2016.

Recht, B., Re, C., Wright, S., and Niu, F. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pp. 693–701, 2011.

Renggli, C., Alistarh, D., Hoefler, T., and Aghagolzadeh, M. Sparcml: High-performance sparse communication for machine learning. *arXiv preprint arXiv:1802.08021*, 2018.

Renggli, C., Ashkboos, S., Aghagolzadeh, M., Alistarh, D., and Hoefler, T. Sparcml: High-performance sparse communication for machine learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15, 2019.

Sakr, C., Wang, N., Chen, C.-Y., Choi, J., Agrawal, A., Shanbhag, N., and Gopalakrishnan, K. Accumulation bit-width scaling for ultra-low precision training of deep networks. *arXiv preprint arXiv:1901.06588*, 2019.

Schmidt, M., Le Roux, N., and Bach, F. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112, 2017.

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.

Shalev-Shwartz, S. and Zhang, T. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14(Feb):567–599, 2013.

Shazeer, N., Cheng, Y., Parmar, N., Tran, D., Vaswani, A., Koanantakool, P., Hawkins, P., Lee, H., Hong, M., Young, C., et al. Mesh-tensorflow: Deep learning for supercomputers. *Advances in neural information processing systems*, 31, 2018.

Smith, S. L., Kindermans, P.-J., Ying, C., and Le, Q. V. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

Stich, S. U., Cordonnier, J.-B., and Jaggi, M. Sparsified sgd with memory. In *Advances in Neural Information Processing Systems*, pp. 4447–4458, 2018.

Strom, N. Scalable distributed dnn training using commodity gpu cloud computing. In *Sixteenth Annual Conference of the Inaernational Speech Communication Association*, 2015.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pp. 1139–1147, 2013.

Tang, H., Gan, S., Zhang, C., Zhang, T., and Liu, J. Communication compression for decentralized training. In *Advances in Neural Information Processing Systems*, pp. 7652–7662, 2018.

Tang, H., Yu, C., Lian, X., Zhang, T., and Liu, J. Doublesqueeze: Parallel stochastic gradient descent with double-pass error-compensated compression. In *International Conference on Machine Learning*, pp. 6155–6165. PMLR, 2019.

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*, pp. 10118–10129. PMLR, 2021.

Tsuzuku, Y., Imachi, H., and Akiba, T. Variance-based gradient compression for efficient distributed deep learning. *arXiv preprint arXiv:1802.06058*, 2018.

Vogels, T., Karimireddy, S. P., and Jaggi, M. Powersgd: Practical low-rank gradient compression for distributed optimization. *arXiv preprint arXiv:1905.13727*, 2019.

Vogels, T., Karimireddy, S. P., and Jaggi, M. Practical low-rank communication compression in decentralized deep learning. *Advances in Neural Information Processing Systems*, 33, 2020.

Wang, H., Sievert, S., Liu, S., Charles, Z., Papailiopoulos, D., and Wright, S. Atomo: Communication-efficient learning via atomic sparsification. In *Advances in Neural Information Processing Systems*, pp. 9850–9861, 2018a.

Wang, N., Choi, J., Brand, D., Chen, C.-Y., and Gopalakrishnan, K. Training deep neural networks with 8-bit floating point numbers. In *Advances in neural information processing systems*, pp. 7675–7684, 2018b.

Wangni, J., Wang, a., Liu, J., and Zhang, T. Gradient sparsification for communication-efficient distributed optimization. In *Advances in Neural Information Processing Systems*, pp. 1299–1309, 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. In *Advances in neural information processing systems*, pp. 1509–1519, 2017.

Wu, J., Huang, W., Huang, J., and Zhang, T. Error compensated quantized sgd and its applications to large-scale distributed optimization. In *International Conference on Machine Learning*, pp. 5325–5333. PMLR, 2018.

Xu, H., Ho, C.-Y., Abdelmoniem, A. M., Dutta, A., Bergou, E. H., Karatsenidis, K., Canini, M., and Kalnis, P. Compressed communication for distributed deep learning: Survey and quantitative evaluation. Technical report, 2020.

You, Y., Gitman, I., and Ginsburg, B. Scaling sgd batch size to 32k for imagenet training. *arXiv preprint arXiv:1708.03888*, 6:12, 2017.

You, Y., Li, J., Reddi, S., Hseu, J., Kumar, S., Bhojanapalli, S., Song, X., Demmel, J., Keutzer, K., and Hsieh, C.-J. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962*, 2019.

Yu, M., Lin, Z., Narra, K., Li, S., Li, Y., Kim, N. S., Schwing, A., Annavaram, M., and Avestimehr, S. Gradiveq: Vector quantization for bandwidth-efficient gradient aggregation in distributed cnn training. *arXiv preprint arXiv:1811.03617*, 2018.

Zhang, H., Li, J., Kara, K., Alistarh, D., Liu, J., and Zhang, C. Zipml: Training linear models with end-to-end low precision, and a little bit of deep learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 4035–4043. JMLR. org, 2017.

