Practical Low-Rank Communication Compression in Decentralized Deep Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Lossy gradient compression has become a practical tool to overcome the communication bottleneck in centrally coordinated distributed training of machine learning models. However, algorithms for decentralized training with compressed communication over arbitrary connected networks have been more complicated, requiring additional memory and hyperparameters. We introduce a simple algorithm that directly compresses the model differences between neighboring workers using low-rank linear compressors applied to model differences. Inspired by the PowerSGD algorithm for centralized deep learning (Vogels et al., 2019), this algorithm uses power iteration steps to maximize the information transferred per bit. We prove that our method requires no additional hyperparameters, converges faster than prior methods, and is asymptotically independent of both the network and the compression. Out of the box, these compressors perform on par with state-of-the-art tuned compression algorithms in a series of deep learning benchmarks.

1 Introduction

The major advances in machine learning in the last decade have been made possible by very large datasets collected by multifaceted organizations. We live in a society where almost every individual owns electronic devices that collect huge amounts of data, which—when used collaboratively—could lead to transformative insights (Nedic, 2020). Often this data is bound to the device it is captured on. This might be for practical reasons of efficiency, or for more fundamental reasons such as privacy constraints. Centralized systems present a single point of failure both for data transfer, as well as for information security and privacy (Kairouz et al., 2019).

The paradigm of decentralized machine learning is key to leveraging the potential of this new kind of data. In this model, each connected device (node) has its own data. Each node can only communicate with few others, and together, the network of sparsely connected nodes aims to collaboratively train a model that minimizes a loss function on their joint dataset. The decentralized approach is not only useful in fundamentally decentralized systems, but the sparse communication patterns can sometimes even lead to efficiency gains in datacenter settings (Assran et al., 2019).

In bringing decentralized optimization algorithms into the realm of deep learning, the more-than gigabytes large model parameters and gradients (Rajbhandari et al., 2019; Brown et al., 2020) have spurred interest in communication compression techniques to reduce the bandwidth requirements of training such models. While practical plug-and-play compressors already exist for communication in centralized deep learning (Seide et al., 2014; Vogels et al., 2019) that can retain full model quality at significant communication reductions, current compression algorithms in decentralized optimization require the tuning of additional hyperparameters. This is unfortunate, since running many experiments to tune these hyperparameters is especially challenging and costly in a decentralized environment.

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

In this paper, we study a specific class of low-rank compressors for decentralized optimization inspired by (Vogels et al., 2019) that are reliable and require no tuning. Our low-rank compressor considers model parameters as matrices $\mathbf{X}$, and runs power iterations on the difference of two node's parameters $\mathbf{X}_i - \mathbf{X}_j$ to obtain a good low-rank approximation. Because these steps are linear, they can be executed in a distributed fashion, avoiding the expensive communication of full matrices.

We validate these plug-and-play compressors on decentralized image classification and language modeling tasks, and show that we can achieve competitive performance to other methods that require additionally tuned hyperparameters. This allows users to tune a learning rate in a simpler centralized setup, and then transition to decentralized learning without extra effort. We prove hyperparameter-free convergence on a subclass of random low-rank approximations. For consensus, our method converges faster than prior methods (Koloskova et al., 2019b). For stochastic optimization, our rates are asymptotically independent of the compression rate.

2 Related work

Communication compression in centrally coordinated learning. Communication compression is an established approach to alleviate the communication bottleneck in parallel optimization in deep learning. While Alistarh et al. (2017); Wen et al. (2017); Seide et al. (2014); Bernstein et al. (2019); Karimireddy et al. (2019b) study gradient quantization, it is also possible to only send gradient coordinates with the largest absolute values Lin et al. (2018); Stich et al. (2018); Wangni et al. (2018).

It has become clear that linear compression operators are practical in the centralized setting because they enable efficient all-reduce aggregation (Yu et al., 2018; Vogels et al., 2019; Cho et al., 2019). Ivkin et al. (2019) use linear sketches to detect which parameter coordinates change most in a distributed setting. Wang et al. (2018) observed that gradients in deep learning can be well approximated as low-rank matrices. The PowerSGD algorithm (Vogels et al., 2019), on which this work is based, is both linear and low-rank and performed well in a recent benchmark (Xu et al., 2020).

Decentralized optimization. Decentralized, or 'gossip'-based, optimization has been studied for many years (Tsitsiklis, 1984). Popular methods include those based on (stochastic) subgradient descent (Nedic & Ozdaglar, 2009) on node's local objective functions and with averaging between sparsely connected neighbors. Lian et al. (2017) evaluated the effectiveness of such schemes in the non-convex setting.

Tang et al. (2018) extend decentralized optimization with compressed communication, but require relatively high precision compression to ensure convergence. Koloskova et al. (2019a) and Tang et al. (2019) alleviate this constraint, supporting arbitrary-strength compression. Lu & Sa (2020) study a compression based on the assumption that model differences across connected nodes are coordinate-wise bounded. However, the abovementioned methods introduce additional hyperparameters specific to compression (e.g. the consensus stepsize)—an inconvenience we overcome in this work.

3 Decentralized machine learning

Decentralized multi-worker training of machine learning models has two key characteristics. Firstly, there is no central 'master' node and nodes can only communicate with a limited number of neighbors. This can either be a physical limitation of the network, or it can be desirable for performance. In a datacenter, sparse, decentralized connectivity leads to excellent scalability (Assran et al., 2019). The second characteristic is distributed data: each worker has their own data that potentially come from non-identical distributions. This can also be a hard limitation (e.g. to protect privacy), or it can be desirable for co-locality of computation and data.

The setup is formalized as follows: n worker nodes aim to collectively minimize a loss function

$$f(\mathbf{X}) \coloneqq \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{X}), \quad f_i(\mathbf{X}) \coloneqq \mathbb{E}_{\boldsymbol{\xi}_i \sim D_i} F_i(\mathbf{X}, \boldsymbol{\xi}_i)$$

over model parameters $\mathbf{X}$, where $f_i(\cdot)$ are smooth potentially non-convex loss functions over local data distributions D_i. We assume that $\mathbf{X} \in \mathbb{R}^{p\times q}$ where p represents the size of the 'input' and q is the output size. For linear models, this matrix representation is natural. For multi-layer networks, each weight and bias is considered separately, and for convolutional layers, q represents the number of input layers and the kernel size and p is the number of output channels.

The network topology is represented by an undirected connected graph G that connects nodes i with their neighbors $\mathcal{N}_i$ (including self-links). Communication between nodes i and j is typically weighted by the i, j-th entry of a *mixing matrix* $\mathbf{W} \in \mathbb{R}^{n,n}$ which is non-zero only for connected nodes. This matrix is chosen such that for any scalars $\mathbf{v} \in \mathbb{R}^n$ held by the nodes, repeated averaging (gossip) between connected nodes, $\mathbf{W}\mathbf{v}$, gradually leads to consensus, $\mathbf{v}_i \to \frac{1}{n}\sum_{i=1}^{n} \mathbf{v}_i \; \forall i$.

In stochastic gradient-based optimization, each worker typically has its own model parameters $\mathbf{X}_i$. Gossip averaging is used to bring the $\mathbf{X}_i$'s closer together and share information between nodes, while local stochastic gradient updates change $\mathbf{X}_i$ to fit local data. Our methods builds on the elegant DP-SGD algorithm (Lian et al., 2017). In DP-SGD, for each timestep t and each worker i,

$$\mathbf{X}_i^{(t+1)} \coloneqq \mathbf{X}_i^{(t)} - \eta \nabla f_i(\mathbf{X}_i^{(t)}, \boldsymbol{\xi}_{i,t}) + \sum_{j \in \mathcal{N}_i} W_{ij} \left(\mathbf{X}_j^{(t)} - \mathbf{X}_i^{(t)} \right), \tag{1}$$

where η_i is the learning rate and $\boldsymbol{\xi}_{i,t} \sim D_i$ represents a local data point. Note that each step requires sending and receiving the full model parameters between all pairs of connected neighbors, but that this communication can be overlapped with computation of the stochastic gradient.

4 Algorithm

Naively applying lossy communication compression (quantization / sparsification) to the gossip update in Eq. (1) leads to non-convergence. To support arbitrary compression, prior approaches introduce algorithmic modifications and additional hyperparameters to tune (Koloskova et al., 2019b; Tang et al., 2019, 2018). In this section, we introduce PowerGossip, a compressed consensus algorithm based on low-rank approximations and power iteration that does not suffer from these issues. Low-rank decomposition has already been shown to perform well in centralized deep learning (Vogels et al., 2019; Cho et al., 2019; Xu et al., 2020), and we find that they can be competitive with expensively tuned quantization- or sparsification-based algorithms for decentralized training as well.

PowerGossip is based on the premise that $\mathcal{C}_\mathbf{v}(\mathbf{X}) \coloneqq (\mathbf{X}\mathbf{v})\mathbf{v}^\top$, for a matrix $\mathbf{X} \in \mathbb{R}^{p \times q}$ and vector $\mathbf{v} \in R^q$ with $\|\mathbf{v}\|_2 = 1$, can be a reasonable low-rank approximation of $\mathbf{X}$ that can be communicated with only p floats instead of $p \times q$, given that all parties know $\mathbf{v}$. For the large weight matrices in deep learning, this reduction is significant. For a random $\mathbf{v}$, $\mathcal{C}_\mathbf{v}$ is a random projection, while for $\mathbf{v}$ being the top right singular vector, $\mathcal{C}_\mathbf{v}(\mathbf{X})$ is the best rank-1 approximation of $\mathbf{X}$ in the Frobenius norm.

