

# Communication-efficient distributed training of machine learning models

Présentée le 11 avril 2023

Faculté informatique et communications  
Laboratoire d'apprentissage automatique et d'optimisation  
Programme doctoral en informatique et communications

pour l'obtention du grade de Docteur ès Sciences

par

## Thijs VOGELS

Acceptée sur proposition du jury

Prof. A.-M. Kermarrec, présidente du jury  
Prof. M. Jaggi, directeur de thèse  
Prof. M. Rabbat, rapporteur  
Prof. D. Alistarh, rapporteur  
Prof. P. Thiran, rapporteur



# Acknowledgements

What a privilege, when learning is your job.

I ended up with that job mainly thanks to my environment growing up. At my parents' home, learning was considered something fun. Practicing multiplications became a game my dad and I played in the bath tub. At school, I was surrounded by friends like Tobias, Willem, Wim and Rens, who still considered learning to be fun—a rare belief amongst teenagers—and by passionate teachers like Jeroen, Quintijn, Henk and Richard, who fed my passion for learning and research with their enthusiasm for mathematical puzzles and computer science.

MLO has been a fantastic working place. Martin has created an exceptionally open and friendly environment for everyone to work. His optimistic down-to-earth attitude has shielded me from much of the stress of academia. Martin's mentorship has provided direction to my research, while allowing for great flexibility to experiment and fail. I am very grateful for the opportunity to work with him.

At MLO, I have been lucky to work with exceptionally skillful collaborators—Anastasia, Annie, Cécile, Daniel, Lie, Omar, Sebastian, Tao—and with two incredible mentors—Praneeth and Hadrien. Working closely with them is the thing I enjoyed most during my PhD time. Together, we translated vague thoughts into concrete ideas, and then into the papers that are featured in this thesis.

During internships at NVIDIA and Google, I was hosted by Jan, Fabrice, Sean, and Florian. It was amazing to see and learn from the energy and the enthusiasm they put into their work. Fabrice's opinions about 'what makes good code' currently form the basis of my beliefs on the topic, and Jan's attention to detail for the presentation of research results inspires me whenever I make figures.

Many of the people that I met on the top floor of INJ have become close friends. Frequent game nights with Anastasia and Maksym, long walks and bike trips with Praneeth, and recycling an IKEA book-case into custom shelving with Jean-Baptiste are some of the things that I will remember most fondly. Jean-Baptiste also deserves an additional acknowledgement for having the courage to alpha-test and to be the single current user of SlideKit.

Throughout my time in Lausanne, even though it clearly was nice to spend time at MLO, I have always looked forward to coming back home to see Emma. She has been unconditionally encouraging, and exploring Switzerland with her has been the true highlight of my time here.

Finally, my sister Renee deserves to be acknowledged for the most careful proofreading of most of the papers in this thesis.

Thank you all contributing to my time in Lausanne, and to the great memories I will take with me wherever I go.



# Abstract

In this thesis, we explore techniques for addressing the communication bottleneck in data-parallel distributed training of deep learning models. We investigate algorithms that either reduce the size of the messages that are exchanged between workers, or that reduce the number of messages sent and received.

To reduce the *size* of messages, we propose an algorithm for lossy compression of gradients. This algorithm is compatible with existing high-performance training pipelines based on the all-reduce primitive and leverages the natural approximate low-rank structure in gradients of neural network layers to obtain high compression rates.

To reduce the *number* of messages, we study the decentralized learning paradigm where workers do not average their model updates all-to-all in each step of Stochastic Gradient Descent, but only communicate with a small subset of their peers. We extend the aforementioned compression algorithm to operate in this setting. We also study the influence of the communication topology on the performance of decentralized learning, highlighting shortcomings of the typical ‘spectral gap’ metric to measure the quality of communication topologies, and proposing a new framework for evaluating topologies. Finally, we propose an alternative communication paradigm for distributed learning over sparse topologies. This paradigm, which is based on the concept ‘relaying’ updates over spanning trees of the communication topology, shows benefits over the typical gossip-based approach, especially when the workers have very heterogeneous data distributions.

**Keywords** Deep learning, machine learning, distributed training, decentralized learning, gradient compression, stochastic gradient descent.



# Résumé

Dans cette thèse, nous explorons des techniques pour résoudre les problèmes de goulot d'étranglement des communications qui surviennent lors de l'entraînement des modèles d'apprentissage profond lorsque les données sont traitées de manière parallèle et distribuée. Nous étudions des algorithmes qui réduisent la taille des messages échangés entre les nœuds de calcul ou bien qui réduisent le nombre de messages envoyés et reçus. Nous étudions les algorithmes qui réduisent la taille des messages échangés entre les nœuds de calcul ou qui réduisent le nombre de messages envoyés et reçus.

Pour réduire la *taille* des messages, nous proposons un algorithme de compression non-exacte des gradients. Cet algorithme est compatible avec les pipelines déjà existantes d'entraînement haute performance basées sur la "all-reduce" primitive et exploite la structure approximative de rang faible occurring naturellement dans les gradients des couches du réseau neuronal afin d'obtenir des taux de compression élevés.

Pour réduire le *nombre* de messages, nous étudions le paradigme d'apprentissage décentralisé où les travailleurs ne font pas la moyenne de leurs mises à jour de modèle à chaque itération de la descente de gradient stochastique, mais ne communiquent qu'avec un petit sous-ensemble de leurs pairs. Nous étendons l'algorithme de compression susmentionné pour qu'il fonctionne dans ce cadre. Nous étudions également l'influence de la topologie de communication sur les performances de l'apprentissage décentralisé afin de mettre en évidence les lacunes de la métrique classique du « trou spectral » qui mesure la qualité des topologies de communication et nous proposons un nouveau cadre d'évaluation des topologies. Enfin, nous proposons un paradigme de communication alternatif pour l'apprentissage distribué sur des topologies parsimonieuses. Ce paradigme, qui est basé sur le concept de « relais » des mises à jour sur des arbres couvrants de la topologie de communication, présente des avantages par rapport à l'approche classique basée de "bavardages", en particulier lorsque les travailleurs ont des distributions de données très hétérogènes.

**Mots-clés** Apprentissage profond, apprentissage automatique, apprentissage distribuée, apprentissage décentralisé, compression de gradient, algorithme du gradient stochastique.





# Contents

<b>Acknowledgements</b>	<b>iii</b>
<b>Abstract (English / Français)</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
1.0.1 Outline of the thesis . . . . .	2
1.0.2 Contributions beyond this thesis . . . . .	3
<b>2 Practical low-rank gradient compression</b>	<b>5</b>
2.1 Preface . . . . .	5
2.2 Introduction . . . . .	5
2.3 Related work . . . . .	6
2.4 Follow-up work . . . . .	8
2.5 Method . . . . .	9
2.6 Analysis of PowerSGD . . . . .	10
2.6.1 Effect of error feedback . . . . .	11
2.6.2 Effect of warm-start . . . . .	11
2.6.3 Effect of varying the rank . . . . .	12
2.7 Results . . . . .	13
2.7.1 Comparison with other compressors . . . . .	13
2.7.2 Scalability of PowerSGD . . . . .	14
2.7.3 Beneficial regularization . . . . .	15
2.7.4 Other tasks and methods . . . . .	15
2.8 Conclusion . . . . .	16
2.9 Acknowledgements . . . . .	16
<b>3 Low-rank gradient compression for decentralized learning</b>	<b>17</b>
3.1 Preface . . . . .	17
3.2 Introduction . . . . .	17
3.3 Related work . . . . .	18
3.4 Decentralized machine learning . . . . .	19
3.5 Algorithm . . . . .	20
3.5.1 Properties . . . . .	22
3.6 Theoretical analysis . . . . .	22
3.6.1 Assumptions and setup . . . . .	22
3.6.2 Convergence rates . . . . .	23
3.7 Experimental analysis . . . . .	25

3.8	Conclusion . . . . .	27
3.9	Acknowledgements . . . . .	27
<b>4</b>	<b>The role of the topology in decentralized learning</b>	<b>29</b>
4.1	Preface . . . . .	29
4.2	Introduction . . . . .	29
4.3	Related work . . . . .	31
4.4	A toy problem: D-SGD on isotropic random quadratics . . . . .	33
4.5	Theoretical analysis . . . . .	34
4.6	Experimental analysis . . . . .	38
4.7	Conclusion . . . . .	40
4.8	Acknowledgements . . . . .	40
<b>5</b>	<b>A relay mechanism for decentralized learning with heterogeneous data</b>	<b>41</b>
5.1	Preface . . . . .	41
5.2	Introduction . . . . .	42
5.3	Related work . . . . .	43
5.4	Method . . . . .	44
5.5	Theoretical analysis . . . . .	46
5.6	Experimental analysis and practical properties . . . . .	48
5.6.1	Effect of network topology . . . . .	48
5.6.2	Spanning trees compared to other topologies . . . . .	49
5.6.3	Effect of data heterogeneity in decentralized deep learning . . . . .	49
5.6.4	Robustness to unreliable communication . . . . .	51
5.7	Conclusion . . . . .	53
5.7.1	Applicability to data center training . . . . .	53
5.8	Acknowledgements . . . . .	53
<b>6</b>	<b>Conclusion</b>	<b>55</b>
6.1	Discussion and future work . . . . .	55
<b>A</b>	<b>Appendix for PowerSGD</b>	<b>57</b>
A.1	Discussion of convergence . . . . .	57
A.1.1	Eigen compression . . . . .	57
A.1.2	Subspace iteration . . . . .	58
A.1.3	Single/multi worker equivalence . . . . .	60
A.2	Cluster specifications . . . . .	61
A.3	Convergence curves . . . . .	62
A.4	Language modeling with transformers . . . . .	64
A.5	The need for error feedback . . . . .	66
A.6	Network parameters . . . . .	67
A.7	Compressor implementation details . . . . .	68
A.7.1	Random Block . . . . .	68
A.7.2	Random K . . . . .	68
A.7.3	Sign+Norm . . . . .	69
A.7.4	Top K . . . . .	69
A.7.5	Signum . . . . .	70
A.7.6	Atomo . . . . .	70

A.7.7	Best-approximation PowerSGD . . . . .	71
A.8	Performance optimizations . . . . .	72
A.9	Learning rate tuning . . . . .	72
<b>B</b>	<b>Appendix for PowerGossip</b>	<b>73</b>
B.1	Compressed Consensus . . . . .	73
B.2	Compressed optimization . . . . .	75
B.3	Experimental settings . . . . .	75
B.4	Convergence curves . . . . .	75
B.4.1	ResNet-20 on Cifar-10 . . . . .	76
B.4.2	LSTM on WikiText-2 . . . . .	76
B.5	The power spectrum of parameter differences . . . . .	77
B.5.1	LSTM Training . . . . .	77
B.5.2	Consensus . . . . .	78
B.6	Changing rank vs changing # power iterations . . . . .	78
B.7	Hyperparameters . . . . .	79
B.7.1	Consensus . . . . .	79
B.7.2	ResNet-20 on Cifar-10 . . . . .	79
B.7.3	LSTM on WikiText-2 . . . . .	80
B.8	Compared-to algorithm implementations . . . . .	80
B.8.1	ChocoSGD . . . . .	80
B.8.2	DeepSqueeze . . . . .	80
B.8.3	Moniqua . . . . .	80
B.9	Parameters in architectures . . . . .	81
B.10	Experiment runtime and compute infrastructure . . . . .	82
<b>C</b>	<b>Appendix for Beyond Spectral gap</b>	<b>83</b>
C.1	Notation . . . . .	83
C.2	Topologies . . . . .	84
C.3	Random quadratics . . . . .	86
C.3.1	Objective . . . . .	86
C.3.2	Algorithm . . . . .	86
C.3.3	Linear convergence of an unrolled error vector . . . . .	87
C.3.4	Random walks with gossip averaging . . . . .	88
C.3.5	Converging random walk . . . . .	89
C.3.6	The rate for D-SGD . . . . .	91
C.4	(Strongly)-Convex case, missing proofs and additional results . . . . .	91
C.4.1	Preliminaries on Bregman divergences . . . . .	91
C.4.2	Main result . . . . .	92
C.4.3	Obtaining Corollary IV . . . . .	97
C.4.4	Deterministic algorithm . . . . .	97
C.5	Cifar-10 experimental setup . . . . .	99
C.6	Additional experiments . . . . .	100
C.6.1	Results on Fashion MNIST . . . . .	101
C.6.2	Heterogeneous data . . . . .	101
C.6.3	The role of $\gamma$ in the experiments . . . . .	102