Zhao, P. and Zhang, T. Stochastic optimization with importance sampling for regularized loss minimization. In *international conference on machine learning*, pp. 1–9. PMLR, 2015.

Zheng, S., Huang, Z., and Kwok, J. T. Communication-efficient distributed blockwise momentum sgd with error-feedback. *arXiv preprint arXiv:1905.10936*, 2019.

Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., and Zou, Y. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

A. Proof of Theorem 1

For ease of presentation, we define $G_{t,d} = \sum_{m=1}^{M} g_{t,d}^m$, and $G_{t,[d]} = \sum_{m=1}^{M} g_{t,[d]}^m$. We also define $G_t = \sum_{d=1}^{D} G_{t,[d]}$. According to the definition of S_d-smoothness of f with respect to x_d and the update rule $x_{t+1,d} = x_{t,d} - \alpha_x G_{t,d}$, we have

$$\begin{aligned} f(x_{t+1}) - f(x_t) &\leq \sum_{d=1}^{D} \langle f'_d(x_t), x_{t+1,d} - x_{t,d}\rangle + \sum_{d=1}^{D} \frac{S_d}{2} \|x_{t+1,d} - x_{t,d}\|^2 \\ &\leq \sum_{d=1}^{D} \langle f'_d(x_t), x_{t+1,d} - x_{t,d}\rangle + \frac{S_\infty}{2} \|x_{t+1} - x_t\|^2 \\ &= -\alpha_x \sum_{d=1}^{D} \langle f'_d(x_t), G_{t,d}\rangle + \frac{S_\infty \alpha_x^2}{2} \|G_t\|^2. \end{aligned} \tag{9}$$

Thus, taking expectation on both sides and using the unbiasedness of the gradient, i.e., $\mathbb{E}G_{t,d} = f'_d(x_t)$, we have

$$\mathbb{E}\left[f(x_{t+1})\right] - \mathbb{E}\left[f(x_t)\right] \leq -\alpha_x \sum_{d=1}^{D} \|f'_d(x_t)\|^2 + \frac{S_\infty \alpha_x^2}{2} \mathbb{E}\|G_t\|^2. \tag{10}$$

Rearranging the order, we have

$$\sum_{d=1}^{D} \|f'_d(x_t)\|^2 \leq \frac{\mathbb{E}\left[f(x_t)\right] - \mathbb{E}\left[f(x_{t+1})\right]}{\alpha_x} + \frac{S_\infty \alpha_x}{2} \mathbb{E}\|G_t\|^2. \tag{11}$$

Then,

$$\begin{aligned} &\sum_{t=1}^{T} \sum_{d=1}^{D} \|f'(x_t)\|^2 \\ \leq &\frac{\mathbb{E}\left[f(x_1)\right] - \mathbb{E}\left[f(x_{T+1})\right]}{\alpha_x} + \frac{S_\infty \alpha_x}{2} \sum_{t=1}^{T} \mathbb{E}\|G_t\|^2 \\ \leq &\frac{\mathbb{E}\left[f(x_1)\right] - \mathbb{E}\left[f(x_{T+1})\right]}{\alpha_x} + \frac{S_\infty \alpha_x}{2M^2} \sum_{m=1}^{M} \sum_{t=1}^{T} \sum_{d=1}^{D} \frac{\|g_{t,[d]}^m\|^2}{p_{t,d}^m} \\ \leq &\frac{R^2}{\alpha_x} + \frac{S_\infty \alpha_x}{2M^2} \sum_{m=1}^{M} \sum_{t=1}^{T} \sum_{d=1}^{D} \frac{\|g_{t,[d]}^m\|^2}{p_{t,d}^m}. \end{aligned} \tag{12}$$

B. Proof of Theorem 2

As for Algorithm 1, it corresponds to mirror descent update with $\psi(p) = \sum_{d=1}^{D} p_d \log p_d$ and $\psi^*(u) = \sum_{d=1}^{D} \exp(u_d - 1)$. Denote $\hat{h}_{t,d}^m = \frac{\tilde{l}_{t,d}^m Z_{t,d}^m}{p_{t,d}^m}$. Based on Theorem 5.3 from (Bubeck et al., 2012), we have

$$\begin{aligned} &\bar{R}_T^m \\ =&\mathbb{E} \sum_{t=1}^{T} L_t^m(p_t^m) - \min_p \mathbb{E} \sum_{t=1}^{T} L_t^m(p) \\ \leq &\frac{B_\psi(p_*^m, p_1^m)}{\alpha_p} + \frac{1}{\alpha_p} \sum_{t=1}^{T} \mathbb{E}\left[B_{\psi^*}(\nabla\psi(p_t^m) - \alpha_p \hat{h}_{t,d}^m, \nabla\psi(p_t^m))\right]. \end{aligned} \tag{13}$$