We use the low-rank compressor $\mathcal{C}_\mathbf{v}$ to reduce communication in the gossip part of Eq. (1):

$$\mathbf{X}_i^{(t+1)} \coloneqq \mathbf{X}_i^{(t)} + \sum_{j \in \mathcal{N}_i} W_{ij}\, \mathcal{C}_{\mathbf{v}_{ij}}(\mathbf{X}_j^{(t)} - \mathbf{X}_i^{(t)}), \tag{2}$$

for a time-varying vector $\mathbf{v}_{ij}$ shared between each pair of connected workers. Due to linearity, $\mathcal{C}_\mathbf{v}(\mathbf{X}_j - \mathbf{X}_i) = (\mathbf{X}_j - \mathbf{X}_i)\mathbf{v}\mathbf{v}^\top = (\mathbf{X}_j\mathbf{v} - \mathbf{X}_i\mathbf{v})\mathbf{v}^\top$. Therefore, the compressed difference can be computed jointly by nodes i and j without ever communicating the full $\mathbf{X}_j - \mathbf{X}_i$. Thus any nodes i and j only need to exchange vectors instead of matrices.

The approximation quality of $\mathcal{C}_\mathbf{v}$ depends on the choice of the projection vector $\mathbf{v}$, and we leverage the mechanism of power iteration to find good ones. Every time (k) the compressor $\mathcal{C}_\mathbf{v}$ is used on some parameter difference $\mathbf{D}^{(k)} \coloneqq \mathbf{X}_j^{(k)} - \mathbf{X}_i^{(k)}$, we choose $\mathbf{v}^{(k)}$ based on the previous low-rank approximation. Starting with a random initial vector $\mathbf{v}^{(0)}$, we use

$$\mathbf{v}^{(2k+1)} \coloneqq \frac{\mathbf{D}^{(2k)}\mathbf{v}^{(2k)}}{\|\mathbf{D}^{(2k)}\mathbf{v}^{(2k)}\|}, \qquad \mathbf{v}^{(2k)} \coloneqq \frac{\mathbf{D}^{(2k-1)\top}\mathbf{v}^{(2k-1)}}{\|\mathbf{D}^{(2k-1)\top}\mathbf{v}^{(2k-1)}\|}, \qquad \forall k \in \mathbb{Z}_{\geq 0}. \tag{3}$$

If $\mathbf{X}_j^{(k)} - \mathbf{X}_i^{(k)}$ changes slowly over time, this procedure approaches power iteration and it finds the top eigenvector $\mathbf{v}$. This approach empirically leads to better approximations and faster convergence than compression with random projections.

Algorithm 1 describes how we use PowerGossip for stochastic optimization. Algorithm 2 presents the details of our compression scheme.

4.1 Properties

Linearity. Due to the linearity of matrix multiplication, we can compute a matrix-vector product $(\mathbf{X}_i - \mathbf{X}_j)\mathbf{v}$ with matrices stored on different workers in a distributed fashion as $(\mathbf{X}_i\mathbf{v}) - (\mathbf{X}_j\mathbf{v})$. This

Algorithm 1 Decentralized SGD with edge-wise compression

input model parameters $\mathbf{X}_i^{(0)} \in \mathbb{R}^{p\times q}$ for each node i out of n, randomly initialized identically
given a symmetric, doubly stochastic, diffusion matrix $\mathbf{W} \in \mathbb{R}^{N\times N}$
given a compressor $\mathcal{C}$ that can approximate $\mathbf{X}_i - \mathbf{X}_j$ with little communication
for each timestep t **at** each worker i **do**
    $\mathbf{G} \leftarrow$ a stochastic gradient $\nabla f(\mathbf{X}_i^{(t-1)}, \boldsymbol{\xi}_{i,t})$ for mini-batch $\boldsymbol{\xi}_{i,t}$
    $\mathbf{X}_i^{(t)} \leftarrow \mathbf{X}_i^{(t-1)} + \sum_{j\in\mathcal{N}_i} W_{ij}\mathcal{C}(\mathbf{X}_j^{(t-1)} - \mathbf{X}_i^{(t-1)}) - \eta\cdot\mathbf{G}$
end for

Algorithm 2 Rank-1 s-step PowerGossip compression for Algorithm 1

initialize a projection vector $\mathbf{v}_{ij} = -\mathbf{v}_{ji} \in \mathbb{R}^q$ for each pair of connected nodes i, j, initialized from an entry-wise standard normal distribution, stored on nodes i and j. Initialize $k \leftarrow 0$.
procedure $\mathcal{C}(\mathbf{X}_j - \mathbf{X}_i)$
    for s power iteration steps **do**
        increment $k \leftarrow k + 1$
        if $k \equiv 1 \bmod 2$ **then**
            $\hat{\mathbf{v}} \leftarrow \frac{\mathbf{v}_{ij}}{\|\mathbf{v}_{ij}\|}$
            $\mathbf{p}_j \leftarrow \mathbf{X}_j\hat{\mathbf{v}}, \qquad \mathbf{p}_i \leftarrow \mathbf{X}_i\hat{\mathbf{v}}$ ▷ computed on nodes i and j
            $\hat{\mathbf{Q}} \leftarrow (\mathbf{p}_j - \mathbf{p}_i)\hat{\mathbf{v}}^\top$
            $\mathbf{v}_{ij} \leftarrow \mathbf{p}_j - \mathbf{p}_i$ ▷ $\mathbf{v}_{ij}$ changes between $\mathbb{R}^p$ and $\mathbb{R}^q$
        else
            do the same, but with $\mathbf{X}$ transposed as in Eq. (3).
        end if
    end for
    return the approximation $\hat{\mathbf{Q}}$
end procedure
note that computations of $\mathcal{C}(\mathbf{X}_j - \mathbf{X}_i) = -\mathcal{C}(\mathbf{X}_i - \mathbf{X}_j)$ overlap and share communication.

circumvents communication of matrices by sending much smaller vectors instead. By compressing the differences of the models, we ensure that the models get closer to the average in every step without the need for additional 'consensus stepsize' like prior protocols. In particular, if two workers agree on the parameters and their difference is 0, then the compressed update will also be 0. This ensures that consensus is always a fixed-point of our method for arbitrary-strength compressors.

Low-rank compression. PowerGossip approximates differences between model parameters by low-rank matrices. The quality of these approximations depends on the power spectra of the differences. Similar to how top-k compression—which approximates a vector by its top k coordinate in absolute value, and zeros otherwise—works best when a few coordinates are much larger than the rest, low-rank compression can leverage the peaky power spectra found in deep learning (Vogels et al., 2019; Cho et al., 2019) to maximize information sent per bit. Our experiments in Section 6 confirm that low-rank compression is competitive with quantization- or sparsification-based approaches, while keeping our algorithm simple and free of hyperparameters.

Memory and computation complexity. The linear projection operations in PowerGossip are well suited for accelerator hardware used in deep learning (Vogels et al., 2019; Cho et al., 2019; Xu et al., 2020), and are typically even faster than compression based on random sparsification or quantization. Like in DP-SGD (Lian et al., 2017), this computation and the communication between nodes can be overlapped with gradient computation. Storing the previous projection vectors $\mathbf{v}$ requires memory linear in the number of connections per worker, but these vectors are very small compared to a full model (0.1–2% of the full model in our experiments). This yields lower memory usage than competing methods ChocoGossip Koloskova et al. (2019a) and DeepSqueeze Tang et al. (2019).

5 Theoretical analysis

5.1 Assumptions and setup

Loss functions. We make standard assumptions about our loss functions. Note that our analysis covers both functions satisfying (A1), as well as more general non-convex functions which do not.

(A1) f_i is μ**-convex** for $\mu \geq 0$ if it satisfies for any $\mathbf{X}$, and $\mathbf{X}^\star$ minimizing f

$$\nabla f_i(\mathbf{X}) \circ (\mathbf{X}^\star - \mathbf{X}) \leq -\Big(f_i(\mathbf{X}) - f_i(\mathbf{X}^\star) + \frac{\mu}{2}\|\mathbf{X} - \mathbf{X}^\star\|_F^2\Big)\,.$$

(A2) We assume $\{f_i\}$ are L**-smooth** and thus satisfy:

$$\|\nabla f_i(\mathbf{X}) - \nabla f_i(\mathbf{Y})\|_F \leq L\|\mathbf{X} - \mathbf{Y}\|_F\,, \text{ for any } i, \mathbf{X}\,, \mathbf{Y}\,.$$

(A3) Bounded variance: We assume there exist constants σ^2 and ζ^2 which bound the variance within and across different nodes, i.e. for any $\mathbf{X}$ we have

$$\mathbb{E}_{\boldsymbol{\xi}_i \sim D_i}\|\nabla F_i(\mathbf{X}, \boldsymbol{\xi}_i) - \nabla f_i(\mathbf{X})\|_F^2 \leq \sigma^2 \quad \text{and} \quad \tfrac{1}{N}\textstyle\sum_{i=1}^N \|\nabla f_i(\mathbf{X}) - \nabla f(\mathbf{X})\|_F^2 \leq \zeta^2\,.$$

Assumption A1 is known as *star-convexity* and is weaker than the usual definition of convexity (Stich & Karimireddy, 2019). While A3 requires both the variance within each node as well as across the nodes be bounded, we allow for heterogeneous (non-iid) data distributions across the nodes.