<b>D Appendix for RelaySGD</b>	<b>107</b>
D.1 Convergence Analysis of RelaySGD . . . . .	107
D.2 Detailed experimental setup . . . . .	107
D.2.1 Cifar-10 . . . . .	107
D.2.2 ImageNet . . . . .	107
D.2.3 BERT finetuning . . . . .	108
D.2.4 Random quadratics . . . . .	108
D.3 Hyperparameters and tuning details . . . . .	109
D.3.1 Cifar-10 . . . . .	109
D.3.2 ImageNet . . . . .	110
D.3.3 BERT finetuning . . . . .	110
D.3.4 Random quadratics . . . . .	111
D.4 Algorithmic details . . . . .	111
D.4.1 Learning-rate correction for RelaySGD . . . . .	111
D.4.2 RelaySGD with momentum . . . . .	111
D.4.3 RelaySGD with Adam . . . . .	112
D.4.4 D2 with momentum . . . . .	112
D.4.5 Gradient Tracking . . . . .	112
D.4.6 Stochastic Gradient Push with the time-varying exponential topology	112
D.5 Additional experiments on RelaySGD . . . . .	113
D.5.1 Rings vs double binary trees on Cifar-10 . . . . .	113
D.5.2 Scaling the number of workers on Cifar-10 . . . . .	113
D.5.3 Independence of heterogeneity . . . . .	114
D.5.4 Star topology . . . . .	114
D.6 RelaySum for distributed mean estimation . . . . .	115
D.7 Alternative optimizer based on RelaySum . . . . .	115
D.7.1 Empirical analysis of RelaySGD/Grad . . . . .	119
<b>Bibliography</b>	<b>119</b>
<b>Curriculum Vitae</b>	<b>131</b>

# Chapter 1

## Introduction

The progress made in deep learning in the last few years has been remarkable. Current large language models are starting to be practical assistants. They can help us to write code and text [Brown et al., 2020], or help us to generate and edit images using text prompts [Ramesh et al., 2021]. At the same time, deep learning models are finding their way into science and engineering. Most notably, AlphaFold has made great progress in protein structure prediction [Jumper et al., 2021]. These, and many other recent successes, rely heavily on scale: they require both large models and large datasets. Attached to this scale is a cost: training these models takes significant *energy* and *time*.

While reducing the energy consumption of training is perhaps the most pressing challenge in deep learning right now, it is not the main focus of this thesis. Instead, we focus on reducing the *time* it takes to train these models. First and foremost, fast training improves the productivity of machine learning researchers and practitioners. It also reduces the latency between collecting new data and releasing a model trained on this data. The primary strategy to achieve fast training, which is used by each of the aforementioned successes, is data parallelism. Multiple compute devices look at different parts of the dataset, and compute updates to the model in parallel.

As we increase the data parallelism to speed up training, we hit two key bottlenecks. The first bottleneck is fundamental to the Stochastic Gradient Descent (SGD) algorithm used to train deep learning models. In SGD, parallelism helps to reduce the variance of model updates by computing gradients on larger batches of training data. If the noise is high enough, reducing it enables larger step sizes, and thus faster training. If the noise is low, however, stochastic noise is no longer the bottleneck that limits the step size and the training speed. What is more, practitioners find that training with high parallelism (large batch sizes) can result in worse accuracy on data outside the training set (generalization) [Goyal et al., 2017]. A second bottleneck in the scalability of data parallel training is *communication*. Communication is required to exchange the model parameters or the gradients between the devices, and if the number of workers is large, the time spent communicating can outweigh the time saved on computation and data loading, forming a bottleneck.

In this thesis, we focus on solutions to the communication bottleneck in data parallel training. The typical solution to this bottleneck is to use expensive networking infrastructure to increase the available bandwidth [Markov et al., 2021]. This solution requires little effort on the part of the practitioners, but the cost can be prohibitive. Another option is to reduce the communication, either by making messages smaller, or by reducing their number. We could reduce the *size* of messages by using small parameter-efficient models or by compressing the

messages. The *number* of messages could be reduced by increasing local batch sizes [Goyal et al., 2017], taking local steps [Stich, 2019] before communicating, or by replacing all-to-all communication by sparser node-to-node synchronization [Lian et al., 2017].

When we use lossy compression to reduce the size of the gradients that are averaged between workers after each gradient computation, we should be careful about two things. Firstly, it is important to preserve enough gradient information. The error feedback mechanism [Seide et al., 2014, Cordonnier, 2018, Stich et al., 2018, Karimireddy et al., 2019b] is a way to ensure this. This mechanism uses memory local to each worker to store compression errors, and compensate for those errors in future steps to avoid losing the information. A second requirement is that the compression should be fast enough to not become a bottleneck itself [Agarwal et al., 2022, Markov et al., 2021]. The gain in data transfer time should always be larger than the cost of compression.

### 1.0.1 Outline of the thesis

Chapter 2 introduces the PowerSGD compression scheme [Vogels et al., 2019]. PowerSGD was designed with the above constraints in mind. By leveraging natural low-rank structure in the gradients of deep learning models, PowerSGD supports high compression rates while being relatively efficient on today’s GPUs and easy to integrate with current optimized parallel training code based on the all-reduce communication pattern. Instead of averaging gradients across workers, PowerSGD computes a low-rank matrix that *approximates* the average worker gradient. Crucially, this low-rank approximation is not computed using an expensive Singular Value Decomposition, but it is computed using a process akin to power iteration that only relies on cheap matrix multiplications and orthogonalization operations.

Apart from sending smaller messages between workers, another approach to alleviating the communication bottleneck can be to reduce the number of messages. Such sparsity in the communication patterns is naturally supported by the *decentralized learning* paradigm [Lian et al., 2017], which avoids the need for synchronizing models between all workers at the start of each iteration. While there are various forms of decentralization in machine learning [Lu and De Sa, 2021], we focus on the setting where workers only communicate with few others in each gradient computation round, resulting in inconsistent models across workers. As long as those inconsistencies are kept small enough, they do not harm the convergence of the training process. Most of the research in decentralized learning assumes that a particular communication topology between the nodes is given. We are interested in using this paradigm mainly for reducing the communication in data parallel training, and therefore we typically assume that we can choose the communication topology ourselves.

Chapter 3 extends the PowerSGD compression scheme to the decentralized learning setting [Vogels et al., 2020]. PowerSGD is closely tied to the all-reduce pattern, but here we generalize the scheme to be a general mechanism for synchronizing models between workers using any connectivity. The resulting PowerGossip algorithm carries the same benefits of PowerSGD, but additionally, it avoids the need for compression-specific hyperparameters that other compression methods for the decentralized learning require.

If we use decentralized learning purely for communication-efficiency, and if we assume that we can choose the communication topology ourselves, we need to be able to reason about how good a topology is. Chapter 4 explores this question, following [Vogels et al., 2022]. We find that the ‘spectral gap’, the most popular metric used to capture the effect of a graph topology in decentralized learning, does not correlate well with empirical performance. It

also does not naturally extend to time-varying graphs, which are popular in practice [Assran et al., 2019]. In this chapter, we improve our understanding of how a topology influences convergence, both theoretically, and empirically in deep learning, and we introduce a notion of ‘effective number of neighbors’ that links decentralized performance to the performance under all-to-all communication.

In chapter 5, we investigate an alternative communication mechanism to gossip communication for decentralized training. The typical gossip-based D-SGD algorithm [Lian et al., 2017] does not work well if the different workers have data sets with different distributions. This is because the workers receive more ‘influence’ from others close by in the communication network, and less from others that are far away. The RelaySum mechanism [Vogels et al., 2021] that we discuss in this chapter replaces gossip communication by a form of ‘re-laying’ or ‘forwarding’ of messages between workers. While relayed messages may arrive with delays, workers receive exactly one update from everyone in the network at each step. This means that, in some way, all peers have the same ‘influence’ on each other. We find that this mechanism can be efficiently implemented on spanning trees, resulting in the same number of messages sent and received as in typical gossip communication, while outperforming gossip-based baselines on deep learning experiments with non-iid data distributions.

### 1.0.2 Contributions beyond this thesis

The chapters in this thesis are a selection of work that the author contributed to during his Ph.D. and that are related to communication efficiency in distributed learning. The author also contributed to projects on other topics.

In [Sivaprasad et al., 2020], we observe that the efficacy of new optimizers in deep learning is usually demonstrated under near-optimal hyperparameters. Finding those parameters in practice can be infeasible for practitioners outside of research. Therefore, we introduce a framework to evaluate optimizers that takes the cost of hyperparameter tuning into account.

In [Zhang et al., 2021], we study using machine learning for denoising Monte Carlo renderings. We introduce a modification to the commonly used kernel-predicting model architecture that improves denoised image quality.

In [Trottet et al., 2022], we present a model architecture tailored to questionnaire data collected for clinical decision support systems. The model works well with the structured missing data that is common in decision-tree based questionnaire data.





## Chapter 2

# Practical low-rank gradient compression

### 2.1 Preface

This chapter follows [Vogels et al., 2019], with minor edits. This paper was written in 2019, and the PowerSGD algorithm has since been adopted in practice and improved by others. The new Section 2.4 discusses follow-up work that was published after the original paper.

**Summary** We study lossy gradient compression methods to alleviate the communication bottleneck in data-parallel distributed optimization. Despite the significant attention received, current compression schemes either do not scale well, or fail to achieve the target test accuracy. We propose a low-rank gradient compressor based on power iteration that can i) compress gradients rapidly, ii) efficiently aggregate the compressed gradients using all-reduce, and iii) achieve test performance on par with SGD. The proposed algorithm is the only method evaluated that achieves consistent wall-clock speedups when benchmarked against regular SGD using highly optimized off-the-shelf tools for distributed communication. We demonstrate reduced training times for convolutional networks as well as LSTMs on common datasets.

**Code** <https://github.com/epfml/powersgd>

**Co-authors** Sai Praneeth Karimireddy and Martin Jaggi.

#### Contributions

T. Vogels: methodology (50%), software, visualization, writing (70%).

S.P. Karimireddy: methodology (50%), formal analysis, writing (30%).

M. Jaggi: writing – review and editing, project administration, supervision.

### 2.2 Introduction

Synchronous data-parallel SGD is the most common method for accelerating training of deep learning models [Dean et al., 2012, Iandola et al., 2016, Goyal et al., 2017]. Because the gradient vectors of such models can be large, the time required to share those gradients across workers limits the scalability of deep learning training [Seide et al., 2014, Iandola et al., 2016, Lin et al., 2018].