Furthermore, we have

$$\begin{aligned}
&\bar{R}_T^m \\
=&\mathbb{E}\sum_{t=1}^{T} L_t^m(p_t^m) - \underset{p}{min}\,\mathbb{E}\sum_{t=1}^{T} L_t^m(p) \\
\leq&\frac{B_\psi(p_*^m, p_1^m)}{\alpha_p} + \frac{1}{\alpha_p}\sum_{t=1}^{T}\mathbb{E}\left[B_{\psi^*}(\nabla\psi(p_t^m) - \alpha_p \hat{h}_{t,d}^m, \nabla\psi(p_t^m))\right] \\
=&\frac{B_\psi(p_*^m, p_1^m)}{\alpha_p} + \frac{1}{\alpha_p}\sum_{t=1}^{T}\mathbb{E}\left[\sum_{d=1}^{D} p_{t,d}^m(\exp(-\alpha_p \hat{h}_{t,d}^m) + \alpha_p \hat{h}_{t,d}^m - 1)\right] \\
&\text{(due to inequality } e^z - z - 1 \leq z^2 \text{ for } z \leq 0) \\
\leq&\frac{B_\psi(p_*^m, p_1^m)}{\alpha_p} + \frac{\alpha_p}{2}\sum_{t=1}^{T}\mathbb{E}\left[\sum_{d=1}^{D} p_{t,d}^m(\hat{h}_{t,d}^m)^2\right] \\
=&\frac{B_\psi(p_*^m, p_1^m)}{\alpha_p} + \frac{\alpha_p}{2}\sum_{t=1}^{T}\mathbb{E}\left[\sum_{d=1}^{D} p_{t,d}^m \frac{(\tilde{l}_{t,d}^m)^2 (Z_{t,d}^m)^2}{(p_{t,d}^m)^2}\right] \\
=&\frac{B_\psi(p_*^m, p_1^m)}{\alpha_p} + \frac{\alpha_p}{2}\sum_{t=1}^{T}\sum_{d=1}^{D} p_{t,d}^m \frac{(\tilde{l}_{t,d}^m)^2 \mathbb{E}\left[(Z_{t,d}^m)^2\right]}{(p_{t,d}^m)^2} \\
=&\frac{B_\psi(p_*^m, p_1^m)}{\alpha_p} + \frac{\alpha_p}{2}\sum_{t=1}^{T}\sum_{d=1}^{D} p_{t,d}^m \frac{(\tilde{l}_{t,d}^m)^2 p_{t,d}^m}{(p_{t,d}^m)^2} \\
=&\frac{B_\psi(p_*^m, p_1^m)}{\alpha_p} + \frac{\alpha_p}{2}\sum_{t=1}^{T}\sum_{d=1}^{D} (\tilde{l}_{t,d}^m)^2.
\end{aligned} \tag{14}$$

Since $\tilde{l}_{t,d}^m = -\frac{\|G_{t,[d]}^m\|^2}{(p_{t,d}^m)^2} + \frac{L^2}{p_{min}^2}$, we have $\tilde{l}_{t,d}^m \leq \frac{L^2}{p_{min}^2}$. Similar to Assumption 1, we also assume that $B_\psi(p_*^m, p_1^m) \leq R^2$. Thus, with choosing $\alpha_p = \sqrt{\frac{2R^2 p_{min}^4}{DTL^4}}$, we could get

$$\begin{aligned}
\bar{R}_T^m \leq&\frac{R^2}{\alpha_p} + \frac{\alpha_p}{2}\sum_{t=1}^{T}\sum_{d=1}^{D}\frac{L^4}{p_{min}^4} \\
=&\frac{R^2}{\alpha_p} + \frac{\alpha_p}{2}\frac{TDL^4}{p_{min}^4} \\
=&\frac{RL^2}{p_{min}^2}\sqrt{2DT}.
\end{aligned} \tag{15}$$

C. Proof of Theorem 3

Let $\bar{R}^* = max_m\, min_p\, \mathbb{E}\left[\sum_{t=1}^{T} L_t^m(p)\right]$. Assume $\sum_{d=1}^{D} p_{t,d}^m = s, \forall m, t$, where s is the sparsity rate. Plugging Theorem 2 into Theorem 1, and choosing $\alpha_x = \frac{R}{DL}\sqrt{\frac{2Ms}{S_\infty T}}$, we have

$$\begin{aligned}
&\sum_{t=1}^{T}\sum_{d=1}^{D}\|f'(x_t)\|^2 \\
\leq&\frac{R^2}{\alpha_x} + \frac{S_\infty \alpha_x}{2M}\left(\frac{RL^2}{p_{min}^2}\sqrt{2DT} + \bar{R}^*\right) \\
=&RDL\sqrt{\frac{S_\infty T}{2Ms}} + \frac{R}{LD}\sqrt{\frac{S_\infty s}{2MT}}\bar{R}^* + \frac{R^2 L}{p_{min}^2}\sqrt{\frac{S_\infty s}{MD}}.
\end{aligned} \tag{16}$$

D. Proof of Lemma 1

Let us now consider each iteration at each machine. First, let us prove the following inequality

$$\min_{p\in\mathcal{P}} \sum_{j=1}^{D} \frac{L^2}{p_j} \leq O(\frac{L^2 D^2}{s}). \tag{17}$$