Communication network. We assume that we are given a mixing matrix $\mathbf{W} \in \mathbb{R}^{n\times n}$ and an underlying communication network over n nodes $([n], E)$ satisfying (A4):

(A4) $W_{ij} \neq 0$ only if $(i,j) \in E$, and $\mathbf{W} \in \mathbb{R}^{n\times n}$ is symmetric ($\mathbf{W}^\top = \mathbf{W}$) and doubly stochastic ($\mathbf{W}\mathbf{1} = \mathbf{1}, \mathbf{1}^\top\mathbf{W} = \mathbf{1}^\top$). Further, $\mathbf{W}^2$ has eigenvalues $1 = \lambda_1^2 \geq \lambda_w^2 \geq \ldots \lambda_n^2$ with **spectral gap** $\rho \coloneqq 1 - \lambda_2^2 > 0$.

Assumption (A4) characterizes the mixing matrix $\mathbf{W}$ for decentralized optimization and controls the rate of information spread in the network (Lian et al., 2017; Pu & Nedic, 2018). If $\mathbf{W}$ satisfies (A4) for $\rho > 0$, then the underlying communication network is undirected and strongly connected.

Compression operators. We introduce a new class of compression operators $\mathcal{C}(\cdot)$ and assume that every compressor used in Algorithm 1 satisfies (A5):

(A5) We assume that $\mathcal{C}$ is a δ-approximate unbiased **linear projection** operator i.e. for any $\mathbf{X}$ and $\mathbf{Y}$, the following are true for some $\delta > 0$:

$$\mathcal{C}(\mathbf{X} + \mathbf{Y}) = \mathcal{C}(\mathbf{X}) + \mathcal{C}(\mathbf{Y})\,, \quad \mathcal{C}(\mathcal{C}(\mathbf{X})) = \mathcal{C}(\mathbf{X})\,, \quad \text{and} \quad \mathbb{E}[\mathcal{C}(\mathbf{X})] = \delta X\,.$$

Consider a random-p sampler whose (i,j) element $[\mathcal{S}_p(\mathbf{X})]_{i,j}$ is $X_{i,j}$ with probability p and 0 otherwise. Then $\mathcal{S}_p(\cdot)$ is a linear projection operator satisfying (A5) with $\delta = p$.

For a second example closer to Algorithm 2, consider the following compressor for $\mathbf{X} \in \mathbb{R}^{p,q}$:

$$\mathcal{R}(\mathbf{X}) \coloneqq (\mathbf{X}\mathbf{u})\mathbf{u}^\top \text{ for } \mathbf{u} \sim S^{(q-1)}\,,$$

i.e. we project $\mathbf{X}$ along $\mathbf{u}$ which is sampled uniformly from the unit sphere. The operator $\mathcal{R}(\mathbf{X})$ approximates $\mathbf{X}$ as a product of two rank-1 matrices $\mathbf{u}$ and $\mathbf{X}\mathbf{u}$. Then, $\mathcal{R}(\cdot)$ is clearly linear in $\mathbf{X}$, is an unbiased projection operator, and satisfies (A5) with $\delta = \frac{1}{q}$. We can also approximate $\mathbf{X}$ by two rank-k matrices as $\mathcal{R}_k(\mathbf{X}) = (\mathbf{X}\mathbf{U})\mathbf{U}^\top$ for $\mathbf{U} \in \mathbb{R}^{q\times k}$ being a uniformly sampled orthonormal matrix. Then $\mathcal{R}_k(\cdot)$ satisfies (A5) with $\delta = \frac{k}{q}$. We can also define a left projection operator $\mathcal{L}(\mathbf{X}) \coloneqq \mathbf{v}(\mathbf{v}^\top\mathbf{X})$ for $\mathbf{v} \sim S^{(p-1)}$. The operator $\mathcal{L}(\cdot)$ approximates $\mathbf{X}$ with two rank-1 matrices $\mathbf{v}$ and $\mathbf{X}^\top\mathbf{v}$ and satisfies (A5) with $\delta = \frac{1}{p}$.

While (A5) defines a specific class of compression operators which are a subset of those considered in (Koloskova et al., 2019b), they can still be of arbitrary approximation quality $\delta > 0$.

5.2 Convergence rates

We study the rate of consensus as well as convergence of the objective function in stochastic optimization with compressed communication. Our analysis shows that our algorithm is not only simpler than the previous approaches, but also significantly faster. To simplify notation, we will use $\bar{\cdot}$ to indicate the average across the n nodes, e.g. $\bar{\mathbf{X}} \coloneqq \frac{1}{n}\sum_{i=1}^n \mathbf{X}_i$.

Compressed consensus. Suppose that every iteration, each worker i performs the following update:

$$\mathbf{X}_i^{(t)} \coloneqq \mathbf{X}_i^{(t-1)} + \sum_{j \in \mathcal{N}_i} W_{ij} \Big(\mathcal{C}_{ij}^{(t)}(\mathbf{X}_j^{(t-1)}) - \mathcal{C}_{ij}^{(t)}(\mathbf{X}_i^{(t-1)}) \Big) . \tag{4}$$

Each edge (i, j) can use a different compressor $\mathcal{C}_{ij}^{(t)}$ that can be varied over time. In this update, only compressed parameters are communicated.

Theorem I. *Assuming all compressors $\mathcal{C}_{ij}^{(t)}$ are δ-approximate satisfying* (A5) *and that the mixing matrix $\mathbf{W}$ has spectral gap ρ as in* (A4)*, then the update* (4) *achieves consensus at a q-linear rate:*

$$\frac{1}{N} \sum_{i=1}^{N} \mathbb{E} \left\| \mathbf{X}_i^{(t)} - \bar{\mathbf{X}}^{(0)} \right\|_F^2 \leq (1 - \rho\delta) \frac{1}{N} \sum_{i=1}^{N} \left\| \mathbf{X}_i^{(t-1)} - \bar{\mathbf{X}}^{(0)} \right\|_F^2 .$$

Note that update (4) requires no additional parameters and that our rate is linear in both δ and ρ. When $\delta = 1$, i.e. with uncompressed messages, the rate in I corresponds to the classical consensus rate (e.g. Xiao & Boyd, 2004a). In contrast, (Koloskova et al., 2019b) require a consensus stepsize, do not obtain q-linear rates, and are slower with a rate depending on $\rho^2\delta$ instead of our $\rho\delta$.

Compressed optimization. Consider the following algorithm where every node i performs the following updates using a sequence of predetermined stepsizes $\{\eta_t\}$:

$$\begin{aligned} \mathbf{Y}_i^{(t)} &\coloneqq \mathbf{X}_i^{(t-1)} - \eta_t \nabla F_i(\mathbf{X}, \boldsymbol{\xi}_{i,t}) \\ \mathbf{X}_i^{(t)} &\coloneqq \mathbf{Y}_i^{(t)} + \sum_{j \in \mathcal{N}_i} W_{ij} (\mathcal{C}_{ij}^{(t)}(\mathbf{Y}_j^{(t))}) - \mathcal{C}_{ij}^{(t)}(\mathbf{Y}_i^{(t))})) . \end{aligned} \tag{5}$$

This algorithm is like PowerGossip, but it applies the consensus update of (4) after a local gradient update rather than simultaneously. Again, the compressors are allowed to vary across edges and with time, and only compressed parameters are communicated. After running for T steps, we will randomly pick the final model given some weights $\{\alpha_t\}$ as

$$\mathbf{X}_i^{\text{out}} \coloneqq \mathbf{X}_i^{(t)} \text{ with probability proportional to } \alpha_t. \tag{6}$$

Theorem II. *Suppose that assumptions A2–A5 hold at every round of* (5)*. Then, in each of the following cases there exist a sequence of stepsizes $\{\eta_t\}$ and weights $\{\alpha_t\}$ such that the output $\bar{\mathbf{X}}^{out}$ computed using* (5) *and* (6) *is ε-accurate.*

- ***Non-convex:*** $\mathbb{E}\|\nabla f(\bar{\mathbf{X}}^{out})\|^2 \leq \varepsilon$ *after*
$$T = \mathcal{O}\left(\frac{L\sigma^2}{n\varepsilon^2} + \frac{\sqrt{L}(\zeta + \sigma)}{\rho\delta\varepsilon^{3/2}} + \frac{L}{\rho\delta\varepsilon} \right) \text{ rounds.}$$
- ***Convex:*** *If $\{f_i\}$ are convex and satisfy* (A1) *with $\mu = 0$, then $\mathbb{E}[f(\bar{\mathbf{X}}^{out})] \leq \varepsilon$ after*
$$T = \mathcal{O}\left(\frac{\sigma^2}{n\varepsilon^2} + \frac{\zeta + \sigma}{\rho\delta\varepsilon^{3/2}} + \frac{L}{\rho\delta\varepsilon} \right) \text{ rounds.}$$
- ***Strongly-convex:*** *If $\{f_i\}$ satisfy* (A1) *with $\mu > 0$, then $\mathbb{E}[f(\bar{\mathbf{X}}^{out})] \leq \varepsilon$ after*
$$T = \tilde{\mathcal{O}}\left(\frac{\sigma^2}{n\mu\varepsilon} + \frac{\zeta + \sigma}{\rho\delta\mu\sqrt{\varepsilon}} + \frac{L}{\rho\delta\mu} \log\Big(\frac{1}{\varepsilon}\Big) \right) \text{ rounds.}$$

Let us focus on the strongly convex case ignoring logarithmic factors. Theorem II proves that the iteration complexity is $\frac{\sigma^2}{n\mu\varepsilon} + \frac{\zeta+\sigma}{\rho\delta\mu\sqrt{\varepsilon}} + \frac{L}{\rho\delta\mu} \log\big(\frac{1}{\varepsilon}\big)$. This can be decomposed into three terms. The first stochastic term $\frac{\sigma^2}{n\mu\varepsilon}$ is independent of both the compression factor δ as well as spectral-gap ρ implying that these terms do not affect the asymptotic rates. It scales linearly with the number of nodes n. The second term $\frac{\zeta+\sigma}{\rho\delta\mu\sqrt{\varepsilon}}$ corresponds to the *drift* experienced and is a penalty due to computation of gradients at inexact points (Karimireddy et al., 2019a). However, this is asymptotically smaller than the stochastic term. Last is the optimization term $\frac{L}{\rho\delta\mu} \log\big(\frac{1}{\varepsilon}\big)$, which is the slowed down by a factor of $\rho\delta$. If $\rho\delta = 1$, this term matches the linear rate of gradient descent on strongly convex functions (Nesterov, 2004). In contrast, the optimization term of (Koloskova et al., 2019b) is sub-linear. The dependence on ρ and δ is linear in our rates while (Koloskova et al., 2019b) have a quadratic dependence on ρ. With exact communication ($\delta = 1$) we recover the rates of (Koloskova et al., 2020).