Previous work proposes lossy gradient compression as a solution to this issue. Notable examples include replacing the coordinates of the gradient with only their sign [Seide et al., 2014, Carlson et al., 2015, Bernstein et al., 2018, 2019, Karimireddy et al., 2019b], quantizing the

individual coordinates [Alistarh et al., 2017, Wen et al., 2017], and low-rank approximation of the gradient [Wang et al., 2018]. While these works demonstrate speedups over full-precision SGD in some settings, we find that their speedups vanish with a fast network and highly optimized communication backend, even on commodity hardware. Some prior work also suffers from degraded test accuracy compared to SGD. We combine three observations to fix these issues: i) Linear compressor operators achieve scalability by enabling aggregation using all-reduce. ii) Error feedback ensures convergence with general biased compressors. iii) Low-rank updates enable aggressive compression without sacrificing quality.

First, we explore the properties of various gradient compression schemes for SGD and identify which ones are crucial for high scalability. In particular, we note that currently proposed gradient compressors are not linear. Their compressed messages cannot be added up directly, unlike raw gradients. This prevents current compressed SGD algorithms from aggregating gradients using an efficient *reduce* operation and instead require a *gather* operation. Current deep learning frameworks rely either solely or predominantly on all-reduce, which is key to why regular SGD scales well with fast communication hardware [cf. Awan et al., 2018, Panda et al., 2019].

Secondly, it was recently shown that using error feedback, i.e., storing the difference between the computed and compressed gradient, and reinserting it at the next iteration, improves both convergence and generalization for compression schemes [Karimireddy et al., 2019b]. This enables the use of general biased gradient compression schemes.

Thirdly, there is growing evidence that the generalization ability of over-parameterized deep learning models is related to low-rankness [Arora et al., 2018, Martin and Mahoney, 2018, Collins et al., 2018]. Using a low-rank update (as we do) can be viewed as implicitly performing spectral regularization [Gunasekar et al., 2018] and hence can be expected to have good generalization properties [Yoshida and Miyato, 2017]. Further, Wang et al. [2018] show that the eigenspectrum of the stochastic gradients for deep learning models decays, suggesting that a rank-based schemes can get away with aggressive compression without sacrificing convergence.

In this work, we design PowerSGD with the above observations in mind. PowerSGD computes a low-rank approximation of the gradient using a generalized *power* iteration (known as subspace iteration [Stewart and Miller, 1975]). The approximation is computationally lightweight, avoiding any prohibitively expensive Singular Value Decomposition. To improve the quality of the efficient approximation, we *warm-start* the power iteration by reusing the approximation from the previous optimization step. Using all-reduce gradient aggregation, we empirically demonstrate that PowerSGD achieves wall-clock speedups over regular SGD in a 16-GPU setting, even with the optimized NCCL communication backend on a fast network (and is the only algorithm to do so.) By compressing gradients more than  $120\times$ , we reduce communication time (including coding and decoding) by 54% for ResNet-18 on Cifar-10 and by 90% for an LSTM on Wikitext-2. End-to-end wall-clock training time to full test quality is reduced by 24% for ResNet-18 and by 55% for the LSTM.

## 2.3 Related work

**Gradient compression** A variety of compression schemes (fig. 2.1) have been proposed: Alistarh et al. [2017] and Wen et al. [2017] quantize each gradient coordinate; Seide et al. [2014], Carlson et al. [2015], Bernstein et al. [2018, 2019] and Karimireddy et al. [2019b] replace each

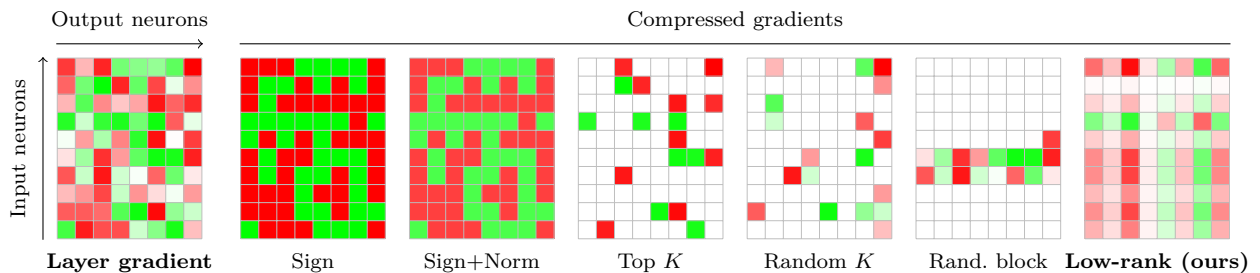


Figure 2.1: Compression schemes compared in this paper. Left: Interpretation of a layer’s gradient as a matrix. Coordinate values are color coded (**positive**, **negative**). Right: The output of various compression schemes on the same input. Implementation details are in Appendix A.7.

coordinate of the gradient with its sign; Lin et al. [2018], Stich et al. [2018] and Wangni et al. [2018] use the largest few coordinates; and Konečný et al. [2016] and Wang et al. [2018] use a low-rank approximation.

Spectral Atomo [Wang et al., 2018] is perhaps closest to our work. It importance-samples the gradient’s singular vectors and is an unbiased compression scheme. It requires, however, a full Singular Value Decomposition every iteration and is hence computationally impractical.

**Commutative compression and addition** Yu et al. [2018] stress that commutability of compression with gradient addition enables efficient aggregation with *ring all-reduce*. Most of the compressors, however, lack this property. Yu et al. utilize temporally-consistent correlations between gradients coordinates to compress them linearly. PowerSGD has a similar property that we call ‘linearity’.

**Error feedback** First introduced in [Seide et al., 2014] and analyzed in [Stich et al., 2018, Cordonnier, 2018] for the convex case, error feedback involves computing the difference between a worker’s gradient and the compressed gradient (*error*) and adding it back to the next gradient (*feedback*). Karimireddy et al. [2019b] and Stich and Karimireddy [2019] further develop and generalize the framework of error feedback with improved rates. In the non-convex setting, Karimireddy et al. [2019b] show that error feedback is crucial both for convergence and generalization when using biased compressors, e.g., sign or top- $K$ . In general, biased compression schemes equipped with error feedback tend to out-perform their unbiased counterparts. The practical algorithm by Lin et al. [2018] is also as an approximate top- $K$  compressor with error feedback.

**Low-rank methods** Recent works argue that in modern over-parameterized deep networks, the final model learned has a ‘low stable rank’ [Martin and Mahoney, 2018, Li et al., 2018]. This can partially explain their impressive generalization properties despite being substantially overparameterized [Arora et al., 2018]. Adding explicit spectral regularization has shown to further improve the performance of such models [Mazumder et al., 2010, Yoshida and Miyato, 2017]. Using a low-rank update (as we do) can be viewed as implicitly performing a similar regularization [Gunasekar et al., 2018]. If the target matrices are known to be exactly low-ranked (instead of just low stable rank), Yurtsever et al. [2017] show that it is sometimes possible to converge to the optima using low rank approximations of the gradients without the need for error feedback.

## 2.4 Follow-up work

**Variations and extensions** While PowerSGD applies low-rank approximations to gradients over the course of training, it is also possible to apply low-rank approximation to the model. The Pufferfish algorithm [Wang et al., 2021a] does this after using PowerSGD for a warm-up period of several epochs, and Yu et al. [2021] use this kind of compression for differentially private learning. Such decompositions can have additional benefits over reduced communication, such as cheaper computation [Wang et al., 2021a], or improved privacy trade-offs [Yu et al., 2021]. It was also found to be beneficial to use adaptive compression rates for PowerSGD. Agarwal et al. [2021] propose to use a lower compression rate in the beginning of training than in the end, and Alimohammadi et al. [2022] demonstrate that layer-wise adaptive compression rates can benefit the algorithm.

In an effort to combine PowerSGD compression with decentralized averaging, Vogels et al. [2020] (chapter 3) also introduce several algorithmic improvements to PowerSGD. These include a more flexible approach to choosing the number of power iteration steps per SGD step, providing the user with a trade-off over communication latency and orthogonalization time. If orthogonalization is a bottleneck, the user can use multiple communication rounds with a smaller rank. If latency is a bottleneck, the user can use a single communication round per SGD step instead of two.

Finally, the error-feedback mechanism in this paper has remained an active area of research, and many alternatives have been proposed [Xie et al., 2020, Richtárik et al., 2021, Xu and Huang, 2022, Horváth and Richtárik, 2021].

**Benchmarks and the need for speed** Several authors have benchmarked PowerSGD and other compression schemes on various datasets and models. Xu et al. [2021] provide a library with unified optimized implementations of many algorithms. Agarwal et al. [2022] make a clear case that, under typical data center bandwidths, the time required for compression can outweigh the savings in communication time. Markov et al. [2021] further identify that PowerSGD is less effective on transformers compared to CNNs, requiring a high compression rank to obtain full accuracy. In general, the consensus is that the focus of the community is better spent optimizing compression speed than further increasing compression rates over PowerSGD [Agarwal et al., 2022, Markov et al., 2021].

**Promising compression techniques** Since the publication of PowerSGD, several new compression algorithms have been proposed. In particular, Chen et al. [2020] and Shi et al. [2021] show that slight modifications to top- $k$  compression can be both computationally efficient and compatible with all-reduce. Sketching methods are also viable compressors, and due to their linearity, they are also compatible with all-reduce [Ge et al., 2022, Ivkin et al., 2019]. Finally, Markov et al. [2021] show that, through low-level programming efforts, quantization can be made very fast and practical.

**Applications outside academia** PowerSGD was implemented in PyTorch as a communication hook for the Distributed Data Parallel API [Contributors, 2020]. It was used with to speed up training of language models [Wang et al., 2021b], and it was used to train DALL-E [Ramesh et al., 2021]. Ramesh et al. [2021] also show that PowerSGD benefits from a fast CUDA implementation of the orthogonalization procedure, and they provide practical guidelines for dealing with low-precision (sub 32-bit) floating point numbers.

## 2.5 Method

In data-parallel optimization of machine learning models, a number of  $W$  workers share the same model parameters  $\mathbf{x} \in \mathbb{R}^d$ . They iteratively update  $\mathbf{x}$  by computing independent stochastic gradients, aggregating these gradients by averaging<sup>1</sup>, and updating the model parameters based on this aggregate.