To prove the above inequality, the Lagrangian of the left hand side is

$$\mathcal{L}(p,\eta,\theta) = \sum_{d=1}^{D} \frac{L^2}{p_d} - \eta(p^T \mathbb{1} - s) - \theta^T (p - p_{min}\mathbb{1}). \tag{18}$$

Setting its first order derivative with respect to p_j to 0, then we have

$$p_j = \frac{L}{\sqrt{|\eta + \theta_j|}}. \tag{19}$$

According to complementary slackness, we have $\theta_j(p_j - p_{min}) = 0$. Bcause p_{min} is small, it implies that

$$p_j = \frac{L}{\sqrt{|\eta|}}. \tag{20}$$

Plugging p_j's into the equality constraint, we get

$$\sum_{j=1}^{D} p_j = \frac{DL}{\sqrt{|\eta|}} = s. \tag{21}$$

It implies that

$$\sqrt{|\eta|} = \frac{DL}{s}. \tag{22}$$

Therefore, we have

$$\min_{p\in\mathcal{P}} \sum_{j=1}^{D} \frac{L^2}{p_j} = O(\frac{L^2 D^2}{s}). \tag{23}$$

Plugging Equation 23 into $\bar{R}^*$, we have

$$\bar{R}^* = \max_{m} \min_{p\in\mathcal{P}} \mathbb{E}\left[\sum_{t=1}^{T}\sum_{d=1}^{D} \frac{\|g^m_{t,[d]}\|^2}{p_d}\right] \leq O(\frac{TL^2D^2}{s}). \tag{24}$$

E. Proof of Theorem 4

Plugging Lemma 1 into the Theorem 4, we have

$$\begin{aligned}
&\sum_{t=1}^{T}\sum_{d=1}^{D} \|f'_d(x_t)\|^2 \\
\leq & RDL\sqrt{\frac{S_\infty T}{2Ms}} + \frac{R}{LD}\sqrt{\frac{S_\infty s}{2MT}}\bar{R}^* + \frac{R^2 L}{p_{min}^2}\sqrt{\frac{S_\infty s}{MD}} \\
\leq & RDL\sqrt{\frac{S_\infty T}{2Ms}} + \frac{R}{LD}\sqrt{\frac{S_\infty s}{2MT}}\frac{TL^2D^2}{s} + \frac{R^2 L}{p_{min}^2}\sqrt{\frac{S_\infty s}{MD}} \\
= & RDL\sqrt{\frac{S_\infty T}{2Ms}} + RDL\sqrt{\frac{S_\infty T}{2Ms}} + \frac{R^2 L}{p_{min}^2}\sqrt{\frac{S_\infty s}{MD}} \\
\leq & O\left(RLD\sqrt{\frac{2S_\infty T}{Ms}}\right).
\end{aligned} \tag{25}$$

Therefore, we have $\frac{1}{T}\sum_{d=1}^{D}\sum_{t=1}^{T}\|f_d'(x_t)\|^2 \leq O\left(RLD\sqrt{\frac{2S_\infty}{MsT}}\right)$. It implies that the average convergence rate is still $O(\frac{1}{\sqrt{T}})$.

F. Analysis for Convex Problems for JOINTSPAR

We discuss convergence guarantee for convex problems in this section, which gives us a tighter convergence bound for JOINTSPAR. We make the following typical assumptions for convex problems (Beck & Teboulle, 2003; Darzentas, 1984):

Assumption 2. (1) For all $x, y \in \mathcal{X}$, we have $B(x,y) \leq R^2$, where $B(x,y)$ is the Bregman divergence between x and y; (2) $L \geq \|g_{t,[d]}^m\|, \forall t, d, m$.

Under Assumption 2, ORACLESPAR achieves

$$\begin{aligned}&\mathbb{E}\left[\sum_{t=1}^{T} f(x_t) - f(x^*)\right]\\ \leq&\frac{R^2}{\alpha_x} + \frac{\alpha_x}{2M^2}\sum_{m=1}^{M}\sum_{t=1}^{T}\sum_{d=1}^{D}\frac{\|G_{t,[d]}^m\|^2}{p_{t,d}^m}.\end{aligned} \tag{26}$$

Proof. This can be proven from the perspective of stochastic mirror descent (Darzentas, 1984; Beck & Teboulle, 2003), which is a generalization of stochastic gradient descent. The specific form of stochastic mirror descent depends on the choice of Bregman Divergence. Based on Proposition 1 from (Namkoong et al., 2017) or Theorem 4.1 from (Beck & Teboulle, 2003), we have

$$\begin{aligned}&\mathbb{E}\left[\sum_{t=1}^{T} f(x_t) - f(x^*)\right]\\ \leq&\frac{R^2}{\alpha_x} + \frac{\alpha_x}{2}\sum_{t=1}^{T}\mathbb{E}\left[\|G_t\|^2\right].\end{aligned} \tag{27}$$