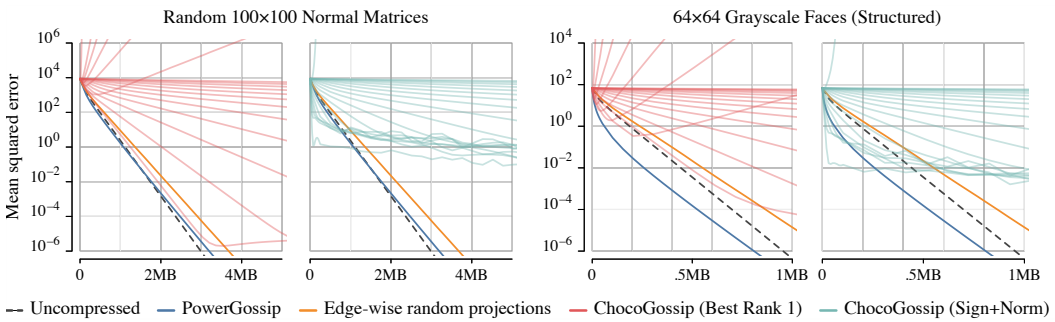


Figure 1: Consensus in an 8-ring. We study the level of consensus achieved as a function of bits transmitted by decentralized averaging. We compare out-of-the-box PowerGossip with power iterations and random projections against ChocoGossip (Koloskova et al., 2019b) with varying diffusion parameters. PowerGossip is competitive to the best tuned instances of ChocoGossip, and can leverage low rank structure in structured data (right).

6 Experimental analysis

We study PowerGossip in three settings. We first evaluate bits of communication required to reach **consensus** between 8 workers in a ring through (compressed) gossip averaging. The workers start with personal data matrices $\mathbf{X}_i$ ($i = 1 \dots 8$) that are either *unstructured*, from a 100×100 standard normal distribution, or *structured*, with 64×64 images from the Faces Database (AT&T Laboratories Cambridge). Then we evaluate PowerGossip in deep learning. We study the algorithm on the Cifar-10 **image classification** benchmark of Koloskova et al. (2019a), using a ResNet-20 and labeled images that are reshuffled between 8 workers every epoch. We also follow the **language modeling** experiment on WikiText-2 with an LSTM from Vogels et al. (2019) and extend it to a decentralized setting with 16 workers in a ring. Here, the training data is strictly partitioned between workers, dividing the source text equally over the workers in the original ordering.

In all experiments, we tune the hyperparameters of our baselines according to Appendix G and use the same learning rate as uncompressed centralized SGD for all instances of PowerGossip. Further details on the experimental settings are specified in Appendix C.

Random projections v.s. power iteration. Power iteration helps PowerGossip to leverage approximate low-rank structure in parameter differences between workers. This is illustrated by the consensus experiments in Figure 1. While on random data no compressed gossip algorithm outperforms full-precision gossip in bits to an arbitrary level of consensus, PowerGossip can reliably use structure in images of faces AT&T Laboratories Cambridge with less communication.

In our deep learning experiments, we also observe that PowerGossip requires less communication than random projections. The table on the right shows that more efficient communication leads to improved test accuracy within a fixed budget of 90 epochs.

Algorithm		Test loss
PowerGossip	w/ Random projections	4.627
	w/ Power iteration	4.565
DP-SGD	$35\times$ communication	4.583

Compression rate. The compression rate in PowerGossip is determined by the number of power iteration steps per stochastic gradient update. For models with large, square parameter tensors, like our LSTM (Appendix I), a single step of PowerGossip uses less than 0.1% of the bits used by an uncompressed averaging step. For a smaller model like the ResNet-20, the compression ratio is much lower. While our algorithm works for any compression rate, more gradient steps may be required to reach the same accuracy under extreme compression.

In our experiments, we use compression levels similar to those studied in related work. At those levels, PowerGossip achieves test performance similar to uncompressed DP-SGD in the same number of steps. Our compression level is varied through the number of power iterations per gradient update. More power iteration steps speed up consensus at the cost of increased communication in the same way as increasing the rank of the compressor does (see Appendix F), but it requires less memory to store the previous approximation and avoids an expensive orthogonalization step (Vogels et al., 2019). Table 1 shows the effect of varying our compression rate while keeping the number of epochs fixed.

Algorithm	η	γ	Test loss	Sent/epoch	
All-reduce (baseline)	tuned		4.46		
Uncompressed (DP-SGD)	tuned		4.58	15.0 GB	
PowerGossip (8 iterations)	default		4.73	127 MB	(122×)
PowerGossip (16 iterations)	default		4.63	230 MB	(67×)
PowerGossip (32 iterations)	default		4.57	437 MB	(35×)
Choco (Sign+Norm)	tuned	tuned	4.49	483 MB	(32×)
Choco (top-1%)	tuned	tuned	5.04	464 MB	(33×)

Table 1: Test loss achieved within 90 epochs on WikiText-2 language modeling with an LSTM on a 16-ring with strictly partitioned training data. PowerGossip requires no tuning, supports varying levels of compression, and is competitive to tuned ChocoSGD (Koloskova et al., 2019a) at a similar compression rate, matching the test loss of uncompressed DP-SGD.

Algorithm	η	γ	θ	Test accuracy	Sent/epoch	
All-reduce (baseline)	tuned			92.3%		
Uncompressed (DP-SGD)	tuned			92.1%	102 MB	
Choco (top-1%)	tuned	tuned		91.2%	3.1 MB	(33×)
Choco (Sign+Norm)	tuned	tuned		92.0%	3.2 MB	(32×)
Moniqua (2-bit)	tuned	tuned	tuned	90.7%	6.4 MB	(16×)
DeepSqueeze (Sign+Norm)	tuned	tuned		91.2%	3.2 MB	(32×)
PowerGossip (1 iteration)	default			91.7%	1.8 MB	(57×)
PowerGossip (2 iterations)	default			91.9%	3.0 MB	(34×)

Table 2: Test accuracy reached on Cifar-10 within 300 epochs with a ResNet-20 by decentralized optimization algorithms. PowerGossip has no additional hyperparameters and is competitive to all related work at a similar compression rate. Other algorithms used tuned learning rate η, averaging stepsize γ. Moniqua has an additional parameter θ that can be computed or tuned.

Hyper-parameter tuning. In our experiments, we have strictly used the same learning rate tuned for centralized, uncompressed SGD for all PowerGossip configurations. Tables 1 and 2 show that we can reach performance competitive to DP-SGD in both tasks, at a similar compression rate to the best tuned configurations of ChocoSGD (Koloskova et al., 2019b) and DeepSqueeze (Tang et al., 2019).

7 Conclusion

The introduction of communication compression to decentralized learning has come with algorithmic changes that introduced new hyperparameters required to support arbitrary compression operators. Focusing on a special class of linear low-rank compression, we presented simple parameter-free algorithms that perform as well as the extensively tuned alternatives in decentralized learning. Using power-iterations, this method can leverage the approximate low-rank structure present in deep learning updates to maximize the information transferred per bit, and reduce the communication between workers significantly at no loss in quality compared to full-precision decentralized algorithms. This is achieved with lower memory consumption than current state-of-the-art decentralized optimization algorithms that use communication compression.

Plug-and-play algorithms like PowerGossip can be directly deployed in a decentralized setting while reusing standard learning rates set in the centralized environment without compression. In view of the environmental, financial, and productivity impact of hyperparameter tuning in deep learning, such tuning-free methods are crucial for practical applicability of communication compression in decentralized machine learning.

8 Broader Impact

We believe that the field of decentralized learning plays a key role in translating the recent successes in deep learning from large organizations with large centralized datasets to smaller industry players and individuals. In particular, decentralized and therefore collaborative training on decentralized data is an important building block towards helping to better align each individual's data ownership and privacy with the resulting utility from jointly trained machine learning models. The ability to train collaboratively on decentralized data may lead to transformative insights in many fields, especially in applications where data is user-provided and privacy sensitive (Nedic, 2020). In addition to privacy, efficiency gains in distributed training reduce the environmental impact of training large machine learning models. The introduction of a practical and reliable communication compression technique is a small step towards achieving these goals on collaborative privacy-preserving and efficient decentralized learning.

References

Alistarh, D., Grubic, D., Li, J., Tomioka, R., and Vojnovic, M. QSGD: communication-efficient SGD via gradient quantization and encoding. In *NeurIPS*, pp. 1709–1720, 2017.

Assran, M., Loizou, N., Ballas, N., and Rabbat, M. Stochastic gradient push for distributed deep learning. In *Proc. ICML*, volume 97 of *Proceedings of Machine Learning Research*, pp. 344–353, 2019.

AT&T Laboratories Cambridge. AT&T database of faces. URL `https://scikit-learn.org/0.19/datasets/olivetti_faces.html`.