---

**Algorithm 1** Rank- $r$  PowerSGD compression

---

```

1: The update vector  $\Delta_w$  is treated as a list of tensors corresponding to individual model
   parameters. Vector-shaped parameters (biases) are aggregated uncompressed. Other
   parameters are reshaped into matrices. The functions below operate on such matrices
   independently. For each matrix  $M \in \mathbb{R}^{n \times m}$ , a corresponding  $Q \in \mathbb{R}^{m \times r}$  is initialized
   from an i.i.d. standard normal distribution.
2: function COMPRESS+AGGREGATE(update matrix  $M \in \mathbb{R}^{n \times m}$ , previous  $Q \in \mathbb{R}^{m \times r}$ )
3:    $P \leftarrow MQ$ 
4:    $P \leftarrow \text{ALL REDUCE MEAN}(P)$   $\triangleright$  Now,  $P = \frac{1}{W}(M_1 + \dots + M_W)Q$ 
5:    $\hat{P} \leftarrow \text{ORTHOGONALIZE}(P)$   $\triangleright$  Orthonormal columns
6:    $Q \leftarrow M^\top \hat{P}$ 
7:    $Q \leftarrow \text{ALL REDUCE MEAN}(Q)$   $\triangleright$  Now,  $Q = \frac{1}{W}(M_1 + \dots + M_W)^\top \hat{P}$ 
8:   return the compressed representation  $(\hat{P}, Q)$ .
9: end function
10: function DECOMPRESS( $\hat{P} \in \mathbb{R}^{n \times r}$ ,  $Q \in \mathbb{R}^{m \times r}$ )
11:   return  $\hat{P}Q^\top$ 
12: end function

```

---

**PowerSGD compression** We approximate each layer in the model independently. The parameters of fully-connected layers (dense matrix multiplication) and their gradients have an inherent matrix structure. The parameters of convolutional layers can be naturally interpreted as fully-connected layers applied repeatedly over a 2D grid of inputs. Practically, this amounts to flattening input and kernel dimensions in the 4D gradient tensors. Neural networks also contain bias vectors, but these typically constitute a tiny fraction of the parameter space and can be aggregated uncompressed.

For each parameter’s gradient  $M \in \mathbb{R}^{n \times m}$ , the aim of rank- $r$  matrix approximation is to find matrices  $P \in \mathbb{R}^{n \times r}$  and  $Q \in \mathbb{R}^{m \times r}$  such that  $PQ^\top$  approximates  $M$  well. To motivate such a representation, consider that a gradient  $M$  of a linear layer is computed as  $M = \sum_{b \in \text{batch}} \frac{\partial \text{loss}_b}{\partial \mathbf{y}_b} \mathbf{x}_b^\top$ , where  $\mathbf{x}_b$  is the input to the layer for data point  $b$  and  $\mathbf{y}_b$  is the output. In the limiting case when the batch size is 1, this gradient is exactly rank-1, and it could be represented much more efficiently than in its dense form. Even with larger batch sizes, or with convolutional or LSTM layers, low-rank structure in either the layer’s inputs or output gradients translates to a peaky power spectrum and effective low-rank compression.

PowerSGD uses a single step of subspace iteration—*power* iteration generalized to  $r > 1$ —to compute such an approximation. This involves performing one right multiplication, one left multiplication, and an orthogonalization. Chapter 3 will outline a more general variant of PowerSGD that can use only a single left or right matrix multiplication step per SGD

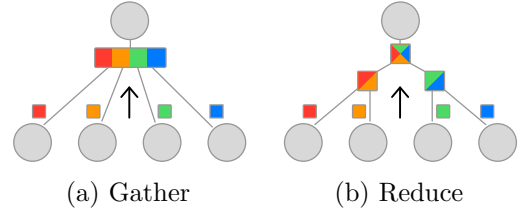
---

<sup>1</sup> Bernstein et al. [2019] propose Signum which aggregates 1-bit gradients by majority voting instead of averaging.

iteration. We use the Gram-Schmidt procedure to orthogonalize our matrices since they have very few columns (1–4), and this is the most expensive part of the compression procedure. Further, we ‘warm-start’ the subspace iteration by reusing the approximation computed at the previous step. With the inclusion of warm-start, a *single* step of subspace iteration yields a factorization  $M \sim PQ^\top$  with the same performance as the best rank- $r$  approximation from an expensive Singular Value Decomposition.

**Efficient aggregation between workers** In data-parallel optimization, we want to approximate the *average* of the worker’s gradients. Suppose PowerSGD operates on a list of corresponding gradients  $[M_1 \dots M_W]$  from  $W$  workers. Both occurrences of  $M$  in the algorithm are a (linear) matrix multiplication followed by a (linear) mean reduction over workers. This introduces a practical invariance: execution on 1 worker with batch size  $B \times W$  is equivalent to execution on  $W$  workers with batch size  $B$  each. We call this property ‘linearity’. Refer to Appendix A.1.3 for more details.

An important benefit of PowerSGD’s linearity is that it can be implemented using the **all-reduce** protocol as opposed to needing a gather operation. To illustrate the difference, suppose that we want to compute the sum of  $W$  matrices  $\sum_{i=1}^W M_i$  for  $W = 4$ . The all-reduce method can use associativity of addition to rewrite the computation as  $(M_1 + M_2) + (M_3 + M_4)$ . This enables a divide-and-conquer approach and allows the summation task to be split over multiple workers, as illustrated on the right. With  $W$  workers, both the computation and the communication timescale as  $\mathcal{O}(\log W)$  for all-reduce, compared to  $\mathcal{O}(W)$  for all-gather.



In addition to improved scaling, all-reduce communication is preferred over a parameter-server setting because it avoids *double compression*. With a parameter server, both the ‘clients  $\rightarrow$  server’ and ‘server  $\rightarrow$  clients’ communication have to be compressed [Caldas et al., 2018, Bernstein et al., 2019, Seide et al., 2014]. We avoid this by merging compression and aggregation into one step.

**Error-feedback SGD** Since the PowerSGD scheme is biased (i.e., compressing and decompressing a random gradient does not yield the original in expectation), we use error feedback [Seide et al., 2014, Karimireddy et al., 2019b]. Our version of error feedback (Algorithm 2) extends the original by introducing post-compression *momentum*. This simple extension allows us to reuse the same learning rate and hyperparameters as those tuned for SGD with momentum.

**Adaptive optimizers like Adam** We experimentally observe that PowerSGD can be used with optimizers like Adam. The recommended way is to replace line 11 of Algorithm 2 with the Adam update rule. Applying adaptive optimizers after compression and decompression ensures that the optimizer’s state remains synchronized across workers by design.

## 2.6 Analysis of PowerSGD

In this section, we consider different aspects of PowerSGD in isolation and hope to empirically understand: i) the effect of using error feedback, ii) the effect of ‘warm-start’, and iii) the trade-off between test accuracy and compression rate with varying approximation rank.

**Algorithm 2** Distributed Error-feedback SGD with Momentum

---

```

1: hyperparameters: learning rate  $\gamma$ , momentum parameter  $\lambda$ 
2: initialize model parameters  $\mathbf{x} \in \mathbb{R}^d$ , momentum  $\mathbf{m} \leftarrow \mathbf{0} \in \mathbb{R}^d$ , replicated across workers
3: at each worker  $w = 1, \dots, W$  do
4:   initialize memory  $\mathbf{e}_w \leftarrow \mathbf{0} \in \mathbb{R}^d$ 
5:   for each iterate  $t = 0, \dots$  do
6:     Compute a stochastic gradient  $\mathbf{g}_w \in \mathbb{R}^d$ .
7:      $\Delta_w \leftarrow \mathbf{g}_w + \mathbf{e}_w$  ▷ Incorporate error-feedback into update
8:      $\mathcal{C}(\Delta_w) \leftarrow \text{COMPRESS}(\Delta_w)$ 
9:      $\mathbf{e}_w \leftarrow \Delta_w - \text{DECOMPRESS}(\mathcal{C}(\Delta_w))$  ▷ Memorize local errors
10:     $\mathcal{C}(\Delta) \leftarrow \text{AGGREGATE}(\mathcal{C}(\Delta_1), \dots, \mathcal{C}(\Delta_W))$  ▷ Exchange gradients
11:     $\Delta' \leftarrow \text{DECOMPRESS}(\mathcal{C}(\Delta))$  ▷ Reconstruct an update  $\in \mathbb{R}^d$ 
12:     $\mathbf{m} \leftarrow \lambda \mathbf{m} + \Delta'$ 
13:     $\mathbf{x} \leftarrow \mathbf{x} - \gamma (\Delta' + \mathbf{m})$ 
14:   end for
15: end at

```

---

Table 2.1: Rank-based compression with and without error feedback. The biased PowerSGD outperforms an unbiased linear rank- $r$  compressor on test accuracy.





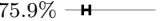



Algorithm	Test accuracy	Data/epoch
SGD	94.3% 	1023 MB
Rank-1 PowerSGD	93.6% 	4 MB
Rank-2 PowerSGD	94.4% 	8 MB
Unbiased Rank 1	71.2% 	3 MB
Unbiased Rank 2	75.9% 	4 MB

Table 2.2: Best rank-2 approximation vs. PowerSGD. Warm-start improves test accuracy, even matching the performance of the best rank-2 approximation.

Algorithm	Test accuracy
Best approximation	94.4% 
Warm start (default)	94.4% 
Without warm start	94.0% 

### 2.6.1 Effect of error feedback

Using error-feedback SGD as a base algorithm for PowerSGD has two advantages. First, it enables our use of a biased compressor. Secondly, EF-SGD improves convergence and obtains better test accuracy [Karimireddy et al., 2019b].

To illustrate the improved test accuracy, we compare PowerSGD—a biased compressor with error feedback—to an unbiased low-rank approximation. To approximate a matrix  $M \in \mathbb{R}^{n \times m}$ , the unbiased rank- $r$  approximator samples a random matrix  $U \in \mathbb{R}^{m \times r}$  such that  $\mathbb{E}[UU^\top] = I_m$  and outputs  $(MU, U)$  as the low-rank approximation. This scheme is unbiased since





$$\mathbb{E}[(MU)U^\top] = M \mathbb{E}[UU^\top] = MI = M.$$

PowerSGD can be seen as the natural biased counterpart of this unbiased scheme. Table 2.1 demonstrates that our biased approximator with error feedback outperforms the unbiased operator on image classification.





### 2.6.2 Effect of warm-start

PowerSGD does not compute the best rank- $r$  approximation of a gradient matrix, but uses a cheaper, low-fidelity approximation based on power iteration. Comparing the time per batch

Table 2.3: PowerSGD with varying rank. With sufficient rank, PowerSGD accelerates training of a ResNet-18 and an LSTM by reducing communication, achieving test quality on par with regular SGD in the same number of iterations. The time per batch includes the forward/backward pass (constant). See Section 2.7 for the experimental setup.

Image classification — ResNet on Cifar					
Algorithm	Test accuracy	Data sent per epoch		Time per batch	
SGD	94.3% 	1023 MB	(1×)	312 ms	+0%
Rank 1	93.6% 	4 MB	(243×)	229 ms	−26%
Rank 2	94.4% 	8 MB	(136×)	239 ms	−23%
Rank 4	94.5% 	14 MB	(72×)	260 ms	−16%

Language modeling — LSTM on WikiText					
Algorithm	Test perplexity	Data sent per epoch		Time per batch	
SGD	91 	7730 MB	(1×)	300 ms	+0%
Rank 1	102 	25 MB	(310×)	131 ms	−56%
Rank 2	93 	38 MB	(203×)	141 ms	−53%
Rank 4	91 	64 MB	(120×)	134 ms	−55%

of PowerSGD and Spectral Atomo in Table 2.6, we see the importance of avoiding a Singular Value Decomposition. With ResNet-18 gradients shaped like in PowerSGD, computing the SVD of a stochastic gradient takes 673ms, the equivalent of computing 6 mini-batch gradients on the system described in appendix A.2. In contrast, one full step of rank-2 PowerSGD, including communication between 16 workers, takes only 105ms.

Given that we only use a single step of power iteration, the quality of the approximation suffers—compare the test accuracy of ‘without warm start’ and ‘best approximation’ in Table 2.2. A key feature of PowerSGD is the *warm start* strategy which reuses previously computed matrix approximations to initialize the power iteration algorithm. If the matrix on which we perform power iteration remains constant, then this recovers the best rank- $r$  approximation (see Theorem VI in the Appendix). We argue that this strategy sometimes makes sense even if the underlying matrices are varying.





