According to the update rule, we know $G_t = \frac{1}{M}\sum_{m=1}^{M} Q(G_t^m)$. Therefore, we have

$$\begin{aligned}&\mathbb{E}\left[\sum_{t=1}^{T} f(x_t) - f(x^*)\right]\\ =&\frac{R^2}{\alpha_x} + \frac{\alpha_x}{2}\sum_{t=1}^{T}\mathbb{E}\left[\|\frac{1}{M}\sum_{m=1}^{M} Q(G_t^m)\|^2\right]\\ \leq&\frac{R^2}{\alpha_x} + \frac{\alpha_x}{2M^2}\sum_{t=1}^{T}\sum_{m=1}^{M}\mathbb{E}\left[\|Q(G_t^m)\|^2\right]\\ =&\frac{R^2}{\alpha_x} + \frac{\alpha_x}{2M^2}\sum_{t=1}^{T}\sum_{m=1}^{M}\mathbb{E}\left[\left\|\sum_{d=1}^{D}\frac{G_{t,[d]}^m}{p_{t,d}^m}Z_{t,d}^m\right\|^2\right]\\ \leq&\frac{R^2}{\alpha_x} + \frac{\alpha_x}{2M^2}\sum_{t=1}^{T}\sum_{m=1}^{M}\sum_{d=1}^{D}\frac{\|G_{t,[d]}^m\|^2}{(p_{t,d}^m)^2}\mathbb{E}\left[(Z_{t,d}^m)^2\right]\\ =&\frac{R^2}{\alpha_x} + \frac{\alpha_x}{2M^2}\sum_{t=1}^{T}\sum_{m=1}^{M}\sum_{d=1}^{D}\frac{\|G_{t,[d]}^m\|^2}{p_{t,d}^m}\\ =&\frac{R^2}{\alpha_x} + \frac{\alpha_x}{2M^2}\sum_{m=1}^{M}\sum_{t=1}^{T}\sum_{d=1}^{D}\frac{\|G_{t,[d]}^m\|^2}{p_{t,d}^m}.\end{aligned} \tag{28}$$

□

Under Assumption 2, JOINTSPAR achieves

$$\begin{aligned}&\mathbb{E}\left[\sum_{t=1}^{T} f(x_t)-f(x^*)\right]\\ \le&RDL\sqrt{\frac{T}{2Ms}}+\frac{R}{LD}\sqrt{\frac{s}{2MT}}\bar{R}^*+\frac{R^2L}{p_{min}^2}\sqrt{\frac{s}{MD}}\end{aligned} \tag{29}$$

where

$$\bar{R}^*=\max_{m}\min_{p\in\mathcal{P}}\mathbb{E}\left[\sum_{t=1}^{T}\sum_{d=1}^{D}\frac{\|g_{t,[d]}^m\|^2}{p_d}\right]. \tag{30}$$

Proof. Plugging the original Theorem 2 into the above Theorem F, and choosing $\alpha_x=\frac{R}{DL}\sqrt{\frac{2Ms}{T}}$, we have

$$\begin{aligned}&\mathbb{E}\left[\sum_{t=1}^{T} f(x^t)-f(x^*)\right]\\ \le&\frac{R^2}{\alpha_x}+\frac{\alpha_x}{2M^2}\sum_{m=1}^{M}\mathbb{E}\left[\sum_{t=1}^{T}L_t^m(p_t^m)\right]\\ \le&\frac{R^2}{\alpha_x}+\frac{\alpha_x}{2M^2}\sum_{m=1}^{M}\left(\frac{RL^2}{p_{min}^2}\sqrt{2DT}+\bar{R}^*\right)\\ =&\frac{R^2}{\alpha_x}+\frac{\alpha_x}{2M}\left(\frac{RL^2}{p_{min}^2}\sqrt{2DT}+\bar{R}^*\right)\\ =&RLD\sqrt{\frac{T}{2Ms}}+\frac{R}{LD}\sqrt{\frac{s}{2MT}}\bar{R}^*+\frac{R^2L}{p_{min^2}}\sqrt{\frac{s}{MD}}.\end{aligned} \tag{31}$$

□

Under Assumption 2, our distributed learning method JOINTSPAR has the following convergence rate

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} f(x_t)-f(x^*)\right]\le O\left(RLD\sqrt{\frac{2S}{MsT}}\right) \tag{32}$$

where f is the non-convex and S-smooth objective function, and s is the sparsity budget.

Proof. Plugging the original Lemma 1 into the Theorem F, we have

$$\begin{aligned}&\mathbb{E}\left[\sum_{t=1}^{T} f(x_t)-f(x^*)\right]\\ =&RLD\sqrt{\frac{T}{2Ms}}+\frac{R}{LD}\sqrt{\frac{s}{2MT}}\bar{R}^*+\frac{R^2L}{p_{min^2}}\sqrt{\frac{s}{MD}}\\ \le&O\left(RLD\sqrt{\frac{T}{2Ms}}+\frac{R}{LD}\sqrt{\frac{s}{2MT}}T\frac{L^2D^2}{s}\right)\\ =&O\left(RLD\sqrt{\frac{2T}{Ms}}\right).\end{aligned} \tag{33}$$