Bernstein, J., Zhao, J., Azizzadenesheli, K., and Anandkumar, A. signsgd with majority vote is communication efficient and fault tolerant. In *ICLR*, 2019.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020.

Cho, M., Muthusamy, V., Nemanich, B., and Puri, R. GradZip: Gradient compression using alternating matrix factorization for large-scale deep learning, 2019.

Frankle, J., Schwab, D. J., and Morcos, A. S. The early phase of neural network training. In *ICLR*, 2020.

Ivkin, N., Rothchild, D., Ullah, E., Braverman, V., Stoica, I., and Arora, R. Communication-efficient distributed SGD with sketching. In *NeurIPS*, pp. 13144–13154, 2019.

Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., D'Oliveira, R. G. L., Rouayheb, S. E., Evans, D., Gardner, J., Garrett, Z., Gascón, A., Ghazi, B., Gibbons, P. B., Gruteser, M., Harchaoui, Z., He, C., He, L., Huo, Z., Hutchinson, B., Hsu, J., Jaggi, M., Javidi, T., Joshi, G., Khodak, M., Konecný, J., Korolova, A., Koushanfar, F., Koyejo, S., Lepoint, T., Liu, Y., Mittal, P., Mohri, M., Nock, R., Özgür, A., Pagh, R., Raykova, M., Qi, H., Ramage, D., Raskar, R., Song, D., Song, W., Stich, S. U., Sun, Z., Suresh, A. T., Tramèr, F., Vepakomma, P., Wang, J., Xiong, L., Xu, Z., Yang, Q., Yu, F. X., Yu, H., and Zhao, S. Advances and open problems in federated learning. *CoRR*, abs/1912.04977, 2019.

Karimireddy, S. P., Kale, S., Mohri, M., Reddi, S. J., Stich, S. U., and Suresh, A. T. SCAFFOLD: stochastic controlled averaging for on-device federated learning. *CoRR*, abs/1910.06378, 2019a.

Karimireddy, S. P., Rebjock, Q., Stich, S. U., and Jaggi, M. Error feedback fixes signsgd and other gradient compression schemes. In *Proc. ICML*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3252–3261, 2019b.

Koloskova, A., Lin, T., Stich, S. U., and Jaggi, M. Decentralized deep learning with arbitrary communication compression. *CoRR*, abs/1907.09356, 2019a.

Koloskova, A., Stich, S. U., and Jaggi, M. Decentralized stochastic optimization and gossip algorithms with compressed communication. In *Proc. ICML*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3478–3487, 2019b.

Koloskova, A., Loizou, N., Boreiri, S., Jaggi, M., and Stich, S. U. A unified theory of decentralized SGD with changing topology and local updates. *CoRR*, abs/2003.10422, 2020.

Lian, X., Zhang, C., Zhang, H., Hsieh, C., Zhang, W., and Liu, J. Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent. In *NeurIPS*, pp. 5330–5340, 2017.

Lin, Y., Han, S., Mao, H., Wang, Y., and Dally, B. Deep gradient compression: Reducing the communication bandwidth for distributed training. In *ICLR*, 2018.

Lu, Y. and Sa, C. D. Moniqua: Modulo quantized communication in decentralized SGD. *CoRR*, abs/2002.11787, 2020.

Nedic, A. Distributed gradient methods for convex machine learning problems in networks: Distributed optimization. *IEEE Signal Process. Mag.*, 37(3):92–101, 2020.

Nedic, A. and Ozdaglar, A. E. Distributed subgradient methods for multi-agent optimization. *IEEE Trans. Automat. Contr.*, 54(1):48–61, 2009.

Nesterov, Y. E. *Introductory Lectures on Convex Optimization - A Basic Course*, volume 87 of *Applied Optimization*. Springer, 2004. ISBN 978-1-4613-4691-3. URL https://doi.org/10.1007/978-1-4419-8853-9.

Pu, S. and Nedic, A. Distributed stochastic gradient tracking methods. *CoRR*, abs/1805.11454, 2018.

Rajbhandari, S., Rasley, J., Ruwase, O., and He, Y. Zero: Memory optimization towards training A trillion parameter models. *CoRR*, abs/1910.02054, 2019.

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 *INTERSPEECH*, pp. 1058–1062, 2014.

Stich, S. U. and Karimireddy, S. P. The error-feedback framework: Better rates for SGD with delayed gradients and compressed communication. *CoRR*, abs/1909.05350, 2019.

Stich, S. U., Cordonnier, J., and Jaggi, M. Sparsified SGD with memory. In *NeurIPS*, pp. 4452–4463, 2018.

Tang, H., Gan, S., Zhang, C., Zhang, T., and Liu, J. Communication compression for decentralized training. In *NeurIPS*, pp. 7663–7673, 2018.

Tang, H., Lian, X., Qiu, S., Yuan, L., Zhang, C., Zhang, T., and Liu, J. Deepsqueeze: Parallel stochastic gradient descent with double-pass error-compensated compression. *CoRR*, abs/1907.07346, 2019.

Tsitsiklis, J. N. *Problems in decentralized decision making and computation*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1984. URL http://hdl.handle.net/1721.1/15254.

Vogels, T., Karimireddy, S. P., and Jaggi, M. Powersgd: Practical low-rank gradient compression for distributed optimization. In *NeurIPS*, pp. 14236–14245, 2019.

Wang, H., Sievert, S., Liu, S., Charles, Z. B., Papailiopoulos, D. S., and Wright, S. ATOMO: communication-efficient learning via atomic sparsification. In *NeurIPS*, pp. 9872–9883, 2018.

Wangni, J., Wang, J., Liu, J., and Zhang, T. Gradient sparsification for communication-efficient distributed optimization. In *NeurIPS*, pp. 1306–1316, 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 *NeurIPS*, pp. 1509–1519, 2017.

Xiao, L. and Boyd, S. P. Fast linear iterations for distributed averaging. *Syst. Control. Lett.*, 53(1): 65–78, 2004a.

Xiao, L. and Boyd, S. P. Fast linear iterations for distributed averaging. *Syst. Control. Lett.*, 53(1): 65–78, 2004b.

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.

Yu, M., Lin, Z., Narra, K., Li, S., Li, Y., Kim, N. S., Schwing, A. G., Annavaram, M., and Avestimehr, S. Gradiveq: Vector quantization for bandwidth-efficient gradient aggregation in distributed CNN training. In *NeurIPS*, pp. 5129–5139, 2018.

A Compressed Consensus (Proof of Theorem I)

Recall that the consensus update for each node i performs (4):

$$\mathbf{X}_i^{(t)} = \mathbf{X}_i^{(t-1)} + \sum_{j \in \mathcal{N}_i} W_{ij}(\mathcal{C}_{ijt}(\mathbf{X}_j^{(t-1)}) - \mathcal{C}_{ijt}(\mathbf{X}_i^{(t-1)}))\,.$$

Lemma 1 (Preserves average)**.** *For every step of* (4)*,* $\bar{\mathbf{X}}^{(t)} = \bar{\mathbf{X}}^{(0)}$.

Proof. Note that for every edge $(i,j) \in E$, we add to node i exactly what is subtracted from node j. This preserves the average:

$$\begin{aligned}
\bar{\mathbf{X}}^{(t)} &= \frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{X}_i^{(t-1)} + \sum_{j\in\mathcal{N}_i} W_{ij}(\mathcal{C}_{ijt}(\mathbf{X}_j^{(t-1)}) - \mathcal{C}_{ijt}(\mathbf{X}_i^{(t-1)}))\right) \\
&= \bar{\mathbf{X}}^{(t-1)} + \frac{1}{n}\sum_{(i,j)\in E}\Big(W_{ij}(\mathcal{C}_{ijt}(\mathbf{X}_j^{(t-1)}) - \mathcal{C}_{ijt}(\mathbf{X}_i^{(t-1)})) + W_{ji}(\mathcal{C}_{jit}(\mathbf{X}_i^{(t-1)}) - \mathcal{C}_{jit}(\mathbf{X}_j^{(t-1)}))\Big) \\
&= \bar{\mathbf{X}}^{(t-1)}\,.
\end{aligned}$$

The last equality follows because $W_{ij} = W_{ji}$ and $\mathcal{C}_{ijt} = \mathcal{C}_{jit}$. □

Lemma 2 (Effect of compression)**.** *Assuming* (A4) *and* (A5) *hold, the iteration* (4) *satisfies*

$$\|\mathbf{\Delta}_i^{(t)}\|_F^2 \le (1-\delta)\|\mathbf{\Delta}_i^{(t-1)}\|_F^2 + \delta\|\sum_{j\in[N]} W_{ij}\mathbf{\Delta}_j^{(t-1)}\|_F^2\,.$$

where we define $\mathbf{\Delta}_i^{(t)} := \mathbf{X}_i^{(t)} - \bar{\mathbf{X}}^{(0)}$.