Suppose we approximate the sequence of gradient matrices  $\{M_t\}$  at time steps  $t$ . At time step  $t$ , we leverage the previous factorization  $M_{t-1} \approx P_{t-1}Q_{t-1}^\top$ . If  $M_t \approx M_{t-1}$  then we would benefit from reusing  $P_{t-1}$  and  $Q_{t-1}$  as our starting point. While this is unlikely to be true, if  $M_t$  and  $M_{t-1}$  are stochastic approximations of the full gradient, we can expect that  $\mathbb{E}[M_t] \approx \mathbb{E}[M_{t-1}]$  since the function is smooth, and we only take small update steps. The result is akin to Oja’s algorithm for *stochastic power iteration* [Oja, 1982], and hence could result in an improved approximation quality. As we show empirically in Table 2.2, this ‘warm starting’ strategy is sufficient to close the gap in test accuracy between PowerSGD and the much more expensive best rank- $r$  approximation.

### 2.6.3 Effect of varying the rank

PowerSGD allows users to choose the rank of its gradient approximations. The trade-off between approximation quality and compression, decompression and transfer cost is explored in Table 2.3. In both the image classification and language modeling tasks we explore, the test quality achieved by PowerSGD grows with increasing rank. In both cases, it reaches a quality that is as good, or even slightly better than regular SGD.



Table 2.4: Comparing different compression operators for Error-feedback SGD in a unified setting; 300 epochs of Error-feedback SGD with Momentum (Algorithm 2) with a learning rate tuned for full-precision SGD on 16 GPUs for Cifar-10. The variations of PowerSGD with ranks 2 and 7 strike the best balance between the achieved test accuracy and time per batch (total time for forward, backward, compression, decompression, and gradient aggregation).

		Test accuracy	Sent/epoch	All-reduce	Time/batch
No compression		94.3% 	1023 MB	✓	312 ms 
Medium	<b>Rank 7</b>	94.6% 	24 MB	✓	285 ms 
	Random Block	93.3% 	24 MB	✓	243 ms 
	Random K	94.0% 	24 MB	✓	540 ms 
	Sign+Norm	93.9% 	32 MB	✗	429 ms 
	Top K	94.4% 	32 MB	✗	444 ms 
High	<b>Rank 2</b>	94.4% 	8 MB	✓	239 ms 
	Random Block	87.8% 	8 MB	✓	240 ms 
	Random K	92.6% 	8 MB	✓	534 ms 
	Top K	93.6% 	8 MB	✗	411 ms 

## 2.7 Results

This section demonstrates the practicality of PowerSGD for distributed deep learning. We show that the compression scheme of PowerSGD i) is fast and matches test performance of SGD, ii) scales well with increasing workers even with a suboptimal communication backend, and iii) significantly reduces training time for larger models.

Most of the analysis is performed on Cifar-10, in the setting described in the table on the right. We verify the generality of PowerSGD by an additional evaluation of an LSTM for language modeling on Wikitext-2. We use 16 GPUs on 8 machines, connected through a fast (10Gbit/s) network. To obtain meaningful timings, we have aimed to optimize all compared optimizers to a similar level. We provide a list of our performance optimizations in Appendix A.8. Throughout these results, we tune the learning rate for full-precision SGD, and use the *same* parameters for PowerSGD and other compression algorithms that use error feedback with momentum. Learning rates for the compared-to Spectral Atomo [Wang et al., 2018] and Signum [Bernstein et al., 2019] were separately tuned cf. Appendix A.9.

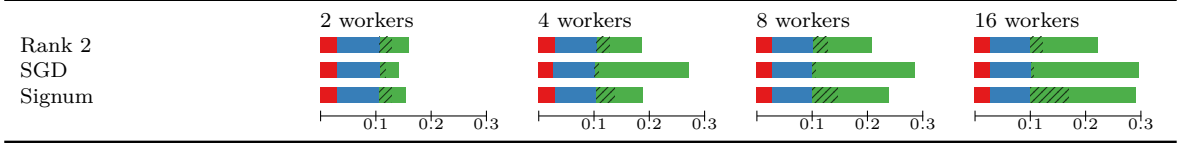
Default experimental setting	
Dataset	Cifar-10
Architecture	ResNet-18
Num. of workers	16
Backend	NCCL (fastest in PyTorch)
Batch size	128 × number of workers
Momentum	0.9
Learning rate	Tuned for 16 workers — 0.1 × 16 for SGD. Scaled linearly by the number of workers
LR decay	/10 at epoch 150 and 250
LR warmup	Linearly within 5 epochs, starting from the single-worker LR
# Epochs	300
Weight decay	10 <sup>-4</sup> , 0 for BatchNorm parameters
Repetitions	3, with varying seeds
Error bars	min — max

### 2.7.1 Comparison with other compressors

Error feedback in compressed optimization enables the use of a multitude of compression schemes, including biased ones. The potential compression operators illustrated in fig. 2.1 are compared in table 2.4. We evaluate compressors based on the test accuracy achieved and the total time taken to process one mini-batch. The former is a holistic measure of the accuracy of the compression operator, and the latter is the net time required for a forward pass, backward pass, gradient compression and decompression and gradient communication.

Table 2.5: Breakdown of time spent (in sec.) in one iteration of ResNet-18 training. Because PowerSGD (Rank 2) uses all-reduce, time spent encoding/decoding gradients is constant. Workers represent GPUs located in pairs of two per computer.

■ Forward pass, ■ Backward pass, ■ Gradient exchange, ■ Encoding and decoding.



We study two compression regimes—medium and high.

At around  $32\times$  compression, achieved by sign-based methods, all compression schemes (other than Random Block) achieve test accuracy close to full-precision SGD. This implies that all schemes in this regime (other than Random Block) obtain a good-enough compression quality. At high compression ( $128\times$ ), PowerSGD particularly stands out as the only method to achieve the target test accuracy.

In both the medium and high compression settings, the only schemes to be faster than full-precision SGD are PowerSGD and Random Block. Note that both are simple linear schemes and hence support all-reduce. While Random  $K$  also supports all-reduce, the overhead for random memory access during both the compression and decompression stages is substantial, making it slower overall than SGD. Thus, on modern GPU-enabled infrastructure, PowerSGD, which relies on matrix multiplication, is faster and much more accurate than the other compression schemes.

### 2.7.2 Scalability of PowerSGD

Here we investigate how PowerSGD scales with an increasing number of workers, shedding light on what we can expect if we use a significantly larger number of workers. Additionally, we investigate how these results depend on the choice of communication backend. We benchmark PowerSGD against SGD and Signum (signSGD with majority vote) from Bernstein et al. [2019] which we believe is the current state-of-the-art for distributed algorithms.

Table 2.5 provides a detailed breakdown of the time spent for each mini-batch (i.e., one step) into the forward pass, backward pass, gradient exchange (communication), and compression/decompression. The time spent in the forward and backward pass is constant across all algorithms and numbers of workers. Since both SGD and PowerSGD use all-reduce, the gradient communication time (solid green in Table 2.5) scales gracefully with increasing number of workers. Signum—which uses all-gather instead of all-reduce—has a steeper increase. It has comparable time to PowerSGD for 4 workers but becomes more expensive for 16 workers.

There is another, more subtle, consequence of all-reduce vs. all-gather on the decoding times. In all-reduce, the *aggregation* step and the *communication* step happen simultaneously. Each worker receives a pre-aggregated gradient, making the cost of decompression independent of the number of workers. On the other hand, in all-gather, a worker receives  $W$  compressed gradients that need to be individually decompressed and aggregated (either using majority vote or averaging). The time for decompression with all-gather therefore scales linearly with number of workers. This shows when comparing the hatched regions in Table 2.5. This observation speaks to the importance of the reduce operation for scalability.

We next study two different backends—the more optimized NCCL and the slower GLOO.

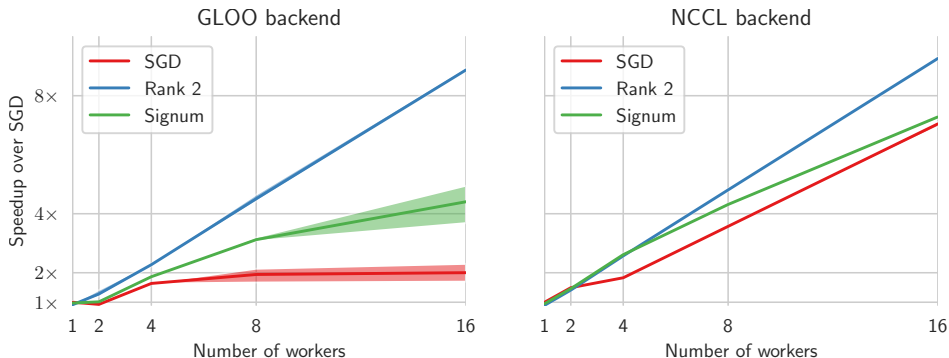


Figure 2.3: Scaling of PowerSGD on Cifar-10 compared to full-precision SGD and Signum [Bernstein et al., 2019] on two communication backends. The batch size increases linearly with the number of workers. We compare training time for one epoch to 1-worker SGD. The faster NCCL backend used throughout benefits the baselines more than our method. Workers represent GPUs located in pairs of two per computer.

All three methods scale reasonably well with the optimized NCCL backend, although Signum has a slope less than 1 in the log-log plot, indicating sublinear scaling. On the slower GLOO backend, PowerSGD is the only method that scales well due to its high compression rate.

### 2.7.3 Beneficial regularization

In some occasions, like in the results in Table 2.4, we observe that PowerSGD with a high enough rank can outperform SGD without compression, using a learning rate that was tuned for SGD without compression. We hypothesize that this is due to the beneficial regularization effect of PowerSGD. This observation also implies that, by tuning the learning rate and other regularization parameters like weight decay specifically for PowerSGD, the method could be used to improve model performance instead of to reduce communication.

### 2.7.4 Other tasks and methods

In Table 2.6, we compare PowerSGD against the state-of-the-art compressed optimization algorithms Signum and Spectral Atomo. The cost of performing a full SVD at each step renders Spectral Atomo impractical in a high-performance setting, especially because it fails to match the test accuracy of the other methods. Signum performs better, with a minor speedup over SGD. PowerSGD is the fastest and most accurate of the compared methods.

The advantage of PowerSGD truly shows when using really large models, i.e., where the communication actually becomes a bottleneck. To verify this, we run Signum, full-precision SGD, and PowerSGD to train an LSTM on a language modeling task which has a substantially larger model size than ResNet-18 (see Appendix A.6). To match the test score of full-precision SGD, we needed to use a rank-4 approximation (see Section 2.6.3). PowerSGD reduces communication by 90% and the overall running time by 55%, while Signum becomes slower than full-precision SGD and also obtains a worse test score.

Convergence curves on test accuracy corresponding to Tables 2.3, 2.6 and 2.7 are provided in Appendix A.3. Those figures show our improvements in time-to-accuracy for any target accuracy. Appendix A.4 contains a case study on using PowerSGD for a novel task (language modeling with transformers on Wikitext-2) and more workers (32) on the public cloud.

Table 2.6: Results on Cifar-10. Contrary to rank-2 Spectral Atomo [Wang et al., 2018] and Signum [Bernstein et al., 2019], PowerSGD achieves the same test accuracy as full-precision SGD within the default epoch budget.