Therefore, we have $\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} f(x_t)-f(x^*)\right]\le O\left(RLD\sqrt{\frac{2}{MsT}}\right)$. It implies that the average convergence rate is still $O(\frac{1}{\sqrt{T}})$. □

G. Proof of Theorem 5

Similar to the proof for Theorem 2 from (You et al., 2019), setting $\alpha_x = \sqrt{\frac{2(f(x_1)-f(x^*))}{\alpha_u^2\|S\|_1 T}}$, we could obtain

$$\begin{aligned}
&\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\frac{1}{\sqrt{D}}\sum_{d=1}^{D}\|f'_d(x_t)\|\right] \\
\leq&\frac{f(x_1)-f(x^*)}{T\alpha_l\alpha_x}+\frac{\alpha_x\alpha_u^2}{2\alpha_l}\|S\|_1+\frac{2\alpha_u}{\alpha_l}\frac{1}{\sqrt{D}T}\sum_{t=1}^{T}\sum_{d=1}^{D}\mathbb{E}\left[\|\Delta_{t,d}\|\right] \\
&(\text{due to } \alpha_x=\sqrt{\frac{2(f(x_1)-f(x^*))}{\alpha_u^2\|S\|_1 T}}) \\
=&\frac{\alpha_u}{\alpha_l}\sqrt{\frac{2(f(x_1)-f(x^*))\|S\|_1}{TD}}+\frac{2\alpha_u}{\alpha_l}\frac{1}{\sqrt{D}T}\sum_{t=1}^{T}\sum_{d=1}^{D}\mathbb{E}\left[\|\Delta_{t,d}\|\right]
\end{aligned} \tag{34}$$

where $\Delta_{t,d} = G_{t,d} - f'_d(x_t)$ and $G_{t,d} = \frac{1}{MB}\sum_{m=1}^{M}\sum_{b=1}^{B}\frac{g^{m,b}_{t,[d]}}{p^m_{t,d}}Z^m_{t,d}$. Note that b is the index for individual example in a batch, and B is the batch size.

Applying square to both sides, we have

$$\begin{aligned}
&\left(\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\frac{1}{\sqrt{D}}\sum_{d=1}^{D}\|f'_d(x_t)\|\right]\right)^2 \\
\leq&\frac{\alpha_u}{\alpha_l}\frac{2(f(x_1)-f(x^*))\|S\|_1}{TD}+\frac{4\alpha_u^2}{\alpha_l^2}\frac{1}{DT^2}\left(\sum_{t=1}^{T}\sum_{d=1}^{D}\mathbb{E}\|\Delta_{t,d}\|\right)^2 \\
\leq&\frac{\alpha_u}{\alpha_l}\frac{2(f(x_1)-f(x^*))\|S\|_1}{TD}+\frac{4\alpha_u^2}{\alpha_l^2}\frac{1}{T}\sum_{t=1}^{T}\sum_{d=1}^{D}(\mathbb{E}\|\Delta_{t,d}\|)^2 \\
&(\text{due to Jensen's inequality}) \\
\leq&\frac{\alpha_u}{\alpha_l}\frac{2(f(x_1)-f(x^*))\|S\|_1}{TD}+\frac{4\alpha_u^2}{\alpha_l^2}\frac{1}{T}\sum_{t=1}^{T}\sum_{d=1}^{D}\mathbb{E}\|\Delta_{t,d}\|^2
\end{aligned} \tag{35}$$

Next, we will obtain an upper bound on $\sum_{t=1}^{T}\sum_{d=1}^{D}\mathbb{E}\|\Delta_{t,d}\|^2$. Since $\mathbb{E}G_{t,d} = f'_d(x_t)$, we have

$$\begin{aligned}
\sum_{t=1}^{T}\sum_{d=1}^{D}\mathbb{E}\|\Delta_{t,d}\|^2 &= \sum_{t=1}^{T}\sum_{d=1}^{D}\frac{1}{M^2B^2}\sum_{m=1}^{M}\sum_{b=1}^{B}\mathbb{E}\|\frac{g^{m,b}_{t,[d]}}{p^m_{t,d}}Z^m_{t,d}-f'_d(x_t)\|^2 \\
&= \sum_{t=1}^{T}\sum_{d=1}^{D}\frac{1}{M^2B^2}\sum_{m=1}^{M}\sum_{b=1}^{B}\left(\mathbb{E}\|\frac{g^{m,b}_{t,[d]}}{p^m_{t,d}}Z^m_{t,d}\|^2-\|f'_d(x_t)\|^2\right) \\
&\leq \sum_{t=1}^{T}\sum_{d=1}^{D}\frac{1}{M^2B^2}\sum_{m=1}^{M}\sum_{b=1}^{B}\mathbb{E}\|\frac{g^{m,b}_{t,[d]}}{p^m_{t,d}}Z^m_{t,d}\|^2 \\
&= \frac{1}{M^2B^2}\sum_{m=1}^{M}\sum_{b=1}^{B}\sum_{t=1}^{T}\sum_{d=1}^{D}\frac{\|g^{m,b}_{t,[d]}\|^2}{p^m_{t,d}}
\end{aligned} \tag{36}$$