Proof. Starting from the consensus update and the fact that $\sum_j W_{ij} = 1$, we have

$$\begin{aligned}
\mathbf{X}_i^{(t)} &= \mathbf{X}_i^{(t-1)} + \sum_{j\in\mathcal{N}_i} W_{ij}(\mathcal{C}_{ijt}(\mathbf{X}_j^{(t-1)}) - \mathcal{C}_{ijt}(\mathbf{X}_i^{(t-1)})) \\
&= \mathbf{X}_i^{(t-1)} + \sum_{j\in[N]} W_{ij}(\mathcal{C}_{ijt}(\mathbf{X}_j^{(t-1)} - \mathbf{X}_i^{(t-1)})) \\
&= \mathbf{X}_i^{(t-1)} + \sum_{j\in[N]} W_{ij}\Pi_{ijt}(\mathbf{X}_j^{(t-1)} - \mathbf{X}_i^{(t-1)})\,.
\end{aligned}$$

The second equality used $W_{ij} \neq 0$ only if $(i,j) \in E$ and the linearity of the compressor. Finally, since $\mathcal{C}_{ijt}$ is a linear projection, we can replace it by a projection matrix Π_{ijt}. Recall that $\mathcal{C}_{ijt}$ is an δ-approximate linear projection which implies that Π_{ijt} satisfies

$$\mathbb{E}[\Pi_{ijt}] = \mathbb{E}[\Pi_{ijt}^\top] = \mathbb{E}[\Pi_{ijt}^\top\Pi_{ijt}] = \delta I\,. \tag{7}$$

Further, since Π_{ijt} is a projection matrix, we have for any i, j

$$\begin{aligned}
&\Pi_{ijt}^\top \preceq I \\
\Rightarrow &\Pi_{ijt}^\top\Pi_{ikt} \preceq \Pi_{ikt} \\
\Rightarrow &\mathbb{E}[\Pi_{ijt}^\top\Pi_{ikt}] \preceq \mathbb{E}[\Pi_{ikt}] = \delta I\,.
\end{aligned}$$

Note that we did not require any sort of independence between the projections $\Pi_{ijt}^\top\Pi_{ikt}$ in the above derivation. Armed with these properties of the projection matrices, we turn our attention to the error term defined as $\mathbf{\Delta}_i^{(t)} := \mathbf{X}_i^{(t)} - \bar{\mathbf{X}}^{(0)}$. Our previous expression for $\mathbf{X}_i^{(t)}$ implies that

$$\mathbf{\Delta}_i^{(t)} = \mathbf{\Delta}_i^{(t-1)} + \sum_{j\in[n]} W_{ij}\Pi_{ijt}(\mathbf{\Delta}_j^{(t-1)} - \mathbf{\Delta}_i^{(t-1)})\,.$$

Expanding ${\boldsymbol{\Delta}_i^{(t)}}^\top \boldsymbol{\Delta}_i^{(t)}$ and taking expectations on both sides gives

$$\begin{aligned}
\mathbb{E}[{\boldsymbol{\Delta}_i^{(t)}}^\top \boldsymbol{\Delta}_i^{(t)}] &= {\boldsymbol{\Delta}_i^{(t-1)}}^\top \boldsymbol{\Delta}_i^{(t-1)} + \sum_{j\in[n]} W_{ij} {\boldsymbol{\Delta}_i^{(t-1)}}^\top \mathbb{E}[\Pi_{ijt}](\boldsymbol{\Delta}_j^{(t-1)} - \boldsymbol{\Delta}_i^{(t-1)}) \\
&\quad + \sum_{j\in[n]} W_{ij} (\boldsymbol{\Delta}_j^{(t-1)} - \boldsymbol{\Delta}_i^{(t-1)})^\top \mathbb{E}[\Pi_{ijt}^\top] \boldsymbol{\Delta}_i^{(t-1)} \\
&\quad + \sum_{j,k\in[n]} W_{ij} W_{ik} (\boldsymbol{\Delta}_j^{(t-1)} - \boldsymbol{\Delta}_i^{(t-1)})^\top \mathbb{E}[\Pi_{ijt}^\top \Pi_{ikt}](\boldsymbol{\Delta}_k^{(t-1)} - \boldsymbol{\Delta}_i^{(t-1)}) \\
&\preceq {\boldsymbol{\Delta}_i^{(t-1)}}^\top \boldsymbol{\Delta}_i^{(t-1)} + \sum_{j\in[n]} \delta W_{ij} {\boldsymbol{\Delta}_i^{(t-1)}}^\top (\boldsymbol{\Delta}_j^{(t-1)} - \boldsymbol{\Delta}_i^{(t-1)}) \\
&\quad + \sum_{j\in[n]} \delta W_{ij} (\boldsymbol{\Delta}_j^{(t-1)} - \boldsymbol{\Delta}_i^{(t-1)})^\top \boldsymbol{\Delta}_i^{(t-1)} \\
&\quad + \sum_{j,k\in[n]} \delta W_{ij} W_{ik} (\boldsymbol{\Delta}_j^{(t-1)} - \boldsymbol{\Delta}_i^{(t-1)})^\top (\boldsymbol{\Delta}_k^{(t-1)} - \boldsymbol{\Delta}_i^{(t-1)}) \\
&= {\boldsymbol{\Delta}_i^{(t-1)}}^\top \boldsymbol{\Delta}_i^{(t-1)} - \delta \boldsymbol{\Delta}_i^{(t-1)} {\boldsymbol{\Delta}_i^{(t-1)}}^\top + \sum_{j,k\in[n]} \delta W_{ij} W_{jk} {\boldsymbol{\Delta}_j^{(t-1)}}^\top \boldsymbol{\Delta}_k^{(t-1)}.
\end{aligned}$$

The second matrix inequality used the fact that if $A \preceq B$ then $C^\top A C \preceq C^\top B C$ for any C. The equality in the third step pulled out the terms which only depend on i from the expressions and used our assumption (A4) that $\sum_j W_{ij} = \sum_i W_{ij} = 1$. Taking trace on both sides and using $\mathrm{Tr}(AB) = \mathrm{Tr}(BA)$ we can simplify the expression as

$$\mathbb{E}[\mathrm{Tr}({\boldsymbol{\Delta}_i^{(t)}}^\top \boldsymbol{\Delta}_i^{(t)})] \leq (1-\delta)\,\mathrm{Tr}({\boldsymbol{\Delta}_i^{(t-1)}}^\top \boldsymbol{\Delta}_i^{(t-1)}) + \delta\,\mathrm{Tr}((\sum_{j\in[n]} W_{ij}\boldsymbol{\Delta}_j)^\top (\sum_{j\in[n]} W_{ij}\boldsymbol{\Delta}_j))$$

The lemma now follows by the definition of Frobenius norm $\|Z\|_F^2 = \mathrm{Tr}(Z^\top Z)$. □

Lemma 3 (Effect of mixing)**.** *Assuming that* $\mathbf{W}$ *has a spectral gap* ρ *as in* (A4) *and* $\boldsymbol{\Delta}_i^{(t)} := \mathbf{X}_i^{(t)} - \bar{X}^{(0)}$*, we have*

$$\frac{1}{n}\sum_{i\in[n]} \Big\| \sum_{j\in[n]} W_{ij} \boldsymbol{\Delta}_j^{(t-1)} \Big\|_F^2 \leq (1-\rho)\frac{1}{n}\sum_{i\in[n]} \|\boldsymbol{\Delta}_i^{(t-1)}\|_F^2 \,.$$

Proof. Follows from standard mixing arguments such as in (Xiao & Boyd, 2004b). □

Averaging lemma 2 over the nodes i and then applying Lemma 3 gives

$$\begin{aligned}
\frac{1}{n}\sum_{i\in[n]} \|\boldsymbol{\Delta}_i^{(t)}\|_F^2 &\leq (1-\delta)\frac{1}{n}\sum_{i\in[n]} \|\boldsymbol{\Delta}_i^{(t-1)}\|_F^2 + \delta\frac{1}{n}\sum_{i\in[n]} \Big\| \sum_{j\in[n]} W_{ij}\boldsymbol{\Delta}_j^{(t-1)} \Big\|_F^2 \\
&\leq (1-\delta+\delta(1-\rho))\frac{1}{n}\sum_{i\in[n]} \|\boldsymbol{\Delta}_i^{(t-1)}\|_F^2 \\
&= (1-\rho\delta)\frac{1}{n}\sum_{i\in[n]} \|\boldsymbol{\Delta}_i^{(t-1)}\|_F^2 \,.
\end{aligned}$$

This proves the statement of Theorem I. □

B Compressed optimization (Proof of Theorem II)

We will use two main results proved in the previous section about our consensus step: that the average is preserved (Lemma 1), and that every step is a contraction in expectation (Theorem I). Any consensus operator which satisfies these two properties directly ensures convergence of the stochastic optimization method by the proof technique of (Koloskova et al., 2020). In particular, this shows that we satisfy Assumption 4 of (Koloskova et al., 2020) with $p = \rho\delta$. Replacing p with $\rho\delta$ in their Theorem 2 yields the desired rates. □

C Experimental settings

Tables 3, 4 and 5 describe the implementation details of our experiments.