Algorithm	Test accuracy	Data/epoch	Time per batch	
SGD	94.3% 	1023 MB	312 ms	+0%
Atomo	92.6% 	113 MB	948 ms	+204%
Signum	93.6% 	32 MB	301 ms	-3%
<b>Rank 2</b>	94.4% 	8 MB	239 ms	-23%

Table 2.7: In **language modeling**, rank-4 PowerSGD achieves the target test accuracy and provides a significant speedup over SGD.

Algorithm	Test perplexity	Data/epoch	Time per batch	
SGD	91 	7730 MB	300 ms	+0%
Signum	142 	242 MB	424 ms	+41%
<b>Rank 4</b>	91 	64 MB	134 ms	-55%

## 2.8 Conclusion

Gradient compression is a promising approach to tackling the communication bottleneck in synchronous distributed optimization. Thus far, however, it has not found widespread adoption because existing compression schemes either run slower than SGD with optimized all-reduce gradient aggregation, or more importantly do not reach the same test performance. We see PowerSGD as the first practical gradient compression method, and believe it is ready for adaptation in practice.

The key to the practicality of PowerSGD is its linear compression scheme that is cheap to compute and allows for all-reduce gradient aggregation, while simultaneously matching the test performance of full-precision SGD. This speedup gained over SGD actually *increases* for larger models such as those commonly found in NLP. Further, as a result of our modifications to the error-feedback algorithm, PowerSGD is a plug-in replacement for SGD with momentum, avoiding the need for additional hyperparameter tuning. We expect that these properties of PowerSGD will enable training of even larger models with even more workers than what is possible with full-precision SGD.

While PowerSGD enables faster training with larger batch sizes, increasing batch sizes are known to eventually suffer from a ‘generalization gap’ [Shallue et al., 2019]. This is an orthogonal issue that we see as the next step towards solving large-scale training. In our experiments, we have observed that PowerSGD can achieve higher test accuracy than SGD. Combined with the intriguing links between low-rankedness and generalization, this indicates that PowerSGD may also be helpful for closing the generalization gap in large batch training.

## 2.9 Acknowledgements

We thank Alp Yurtsever and Tao Lin for valuable discussions and the reviewers for their feedback. This project was supported by SNSF grant 200021\_175796, as well as a Google Focused Research Award.

## Chapter 3

# Low-rank gradient compression for decentralized learning

### 3.1 Preface

This chapter follows [Vogels et al., 2020], with minor edits.

**Summary** 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] (chapter 2), 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.

**Code** <https://github.com/epfml/powergossip>

**Co-authors** Sai Praneeth Karimireddy and Martin Jaggi.

#### Contributions

T. Vogels: methodology, software, visualization, writing.

S.P. Karimireddy: methodology, formal analysis, writing.

M. Jaggi: writing – review and editing, project administration, supervision.

### 3.2 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 communication efficiency, or for more fundamental reasons such as privacy constraints.

Decentralized machine learning enables collaborative processing of this new kind of data. In this paradigm, devices (nodes) have their own local data. The nodes jointly train a model by minimizing a loss function on their joint dataset. To do so, nodes communicate in a peer-to-peer fashion without any central coordination. A node can only communicate with few ‘neighbor’ nodes. This decentralized approach is not only useful in fundamentally decentralized systems, but the sparse communication patterns can sometimes even lead to efficiency gains in a datacenter [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.

In this paper, we study a specific class of low-rank compressors for decentralized optimization inspired by [Vogels et al., 2019, Cho et al., 2019] (chapter 2) that are reliable and require no tuning. Like in their work, we consider model parameters as matrices  $\mathbf{X}$ . Each pair of connected nodes  $(i, j)$  repeatedly estimates the difference between their parameters  $\mathbf{X}_i - \mathbf{X}_j$  through low-rank approximation. These approximations can be made without communicating full matrices due to the linearity of power iteration steps.

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.

### 3.3 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. For example, Alistarh et al. [2017], Wen et al. [2017], Seide et al. [2014], Bernstein et al. [2019], Karimireddy et al. [2019b] study gradient quantization, and [Lin et al., 2018, Stich et al., 2018, Wangni et al., 2018] sparsify gradients, keeping only large coordinates.

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.

PowerSGD [Vogels et al., 2019] (chapter 2), on which this work is based, is both linear

and low-rank and performed well in a recent benchmark [Xu et al., 2020]. An iteration of PowerSGD makes a low-rank approximation of the average error-corrected gradient across workers. The proposed decentralized scheme “PowerGossip” makes separate approximations for each pair of connected neighbors, directly approximating their pairwise model differences.

**Decentralized optimization** Decentralized, or ‘gossip’-based, optimization has been studied for many years [Tsitsiklis, 1984]. Popular methods include those based on (stochastic) sub-gradient descent [Nedic and 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. [2018a] 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 and Sa [2020] study a compression based on the assumption that model differences across connected nodes are coordinate-wise bounded. However, the above-mentioned methods introduce additional hyperparameters specific to compression (e.g., the consensus step size)—an inconvenience we overcome in this work.

### 3.4 Decentralized machine learning

In decentralized training of machine learning models there is no central ‘master’ node and nodes can only communicate with few other nodes, their ‘neighbors’. This can be a physical limitation of the network, but even in a datacenter, sparse, decentralized connectivity can be desirable for scalability [Assran et al., 2019]. Each worker has its own local training data, and these datasets may be non-identically distributed between workers. The data has to remain local to the nodes, either for privacy reasons, or to co-locate computation with data storage.

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

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

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 channels 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,  $\mathbf{v}_i \rightarrow \frac{1}{n} \sum_{i=1}^n \mathbf{v}_i \forall i$ .

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 method builds on the elegant D-SGD algorithm [Lian et al., 2017]. In D-SGD, for each time

step  $t$  and each worker  $i$ ,

$$\mathbf{X}_i^{(t+1)} := \mathbf{X}_i^{(t)} - \eta \nabla f_i(\mathbf{X}_i^{(t)}, \boldsymbol{\xi}_{i,t}) + \sum_{j \in \mathcal{N}_i} W_{ij} (\mathbf{X}_j^{(t)} - \mathbf{X}_i^{(t)}), \quad (3.1)$$

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

### 3.5 Algorithm

Naively applying lossy communication compression (quantization / sparsification) to the gossip update in Eq. (3.1) 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, 2018a]. 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  $\mathcal{C}_{\mathbf{v}}(\mathbf{X}) := (\mathbf{X}\mathbf{v})\mathbf{v}^\top$ , for a matrix  $\mathbf{X} \in \mathbb{R}^{p \times q}$  and vector  $\mathbf{v} \in \mathbb{R}^q$  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 compress the gossip part of Eq. (3.1):

$$\mathbf{X}_i^{(t+1)} := \mathbf{X}_i^{(t)} + \sum_{j \in \mathcal{N}_i} W_{ij} \mathcal{C}_{\mathbf{v}_{ij}}(\mathbf{X}_j^{(t)} - \mathbf{X}_i^{(t)}), \quad (3.2)$$

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  $\mathbf{D}^{(k)} := \mathbf{X}_j^{(k)} - \mathbf{X}_i^{(k)}$ , we choose  $\mathbf{v}^{(k)}$  based on the previous low-rank approximation. Starting with a random initial vector  $\mathbf{v}^{(0)}$ , we use

$$\mathbf{v}^{(2k+1)} := \frac{\mathbf{D}^{(2k)}\mathbf{v}^{(2k)}}{\|\mathbf{D}^{(2k)}\mathbf{v}^{(2k)}\|}, \quad \mathbf{v}^{(2k)} := \frac{\mathbf{D}^{(2k-1)\top}\mathbf{v}^{(2k-1)}}{\|\mathbf{D}^{(2k-1)\top}\mathbf{v}^{(2k-1)}\|}, \quad \forall k \in \mathbb{Z}_{\geq 0}. \quad (3.3)$$

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 3 describes how we use PowerGossip for stochastic optimization. Algorithm 4 presents the details of our compression scheme.



---

**Algorithm 3** Decentralized SGD with edge-wise compression

---

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

---

**Algorithm 4** Rank-1  $s$ -step PowerGossip compression for Algorithm 3

---

- 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  $k \leftarrow 0$ .
  - 2: **procedure**  $\mathcal{C}(\mathbf{X}_j - \mathbf{X}_i)$
  - 3:   **for**  $s$  power iteration steps **do**
  - 4:     **increment**  $k \leftarrow k + 1$
  - 5:     **if**  $k \equiv 1 \pmod{2}$  **then**
  - 6:        $\hat{\mathbf{v}} \leftarrow \frac{\mathbf{v}_{ij}}{\|\mathbf{v}_{ij}\|}$
  - 7:        $\mathbf{p}_j \leftarrow \mathbf{X}_j \hat{\mathbf{v}}, \quad \mathbf{p}_i \leftarrow \mathbf{X}_i \hat{\mathbf{v}} \quad \triangleright$  computed on nodes  $i$  and  $j$
  - 8:        $\hat{\mathbf{Q}} \leftarrow (\mathbf{p}_j - \mathbf{p}_i) \hat{\mathbf{v}}^\top$
  - 9:        $\mathbf{v}_{ij} \leftarrow \mathbf{p}_j - \mathbf{p}_i \quad \triangleright \mathbf{v}_{ij}$  changes between  $\mathbb{R}^p$  and  $\mathbb{R}^q$
  - 10:     **else**
  - 11:       do the same, but with  $\mathbf{X}$  transposed as in Eq. (3.3).
  - 12:     **end if**
  - 13:   **end for**
  - 14:   **return** the approximation  $\hat{\mathbf{Q}}$
  - 15: **end procedure**
  - 16: **note** that computations of  $\mathcal{C}(\mathbf{X}_j - \mathbf{X}_i) = -\mathcal{C}(\mathbf{X}_i - \mathbf{X}_j)$  overlap and share communication.
-

### 3.5.1 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 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 step size’ 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 3.7 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 D-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].

## 3.6 Theoretical analysis

### 3.6.1 Assumptions and setup

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

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

$$\nabla f_i(\mathbf{X}) \circ (\mathbf{X}^* - \mathbf{X}) \leq -\left(f_i(\mathbf{X}) - f_i(\mathbf{X}^*) + \frac{\mu}{2}\|\mathbf{X} - \mathbf{X}^*\|_F^2\right).$$

**(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  $\sigma^2$  and  $\zeta^2$  which bound the variance within and across different nodes, i.e., for any  $\mathbf{X}$  we have

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

Assumption A1, known as *star-convexity*, is weaker than the usual definition of convexity [Stich and Karimireddy, 2019]. While A3 requires both the variance within each node and across nodes be bounded, we allow heterogeneous (non-iid) data distributions across 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  $1 = \lambda_1^2 \geq \lambda_w^2 \geq \dots \lambda_n^2$  with *spectral gap*  $\rho := 1 - \lambda_2^2 > 0$ .

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 and 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 3 satisfies (A5):

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

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

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 another example closer to Algorithm 4, consider the following compressor for  $\mathbf{X} \in \mathbb{R}^{p,q}$ :

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

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  $\mathcal{L}(\mathbf{X}) := \mathbf{v}(\mathbf{v}^\top \mathbf{X})$  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 the compression operators defined in (A5) are a subset of those in [Koloskova et al., 2019b], they can still be of arbitrary approximation quality  $\delta > 0$ .

### 3.6.2 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.,  $\bar{\mathbf{X}} := \frac{1}{n} \sum_{i=1}^n \mathbf{X}_i$ .