From Theorem 2 and Lemma 1, we could get

$$\begin{aligned}
&\sum_{t=1}^{T}\sum_{d=1}^{D}\frac{\|g^{m,b}_{t,[d]}\|^2}{p^m_{t,d}} \\
\leq&\frac{RL^2}{p^2_{min}}\sqrt{2DT}+\frac{TL^2D^2}{s}
\end{aligned} \tag{37}$$

Therefore, we have

$$\begin{aligned}
&\sum_{t=1}^{T}\sum_{d=1}^{D}\mathbb{E}\|\Delta_{t,d}\|^2 \\
\leq&\frac{1}{M^2B^2}\sum_{m=1}^{M}\sum_{b=1}^{B}\sum_{t=1}^{T}\sum_{d=1}^{D}\frac{\|g_{t,[d]}^{m,b}\|^2}{p_{t,d}^m} \\
\leq&\frac{1}{M^2B^2}\sum_{m=1}^{M}\sum_{b=1}^{B}\left(\frac{RL^2}{p_{min}^2}\sqrt{2DT}+\frac{TL^2D^2}{s}\right) \\
\leq&\frac{1}{MB}\left(\frac{RL^2}{p_{min}^2}\sqrt{2DT}+\frac{TL^2D^2}{s}\right) \\
&\text{(due to the large batch assumption that } B=T\text{)} \\
=&\frac{1}{MT}\left(\frac{RL^2}{p_{min}^2}\sqrt{2DT}+\frac{TL^2D^2}{s}\right) \\
=&\frac{1}{M}\left(\frac{RL^2}{p_{min}^2}\sqrt{\frac{2D}{T}}+\frac{L^2D^2}{s}\right)
\end{aligned} \tag{38}$$

Plugging the above bound to Inequality 35, we finally have

$$\begin{aligned}
&\left(\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\frac{1}{\sqrt{D}}\sum_{d=1}^{D}\|f_d'(x_t)\|\right]\right)^2 \\
\leq&\frac{\alpha_u}{\alpha_l}\frac{2(f(x_1)-f(x^*))\|S\|_1}{TD}+\frac{4\alpha_u^2}{\alpha_l^2}\frac{1}{T}\sum_{t=1}^{T}\sum_{d=1}^{D}\mathbb{E}\|\Delta_{t,d}\|^2 \\
\leq&\frac{\alpha_u}{\alpha_l}\frac{2(f(x_1)-f(x^*))\|S\|_1}{TD}+\frac{4\alpha_u^2}{\alpha_l^2}\frac{1}{T}\frac{1}{M}\left(\frac{RL^2}{p_{min}^2}\sqrt{\frac{2D}{T}}+\frac{L^2D^2}{s}\right) \\
=&O\left(\frac{(f(x_1)-f(x^*))\|S\|_1}{TD}+\frac{L^2D^2}{MsT}\right)
\end{aligned} \tag{39}$$

H. More Details on Experimental Setup

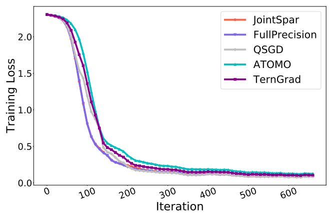
We train the model on each dataset for 90 epochs with the first 5 epochs as the warmup stage as suggested in (Goyal et al., 2017). For the learning rate schedule, we set the initial learning rate as 0.1, and shrink the learning rate by 0.1 at epoch $30, 50, 70$.

I. Experiments on SGD-based JOINTSPAR

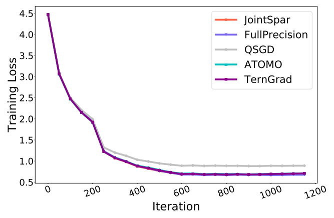
We include experimental results for JOINTSPAR which uses SGD as the base optimizer (its pseudocode in Algorithm 2). The convergence rates in terms of the number of iterations are shown in Figure 4, where x-axis is the number of iterations, and y-axis is the training loss. We can see that JOINTSPAR has the same convergece rate as with baselines, which verifies Theorem 4 in the main paper. The convergence rates in terms of the wall-clock time are shown in Figure 5, where x-axis is the wall-clock, and y-axis is the testing accuracy. We can see that JOINTSPAR in general converges faster than baselines.