Table 3: Default experimental settings for Cifar-10/ResNet-20 (based on Koloskova et al., 2019a)

Dataset	Cifar-10
Data augmentation	random horizontal flip and random 32×32 cropping
Architecture	ResNet-20
Training objective	cross entropy
Evaluation objective	top-1 accuracy
Number of workers	8
Topology	ring
Network W_{ij}	0.436 for neighbors i, j, 0.128 if $i = j$, 0 otherwise (optimized for largest spectral gap)
Data	reshuffled between workers every epoch
Batch size	$128 \times$ number of workers
Momentum	0.9
Learning rate	Tuned. PowerGossip uses the same as uncompressed centralized all-reduce.
LR decay	$/10$ at epoch 150 and 250
LR warmup	Step-wise linearly within 5 epochs, starting from 0.1
# Epochs	300
Weight decay	10^{-4}, 0 for BatchNorm parameters
Repetitions	6, with varying seeds
Reported metric	Worst result of any worker of the worker's mean test accuracy over the last 5 epochs

Table 4: Default experimental settings for WikiText-2 (based on Vogels et al., 2019)

Dataset	Word-level WikiText-2
Tokenizer	Spacy
Architecture	3-layer LSTM
Training objective	cross entropy
Evaluation objective	cross entropy / perplexity
Number of workers	16
Topology	ring
Network W_{ij}	$\frac{1}{3}$ for neighbors i, j, $\frac{1}{3}$ if $i = j$, 0 otherwise (common settings, worked better for DPSGD than weights used for Cifar-10)
Data	Source text strictly divided into 16 equal chunks, always remain on worker
Batch size	$64 \times$ number of workers
Momentum	0.0
Learning rate	Tuned. PowerGossip uses the same as uncompressed centralized all-reduce.
LR decay	$/10$ at epoch 60 and 80
LR warmup	Step-wise linearly within 5 epochs, starting from 1.25
# Epochs	90
Weight decay	0.0
Repetitions	2
Reported metric	Worst result of any worker of the worker's mean test cross entropy over the last 5 epochs

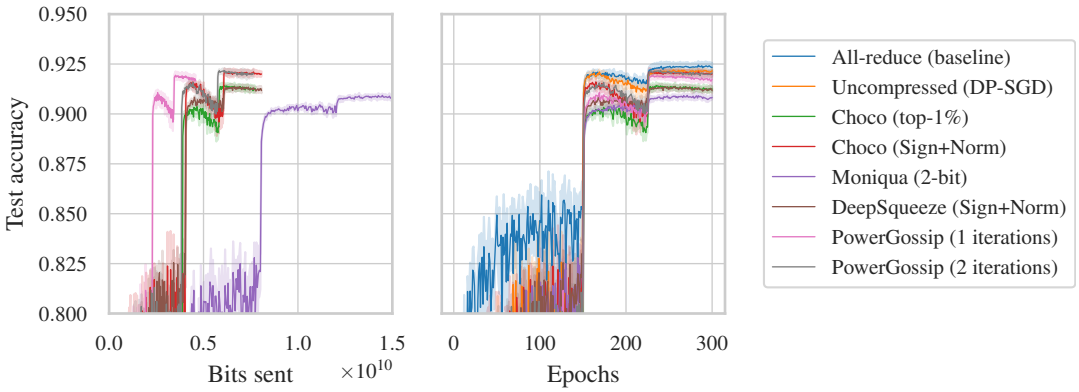
Table 5: Experimental settings for Consensus

Number of workers	8
Topology	ring
Network W_{ij}	0.436 for neighbors i, j, 0.128 if $i = j$, 0 otherwise (optimized for largest spectral gap)
Data	100×100 random normal data or 8 randomly selected 64×64 faces from (AT&T Laboratories Cambridge)
Objective	minimize $\frac{1}{8} \sum_{i=1}^{8} \left(\mathbf{X}_i^{(t)} - \bar{X}^{(0)} \right)^2$

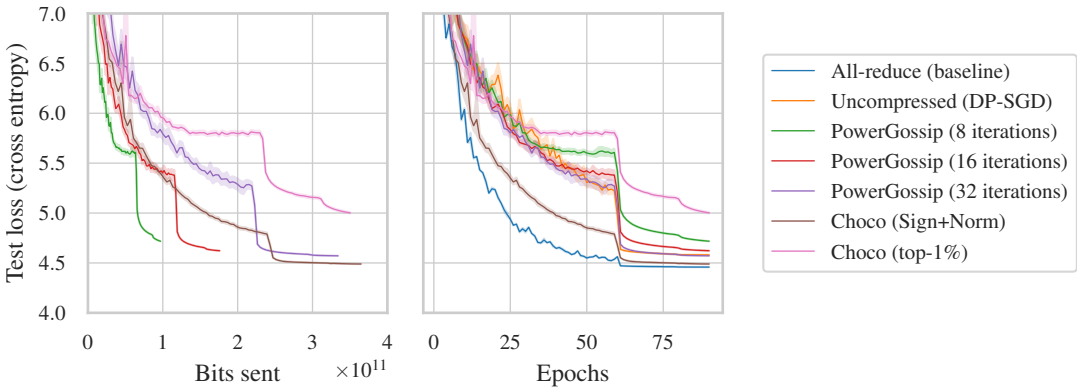
D Convergence curves

Below, we plot the convergence curves in terms of test accuracy, as a function of either gradient updates (epochs) or bits sent per worker. In all our experiments, we have used a fixed number of epochs and a learning rate schedule that is common for full precision centralized training. It is possible that experiments with high communication compression would benefit from more epochs or a slightly different learning rate schedule.

D.1 ResNet-20 on Cifar-10



D.2 LSTM on WikiText-2

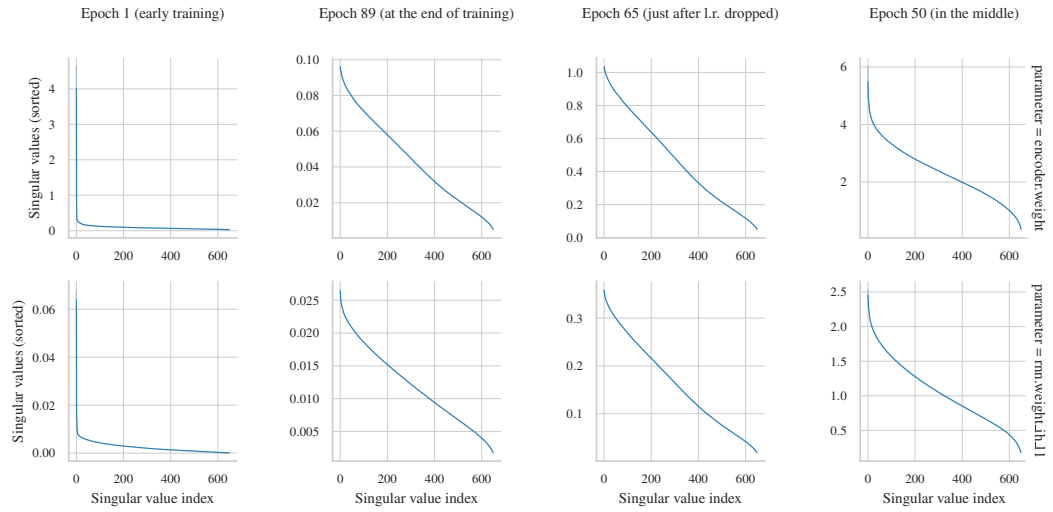


E The power spectrum of parameter differences

E.1 LSTM Training

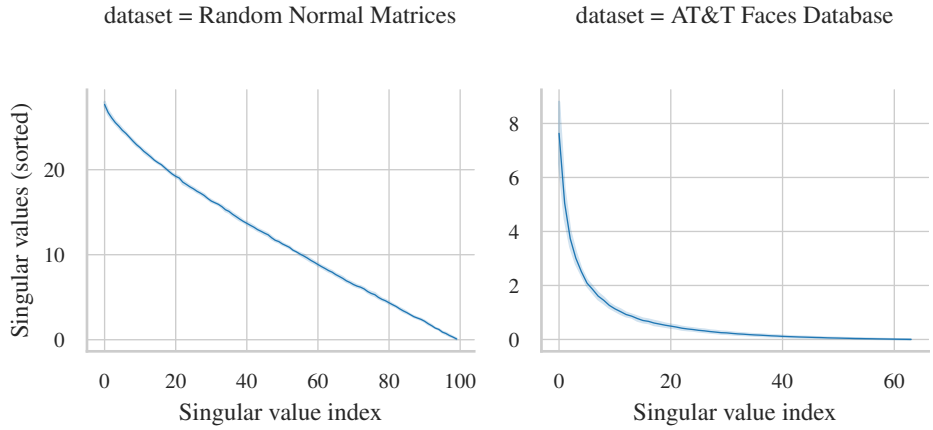
The plots below show the power spectra of parameter differences observed while training the LSTM (Appendix I). We train with 16 workers connected in a ring, using PowerGossip with 32 power iterations per gradient update. During training, we record the power spectra of the differences between the parameters of connected workers 0-1, 4-5 and 8-9 at 4 different training stages. Lines are averages of the spectra observed between the three worker pairs.

The power spectra change significantly over time, but at most stages, they show that a few singular vectors cary more weight than others. This structure can be exploited by PowerGossip with power iterations. Especially in early training, the power spectra are peaky. This phase has been observed to be critical for successful training of non-convex models (Frankle et al., 2020).



E.2 Consensus

The effect of a peaky spectrum on PowerGossip shows in our *consensus* experiments. When we plot the spectra of parameter differences between neighboring workers at initialization, we see that faces from the Faces Database (AT&T Laboratories Cambridge) can be approximated better with a low-rank approximation than random normal matrices. This is the reason why, in Figure 1, PowerGossip with power iterations is more efficient per-bit than uncompressed gossip for this dataset.



F Changing rank vs changing # power iterations

PowerSGD (Vogels et al., 2019), the algorithm on which PowerGossip is inspired, control their compression rate by varying the rank of the low-rank approximations. While this strategy is effective in terms of quality, it requires their projection matrices to be orthogonalized at every step of power iteration, rather than normalized. This operation scales as the square of the approximation rank, and is reported to be the most expensive step of the algorithm. A second disadvantage of using a high rank is that the memory required to store previous low-rank approximations scales linearly with the rank as well.

In PowerGossip, we adopt an alternative approach where we use multiple rank-1 power iteration steps per gradient update instead of one step with higher accuracy. In the table below, we show that this alteration has no impact on the performance of our method, evaluated with a fixed budget of 90 epochs on WikiText-2 language modeling. For the same total communication budget, we reach similar test loss.