**Compressed consensus** At every iteration, each worker  $i$  performs the following update:

$$\mathbf{X}_i^{(t)} := \mathbf{X}_i^{(t-1)} + \sum_{j \in \mathcal{N}_i} W_{ij} \left( \mathcal{C}_{ij}^{(t)}(\mathbf{X}_j^{(t-1)}) - \mathcal{C}_{ij}^{(t)}(\mathbf{X}_i^{(t-1)}) \right). \quad (3.4)$$

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  $\delta$ -approximate satisfying (A5) and that the mixing matrix  $\mathbf{W}$  has spectral gap  $\rho$  as in (A4), then the update (3.4) achieves consensus at a  $q$ -linear rate:

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

Note that update (3.4) requires no additional parameters and that our rate is linear in both  $\delta$  and  $\rho$ . When  $\delta = 1$ , i.e., with uncompressed messages, the rate in I corresponds to the classical consensus rate [Xiao and Boyd, 2004]. In contrast, [Koloskova et al., 2019b] require a consensus step size, 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 step sizes  $\{\eta_t\}$ :

$$\begin{aligned} \mathbf{Y}_i^{(t)} &:= \mathbf{X}_i^{(t-1)} - \eta_t \nabla F_i(\mathbf{X}, \boldsymbol{\xi}_{i,t}) \\ \mathbf{X}_i^{(t)} &:= \mathbf{Y}_i^{(t)} + \sum_{j \in \mathcal{N}_i} W_{ij} (\mathcal{C}_{ij}^{(t)}(\mathbf{Y}_j^{(t)}) - \mathcal{C}_{ij}^{(t)}(\mathbf{Y}_i^{(t)})). \end{aligned} \quad (3.5)$$

This algorithm is like PowerGossip, but it applies the consensus update of (3.4) 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

$$\mathbf{X}_i^{\text{out}} := \mathbf{X}_i^{(t)} \text{ with probability proportional to } \alpha_t. \quad (3.6)$$

**Theorem II** Suppose that assumptions A2–A5 hold at every round of (3.5). Then, in each of the following cases there exist a sequence of step sizes  $\{\eta_t\}$  and weights  $\{\alpha_t\}$  such that the output  $\bar{\mathbf{X}}^{\text{out}}$  computed using (3.5) and (3.6) is  $\varepsilon$ -accurate.

- **Non-convex**  $\mathbb{E} \|\nabla f(\bar{\mathbf{X}}^{\text{out}})\|^2 \leq \varepsilon$  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  $\mathbb{E}[f(\bar{\mathbf{X}}^{\text{out}})] - [f(\mathbf{X}^*)] \leq \varepsilon$  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  $\mathbb{E}[f(\bar{\mathbf{X}}^{\text{out}})] - [f(\mathbf{X}^*)] \leq \varepsilon$  after

$$T = \tilde{O}\left(\frac{\sigma^2}{n\mu\varepsilon} + \frac{\zeta + \sigma}{\rho\delta\mu\sqrt{\varepsilon}} + \frac{L}{\rho\delta\mu} \log\left(\frac{1}{\varepsilon}\right)\right) \text{ rounds.}$$

Let us focus on the strongly convex case ignoring logarithmic factors. Theorem II proves that the iteration complexity is  $\frac{\sigma^2}{n\mu\varepsilon} + \frac{\zeta + \sigma}{\rho\delta\mu\sqrt{\varepsilon}} + \frac{L}{\rho\delta\mu} \log\left(\frac{1}{\varepsilon}\right)$ . This can be decomposed into three terms. The first stochastic term  $\frac{\sigma^2}{n\mu\varepsilon}$  is independent of both the compression factor  $\delta$  as well as spectral-gap  $\rho$  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  $\frac{L}{\rho\delta\mu} \log\left(\frac{1}{\varepsilon}\right)$ , which is slowed down by a factor of  $\rho\delta$ . If  $\rho\delta = 1$  and  $\sigma^2 = \zeta = 0$ , 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  $\rho$  and  $\delta$  is linear in our rates while [Koloskova et al., 2019b] have a quadratic dependence on  $\rho$ . With exact communication ( $\delta = 1$ ) we recover the rates of [Koloskova et al., 2020].

### 3.7 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  $100 \times 100$  standard normal distribution, or *structured*, with  $64 \times 64$  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] (chapter 2) 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 following Appendix B.7 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 B.3.

**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 fig. 3.1. While on random data no compressed gossip algorithm outperforms full-precision gossip in bits to an arbitrary level of consensus, PowerGossip leverages 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 $\leftarrow \text{---} \text{---} \text{---} \text{---} \text{---}$
	w/ Power iteration	4.565 $\leftarrow \text{---} \text{---} \text{---} \text{---} \text{---}$
D-SGD	35 $\times$ communication	4.583 $\leftarrow \text{---} \text{---} \text{---} \text{---} \text{---}$

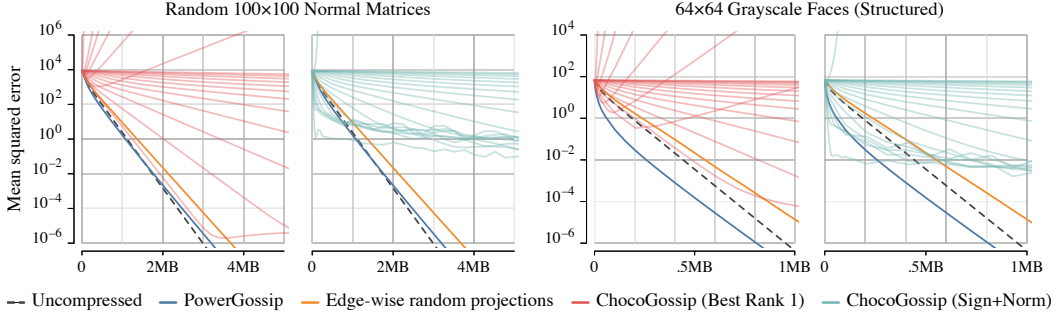


Figure 3.1: Consensus in an 8-ring. We study the level of consensus achieved as a function of bits transmitted by decentralized averaging. We compare out-of-the-box PowerGossip with power iterations and random projections against ChocoGossip [Koloskova et al., 2019b] with varying diffusion parameters. PowerGossip is competitive to the best tuned instances of ChocoGossip, and can leverage low rank structure in structured data (right).

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

Table 3.1: 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 D-SGD.

**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 B.9), 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 D-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 B.6), but it requires less memory to store the previous approximation and avoids an expensive orthogonalization step [Vogels et al., 2019] (chapter 2). Table 3.1 shows the effect of varying our compression rate while keeping the number of epochs fixed.

**Hyper-parameter tuning** We use the same learning rate tuned for centralized, uncompressed SGD for all PowerGossip configurations. Tables 3.1 and 3.2 show that we reach performance competitive to D-SGD on both tasks, at a similar compression rate to the best tuned configurations of ChocoSGD [Koloskova et al., 2019b] and DeepSqueeze [Tang et al., 2019].









Algorithm	$\eta$	$\gamma$	$\theta$	Test accuracy	Sent/epoch
All-reduce (baseline)	tuned			92.3% ——— 	
Uncompressed (D-SGD)	tuned			92.1% ——— 	102 MB
Choco (top-1%)	tuned	tuned		91.2% — 	3.1 MB (33 $\times$ )
Choco (Sign+Norm)	tuned	tuned		92.0% ——— 	3.2 MB (32 $\times$ )
Moniqua (2-bit)	tuned	tuned	tuned	90.7% 	6.4 MB (16 $\times$ )
DeepSqueeze (Sign+Norm)	tuned	tuned		91.2% — 	3.2 MB (32 $\times$ )
PowerGossip (1 iteration)	default			91.7% — 	1.8 MB (57 $\times$ )
PowerGossip (2 iterations)	default			91.9% ——— 	3.0 MB (34 $\times$ )

Table 3.2: 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  $\eta$ , averaging step size  $\gamma$ . Moniqua has an additional parameter  $\theta$  that can be computed or tuned.

### 3.8 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.

### 3.9 Acknowledgements

This project was supported by SNSF grant 200021\_175796, as well as a Google Focused Research Award. The experiments were run using Google Cloud credits donated by Google.





## Chapter 4

# The role of the topology in decentralized learning

### 4.1 Preface

This chapter follows [Vogels et al., 2022], with minor edits.

**Summary** In data-parallel optimization of machine learning models, workers collaborate to improve their estimates of the model: more accurate gradients allow them to use larger learning rates and optimize faster. We consider the setting in which all workers sample from the same dataset, and communicate over a sparse graph (decentralized). In this setting, current theory fails to capture important aspects of real-world behavior. First, the ‘spectral gap’ of the communication graph is not predictive of its empirical performance in (deep) learning. Second, current theory does not explain that collaboration enables *larger* learning rates than training alone. In fact, it prescribes *smaller* learning rates, which further decrease as graphs become larger, failing to explain convergence in infinite graphs. This paper aims to paint an accurate picture of sparsely-connected distributed optimization when workers share the same data distribution. We quantify how the graph topology influences convergence in a quadratic toy problem and provide theoretical results for general smooth and (strongly) convex objectives. Our theory matches empirical observations in deep learning, and accurately describes the relative merits of different graph topologies.

**Code** <https://github.com/epfml/topology-in-decentralized-learning>

**Co-authors** Hadrien Hendrikx and Martin Jaggi.

#### Contributions

T. Vogels: methodology, software, visualization, formal analysis (quadratic toy problem).

H. Hendrikx: methodology, formal analysis (full convex analysis), writing.

M. Jaggi: writing – review and editing, project administration, supervision.

### 4.2 Introduction

Distributed data-parallel optimization algorithms help us tackle the increasing complexity of machine learning models and of the data on which they are trained. We can classify those training algorithms as either *centralized* or *decentralized*, and we often consider those settings

to have different benefits over training ‘alone’. In the *centralized* setting, workers compute gradients on independent mini-batches of data, and they average those gradients between all workers. The resulting lower variance in the updates enables larger learning rates and faster training. In the *decentralized* setting, workers average their models with only a sparse set of ‘neighbors’ in a graph instead of all-to-all, and they may have private datasets sampled from different distributions. As the benefit of decentralized learning, we usually focus only on the (indirect) access to other worker’s datasets, and not of faster training.

While decentralized learning is typically studied with heterogeneous datasets across workers, sparse (decentralized) averaging between is also useful when worker’s data is identically distributed (i.i.d.) [Lu and Sa, 2021]. As an example, sparse averaging is used in data centers to mitigate communication bottlenecks [Assran et al., 2019]. In fact the D-SGD algorithm [Lian et al., 2017], on which we focus in this work, performs well mainly in this setting, while algorithmic modifications [Lorenzo and Scutari, 2016, Tang et al., 2018b, Vogels et al., 2021] are required to yield good performance on heterogeneous objectives. With i.i.d. data, the goal of sparse averaging is to optimize faster, just like with all-to-all averaging.

Yet, current decentralized learning theory poorly explains the i.i.d. case. Analyses typically show that, for *small enough* learning rates, training with sparse averaging behaves the same as with all-to-all averaging [Lian et al., 2017, Koloskova et al., 2020]. Compared to training alone with the *same small learning rate*, all-to-all averaging reduces the gradient variance by the number of workers. In practice, however, such small learning rates would never be used. In fact, a reduction in variance should allow us to use a *larger* learning rate than training alone, rather than imposing a *smaller* one. Contrary to current theory, we show that averaging reduces the variance from the start, instead of just asymptotically. Lower variance increases the maximum learning rate, which directly speeds up convergence. We characterize how much averaging with various communication graphs reduces the variance, and show that centralized performance is not always achieved when using optimal large learning rates. The behavior we explain is illustrated in fig. 4.1.