J. Gradient Computation Time per Epoch

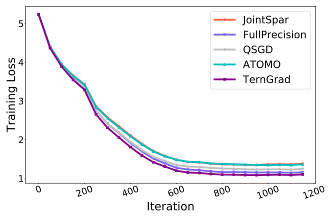
By joint sparsification of the gradient computation and communication, JOINTSPAR aims to reduce both the time spent on computing the gradient and the time spent on communicating it, whereas state-of-the-art communication-efficient methods focus only on the latter. We thus expect JOINTSPAR to spend less time on gradient computation per iteration. However, the total time per iteration can vary based on the method and the number of processes used. Therefore, to ensure a fair



(a) MNIST

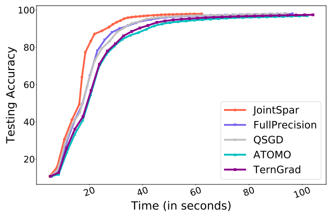


(b) CIFAR10

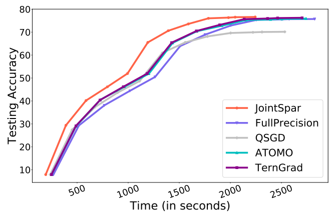


(c) CIFAR100

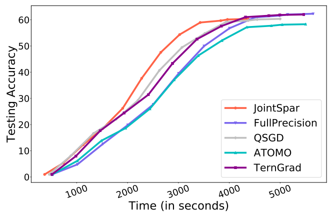
Figure 4. Convergence in terms of the number of iterations, confirming that our method JOINTSPAR enjoys a similar convergence rate as most baselines.



(a) MNIST



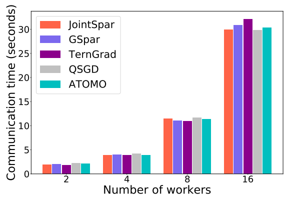
(b) CIFAR10



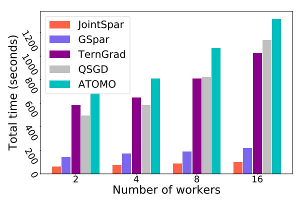
(c) CIFAR100

Figure 5. Convergence in terms of wall-clock time, confirming our method JOINTSPAR's faster convergence than baselines.

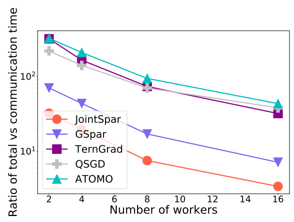
comparison, we compare the ratio of the computation time to the total time per iteration, as we increase the number of processes. By examining this ratio, we are accounting for the different total time per iteration. Specifically, we measure the total time per epoch, the communication time per epoch, and their ratio. Here, we still use the MNIST dataset, and train a small convolutional neural network (CNN) model, i.e., LeNet (Wen et al., 2017; LeCun et al., 1998), on it. We use $M = 2, 4, 8, 16$ processes to collectively train the CNN model. We set all methods to keep half of the size of a full gradient vector, as in the previous subsection. The results are shown in Figure 6. We can see that all methods spend approximately the same amount of time on communication, due to their identical sparsity levels. However, JOINTSPAR enjoys a much smaller total time per epoch than the baselines. This is because JOINTSPAR reduces both the gradient computation and communication time thanks to its joint sparsification. As the number of processes increases, the ratio of total time vs. communication time becomes smaller. This increase in communication time is due to heavier overhead when there are more processes involved in communication. We can see that, for baseline methods, the ratio of total time vs. communication time is very large, i.e., at least 10 (up to 10^3). While being successful at reducing communication time, baseline methods causes the gradient computation time to become dominant among the total time. This verifies our very motivation to propose a new method that jointly performs sparsification on gradient computation and communication. From this experiment, we empirically observe that JOINTSPAR has the smallest ratio, meaning our method can effectively reduce the gradient computation time for each iteration.



(a) Communication Time

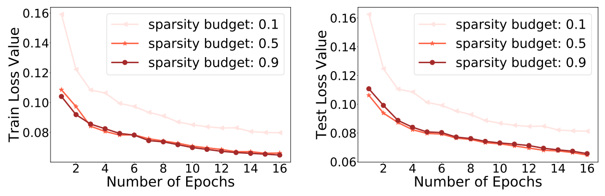


(b) Total Time



(c) Total to Communication Time Ratio

Figure 6. (a) Communication time per iteration of each method, (b) total time per iteration, and (c) total time to communication time ratio.



(a) Train (b) Test

Figure 7. Convergence rate for three different levels of sparsity budget s: low (0.1), medium (0.5), and high (0.9). Train loss is measured in (a), and test loss is measure in (b).

K. Parameter Sensitivity

We investigate the convergence behavior of JOINTSPAR when changing the sparsity budget s. The smaller s is, the sparser the compressed gradient is. We vary sparsity budget among $10\%, 50\%, 90\%$ of the full gradient size. We use the same LeNet on MNIST dataset. The results are shown in Figure 7, where x-axis and y-axis are number of epochs and loss value measured at the end of each epoch, respectively. Here, we use the convergence rate in terms of the number of epochs because we want to see effect of information change (caused by gradient sparsification) on the loss value. We observe that when there is less sparsification (i.e., larger sparsity budget), the time for each epoch is longer. Less sparsification can lead to faster convergence rate in training loss. This is because less compressed gradient loses less information of the full gradient. However, the convergence rate will not gain much if we increase s from 50% to 90%, due to some redundancy in the gradient. On the other hand, the best convergence rate in testing loss is obtained when the sparsity budget is not too small or too large, likely because very large sparsity budget can lead to overfitting. Cross validation can be used to choose the optimal value of s.