Sent/epoch	PowerGossip rank	Num. power iterations	WikiText-2 test loss
127 MB	1	8	4.73
230 MB	1	16	4.63
437 MB	1	32	4.58
	2	16	4.58
	4	8	4.58
	8	4	4.58

G Hyperparameters

G.1 Consensus

In Figure 1, we plot results obtained with two compressors in ChocoGossip Koloskova et al. (2019b), using 20 consensus step size parameters γ ranging from 7.6×10^{-5} to 1 on an exponential grid. The optimal hyperparameter depends on the compressor used.

G.2 ResNet-20 on Cifar-10

The table below specifies the optimizer-specific hyperparameters that we used in our experiments. For our baselines DeepSqueeze and ChocoSGD, we use tuned hyperparameters from Koloskova et al. (2019a).

	Learning rate η		Consensus rate γ		Modulo parameter θ	
Method	Tested	Used	Tested	Used	Tested	Used
All-reduce (baseline)	$\{0.8, 1.13, 1.6\}$	1.13				
Uncompressed DP-SGD	$\{0.8, 1.13, 1.6\}$	1.13				
Choco (top-1%)	$\{0.96, 1.2, 1.6\}^\star$	1.13	$\{0.025, 0.0375, 0.075, 0.15\}^\star$	0.0375		
Choco (Sign+Norm)	$\{1.2, 1.6, 2.4\}^\star$	1.6	$\{0.15, 0.2, 0.45, 1\}^\star$	0.45		
Moniqua (2-bit)	$\{0.1, 0.2, 0.4, 0.8\}$	0.4	$\{0.01, 0.005, 0.0025, 0.0012\}^\dagger$	0.005	$\{0.125, 0.25, 0.5\}$	0.25
DeepSqueeze (Sign+Norm)	$\{0.24, 0.48, 0.96\}$	0.48	$\{0.005, 0.01, 0.05\}^\star$	0.01		
PowerGossip (1 iteration)		11.3				
PowerGossip (2 iterations)		11.3				

$\star$: based on published tuned parameters and the tuning strategy from the authors of ChocoSGD (Koloskova et al., 2019a).

†: the concensus step size was tuned after the other parameters, not in a full grid.

G.3 LSTM on WikiText-2

The table below specifies the optimizer-specific hyperparameters that we used in our experiments.

	Learning rate η		Consensus rate γ		Modulo parameter θ	
Method	Tested	Used	Tested	Used	Tested	Used
All-reduce (baseline)	$\{15, 20, 27.5, 35, 47.5\}$	47.5				
Uncompressed DP-SGD	$\{15, 20, 27.5, 35, 47.5\}$	47.5				
Choco (top-1%)†	$\{47.5\}$		$\{0.01, 0.1, 0.2, 0.4, 0.8\}$			
Choco (Sign+Norm)	$\{35, 47.5\}$	47.5	$\{0.4, 0.6, 0.8, 1.0\}$	0.8		
PowerGossip ($\star$ iterations)		47.5				

†: did not converge. We did not report this result, as more tuning may help.

H Compared-to algorithm implementations

In the sections below, we describe the implementation details of the algorithms we compare to. We provide the code for our implementations on Github (after deanonimization).

H.1 ChocoSGD

We implement Algorithm 1 of (Koloskova et al., 2019a), which differs slightly from Algorithm 2 in (Koloskova et al., 2019b), in that it executes consensus steps and gradient updates in parallel like DP-SGD.

We use three compressors in our experiments. As customary, we compress each tensor parameter of our neural networks separately.

- **Sign+Norm** $Q(\mathbf{x}) = \text{sign}(\mathbf{x}) \cdot \frac{\|\mathbf{x}\|_1}{\text{length}(\mathbf{x})}$. We confirm the author's observations that this compressor gives the best and most reliable results.

- **top-1%** Let $p_{99}(\mathbf{x})$ represent the 99th percentile of coordinates in $\mathbf{x}$ by absolute value. Here

$$Q(\mathbf{x})_i = \mathbf{x}_i \text{ if } \mathbf{x}_i \geq p_{99}(\mathbf{x}), 0 \text{ otherwise.}$$

  To communicate the top 1% of a vector, we communicate 32-bit float values and 64-bit integer indices, following the authors.

- **SVD** This low-rank compressor has not been used with ChocoSGD, but we have evaluated it because our proposed method is also based on low-rank compression. This compressor represents a matrix X by $(X\mathbf{v})\mathbf{v}^\top$, where $\mathbf{v}$ is the (normalized) top right singular vector found by a Singular Value Decomposition (SVD).

H.2 DeepSqueeze

We implement DeepSqueeze according to Algorithm 1 in (Tang et al., 2019), and use the same compressors described for ChocoSGD above.

H.3 Moniqua

Because the 1-bit version of Moniqua (Lu & Sa, 2020) is derived from the 2-bit version with added BZIP compression, we focus on the 2-bit version. We implement the algorithm according to Algorithm 1 in (Lu & Sa, 2020). We use the same step size schedule $\{\alpha_k\}$ as for the optimizers we evaluated, and tune the a priori bound θ as a gobal constant, as suggested by the authors. As a stochastic rounding operator $\mathcal{Q}$, we quantize stochastically in an unbiased fashion to the points $\{-\frac{1}{2}, -\frac{1}{6}, \frac{1}{6}, \frac{1}{2}\}$. This yields $\delta = \frac{1}{3}$. Note that the modulo operator 'mod B_θ' in the algorithm yields values between $-\frac{1}{2}B_\theta$ and $\frac{1}{2}B_\theta$.

I Parameters in architectures

See Table 6 and Table 7 for an overview of parameters in the models used.

Table 6: Parameters in the ResNet20 architecture and their shapes. The table shows the per-tensor compression ratio achieved by rank-1 PowerGossip with r iterations.

Parameter	Parameter shape	Matrix shape	Uncompressed	Compression
layer3.1.conv1	$64 \times 64 \times 3 \times 3$	64×576	144 KB	$115/r \times$
layer3.2.conv1	$64 \times 64 \times 3 \times 3$	64×576	144 KB	$115/r \times$
layer3.0.conv2	$64 \times 64 \times 3 \times 3$	64×576	144 KB	$115/r \times$
layer3.1.conv2	$64 \times 64 \times 3 \times 3$	64×576	144 KB	$115/r \times$
layer3.2.conv2	$64 \times 64 \times 3 \times 3$	64×576	144 KB	$115/r \times$
layer3.0.conv1	$64 \times 32 \times 3 \times 3$	64×288	72 KB	$105/r \times$
layer2.2.conv2	$32 \times 32 \times 3 \times 3$	32×288	36 KB	$58/r \times$
layer2.1.conv1	$32 \times 32 \times 3 \times 3$	32×288	36 KB	$58/r \times$
layer2.0.conv2	$32 \times 32 \times 3 \times 3$	32×288	36 KB	$58/r \times$
layer2.1.conv2	$32 \times 32 \times 3 \times 3$	32×288	36 KB	$58/r \times$
layer2.2.conv1	$32 \times 32 \times 3 \times 3$	32×288	36 KB	$58/r \times$
layer2.0.conv1	$32 \times 16 \times 3 \times 3$	32×144	18 KB	$52/r \times$
layer1.1.conv1	$16 \times 16 \times 3 \times 3$	16×144	9 KB	$29/r \times$
layer1.1.conv2	$16 \times 16 \times 3 \times 3$	16×144	9 KB	$29/r \times$
layer1.0.conv2	$16 \times 16 \times 3 \times 3$	16×144	9 KB	$29/r \times$
layer1.2.conv1	$16 \times 16 \times 3 \times 3$	16×144	9 KB	$29/r \times$
layer1.0.conv1	$16 \times 16 \times 3 \times 3$	16×144	9 KB	$29/r \times$
layer1.2.conv2	$16 \times 16 \times 3 \times 3$	16×144	9 KB	$29/r \times$
layer3.0.downsample.0	$64 \times 32 \times 1 \times 1$	64×32	8 KB	$43/r \times$
fc	10×64	10×64	2 KB	$17/r \times$
layer2.0.downsample.0	$32 \times 16 \times 1 \times 1$	32×16	2 KB	$21/r \times$
conv1	$16 \times 3 \times 3 \times 3$	16×27	2 KB	$20/r \times$
Bias vectors (total)			6 KB	None

Table 7: Parameters in the LSTM architecture and their shapes. The table shows the per-tensor compression ratio achieved by rank-1 PowerGossip with r iterations.

Parameter	Parameter shape	Matrix shape	Uncompressed	Compression
encoder	28869×650	28869×650	73300 KB	$1271/r \times$
rnn-ih-l0	2600×650	2600×650	6602 KB	$1040/r \times$
rnn-hh-l0	2600×650	2600×650	6602 KB	$1040/r \times$
rnn-ih-l1	2600×650	2600×650	6602 KB	$1040/r \times$
rnn-hh-l1	2600×650	2600×650	6602 KB	$1040/r \times$
rnn-ih-l2	2600×650	2600×650	6602 KB	$1040/r \times$
rnn-hh-l2	2600×650	2600×650	6602 KB	$1040/r \times$
Bias vectors (total)			174 KB	None

J Experiment runtime and compute infrastructure

We have executed our deep learning experiments on Nvidia Tesla K80 GPUs on `n1`-series virtual machines on Google Cloud. The algorithms were implemented in PyTorch, and run using a custom build that includes MPI for decentralized communication. We refer to the supplemental code for additional details on our runtime environment.

For our LSTM experiments with 16 workers, we use 4 GPUs with 4 processes per GPU. The experiments took approximately 4 hours in this setup.

For our Cifar-10 experiments with 8 workers, use 2 GPUs with 4 processes each. Those experiments took around 1.5 hours.