In current convergence rates, the graph topology appears through the *spectral gap* of its averaging (gossip) matrix. The spectral gap poses a conservative lower bound on how an averaging step brings all worker’s models closer together. The larger, the better. If the spectral gap is small, a significantly smaller learning rate is required to make the algorithm behave close to SGD with all-to-all averaging with the same learning rate. Unfortunately, we experimentally observe that, both in deep learning and in convex optimization, the spectral gap of a graph is *not predictive* of its performance under realistically tuned learning rates.

The problem with the spectral gap quantity is clearly illustrated in a simple example. Let the communication graph be a ring of varying size. As the size of the ring increases to infinity, its spectral gap goes to zero, since it becomes harder and harder to achieve consensus between all the workers. This leads to the optimization progress predicted by current theory to go to zero as well. Yet, this behavior does not match the empirical behavior of the rings with i.i.d. data. As the size of the ring increases, the convergence rate actually *improves* (fig. 4.1), until it saturates at a point that depends on the problem.

In this work, we aim to accurately describe the behavior of i.i.d. distributed learning algorithms with sparse averaging, both in theory and in practice. We quantify the role of the graph in a quadratic toy problem designed to mimic the initial phase of deep learning (section 4.4), showing that averaging enables a larger learning rate. From these insights, we derive a problem-independent notion of ‘effective number of neighbors’ in a graph that is consistent with time-varying topologies and infinite graphs, and is predictive of a graph’s

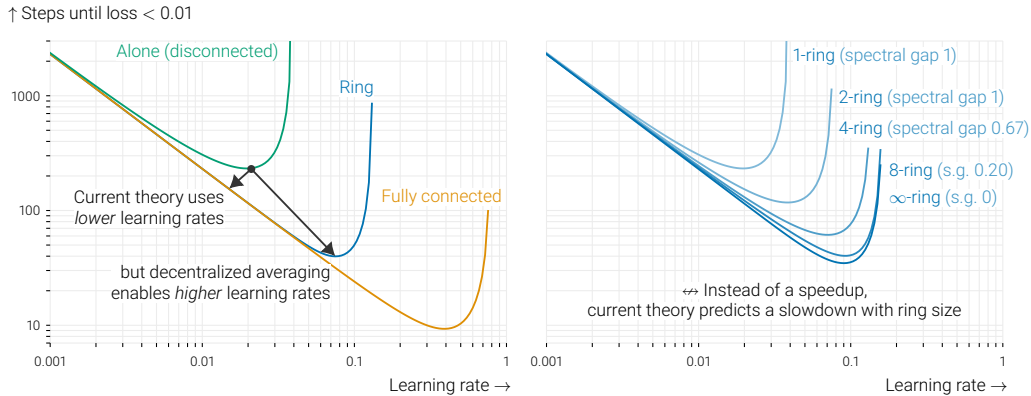


Figure 4.1: ‘Time to target’ for D-SGD [Lian et al., 2017] with constant learning rates on an i.i.d. isotropic quadratic dataset (section 4.4). The noise disappears at the optimum. Compared to optimizing alone, 32 workers in a ring (*left*) are faster for any learning rate, but the largest improvement comes from being able to use a large learning rate. This benefit is not captured by current theory, which prescribes a smaller learning rate than training alone. On the *right*, we see that rings of increasing size enable larger learning rates and faster optimization. Because a ring’s spectral gap goes to zero with the size, this cannot be explained by current theory.

empirical performance in both convex and deep learning. We provide convergence proofs for convex and (strongly) convex objectives that only mildly depend on the spectral gap of the graph (section 4.5), and consider the whole spectrum instead. At its core, our analysis does not enforce global consensus, but only between workers that are close to each other in the graph. Our theory shows that sparse averaging provably enables larger learning rates and thus speeds up optimization. These insights prove to be relevant in deep learning, where we accurately describe the performance of a variety of topologies, while their spectral gap does not (section 4.6).

## 4.3 Related work

**Decentralized SGD** This paper studies decentralized SGD. Koloskova et al. [2020] obtain the tightest bounds for this algorithm in the general setting where workers optimize heterogeneous objectives. Contrary to their work, we focus primarily on the case where all workers sample i.i.d. data from the same distribution. This important case is not described meaningfully by their analysis: while they show that gossip averaging reduces the asymptotic variance suffered by the algorithm, the fast initial linear decrease term in their convergence rate depends on the spectral gap of the gossip matrix. This key term does not improve through collaboration and gives rise to a *smaller learning rate* than training alone. Besides, as discussed above, this implies that optimization is not possible in the limit of large graphs, even in the absence of heterogeneity: for instance, the spectral gap of an infinite ring is zero, which would lead to a learning rate of zero as well.

These rates suggest that decentralized averaging speeds up the last part of training (dominated by variance), at the cost of slowing down the initial (linear convergence) phase. Beyond the work of Koloskova et al. [2020], many papers focus on *linear speedup* (in the variance phase)

over optimizing alone, and prove similar results in a variety of settings [Lian et al., 2017, Tang et al., 2018b, Lian et al., 2018]. All these results rely on the following insight: while linear speedup is only achieved for small learning rates, SGD eventually requires such small learning rates anyway (because of, e.g., variance, or non-smoothness). This observation leads these works to argue that “topology does not matter”. This is the case indeed, but only for very small learning rates, as shown in fig. 4.1. In practice, averaging speeds up both the initial *and* last part of training. This is what we show in this work, both in theory and in practice.

Another line of work studies D-(S)GD under statistical assumptions on the local data. In particular, Richards and Rebeschini [2020] show favorable properties for D-SGD with graph-dependent implicit regularization and attain optimal statistical rates. Their suggested learning rate does depend on the spectral gap of the communication network, and it goes to zero when the spectral gap shrinks. Richards and Rebeschini [2019] also show that larger (constant) learning rates can be used in decentralized GD, but their analysis focuses on decentralized kernel regression. It does not cover stochastic gradients, and relies on statistical concentration of local objectives rather than analysis on local neighborhoods.

**Gossiping in infinite graphs** An important feature of our results is that they only mildly depend on the spectral gap, and so they apply independently of the size of the graph. Berthier et al. [2020] study acceleration of gossip averaging in infinite graphs, and obtain the same conclusions as we do: although spectral gap is useful for asymptotics, it fails to accurately describe the transient regime of averaging. This is especially limiting for optimization (compared to of just averaging), as new local updates need to be averaged at every step. The transient regime of averaging deeply matters. Indeed, it impacts the quality of the gradient updates, and so it rules the asymptotic regime of optimization.

**The impact of the topology** Some works on linear speedup [Lian et al., 2017] argue that the topology of the graph does not matter. This is only true for asymptotic rates in specific settings, as illustrated in fig. 4.1. Neglia et al. [2020] investigate the impact of the topology on decentralized optimization, and contradict this claim. Compared to us, they make different noise assumptions, which in particular depend on the spectral distribution of the noise over the eigenvalues of the Laplacian (thus mixing computation and communication aspects). Although they show that the topology has an impact in the early phases of training (just like we do), they still get an unavoidable dependence on the spectral gap of the graph. Our results are different in nature, and show the benefits of averaging and the impact of the topology through the choice of large learning rates.

Another line of work studies the interaction of topology with particular patterns of data heterogeneity [Dandi et al., 2022, Bars et al., 2022], and how to optimize graphs with this heterogeneity in mind. These works “only” show a benefit from one-step gossip averaging and this is thus what they optimize the graph for. In contrast, we show that it is possible to benefit from distant workers beyond direct neighbors, too. This is an orthogonal direction, though the insights from our work could be used to strengthen their results.

**Time-varying topologies** Time-varying topologies are popular for decentralized deep learning in data centers due to their strong mixing [Assran et al., 2019, Wang et al., 2019]. The benefit of varying the communication topology over time is not easily explained through standard theory, but requires dedicated analysis [Ying et al., 2021]. While our proofs only cover static topologies, the quantities that appear in our analysis can be computed for time-varying schemes, too. With these quantities, we can empirically study static and time-varying schemes in the same framework.

#### 4.4 A toy problem: D-SGD on isotropic random quadratics

Before analyzing decentralized stochastic optimization through theory for general convex objectives and deep learning experiments, we first investigate a simple toy example that illustrates the behavior we want to explain in the analysis. In this setting, we can exactly characterize the convergence of decentralized SGD. We also introduce concepts that will be used throughout the paper.

We consider  $n$  workers that jointly optimize an isotropic quadratic  $\mathbb{E}_{\mathbf{d} \sim \mathcal{N}^d(0,1)} \frac{1}{2}(\mathbf{d}^\top \mathbf{x})^2 = \frac{1}{2}\|\mathbf{x}\|^2$  with a unique global minimum  $\mathbf{x}^* = \mathbf{0}$ . The workers access the quadratic through stochastic gradients of the form  $\mathbf{g}(\mathbf{x}) = \mathbf{d}\mathbf{d}^\top \mathbf{x}$ , with  $\mathbf{d} \sim \mathcal{N}^d(0,1)$ . This corresponds to a linear model with infinite data, and where the model can fit the data perfectly, so that stochastic noise goes to zero close to the optimum. We empirically find that this simple model is a meaningful proxy for the initial phase of (over-parameterized) deep learning (section 4.6). A benefit of this model is that we can compute exact rates for it. These rates illustrate the behavior that we capture more generally in the theory of section 4.5. Appendix C.3 contains a detailed version of this section that includes full derivations.

The stochasticity in this toy problem can be quantified by the *noise level*

$$\zeta = \sup_{\mathbf{x} \in \mathbb{R}^d} \frac{\mathbb{E}_{\mathbf{d}} \|\mathbf{d}\mathbf{d}^\top \mathbf{x}\|^2}{\|\mathbf{x}\|^2}, \quad (4.1)$$

which is equal to  $\zeta = d + 2$ , due to the random normal distribution of  $\mathbf{d}$ .

The workers run the D-SGD algorithm [Lian et al., 2017]. Each worker  $i$  has its own copy  $\mathbf{x}_i \in \mathbb{R}^d$  of the model, and they alternate between local model updates  $\mathbf{x}_i \leftarrow \mathbf{x}_i - \eta \mathbf{g}(\mathbf{x}_i)$  and averaging their models with others:  $\mathbf{x}_i \leftarrow \sum_{j=1}^n w_{ij} \mathbf{x}_j$ . The averaging weights  $w_{ij}$  are summarized in the *gossip matrix*  $\mathbf{W} \in \mathbb{R}^{n \times n}$ . A non-zero weight  $w_{ij}$  indicates that  $i$  and  $j$  are directly connected. In the following, we assume that  $\mathbf{W}$  is symmetric and doubly stochastic:  $\sum_{j=1}^n w_{ij} = 1 \forall i$ .

On our objective, D-SGD either converges or diverges linearly. Whenever it converges, i.e., when the learning rate is small enough, there is a convergence rate  $r$  such that

$$\mathbb{E} \|\mathbf{x}_i^{(t)}\|^2 \leq (1 - r) \|\mathbf{x}_i^{(t-1)}\|^2,$$

with equality as  $t \rightarrow \infty$  (proofs in appendix C.3). When the workers train alone ( $\mathbf{W} = \mathbf{I}$ ), the convergence rate for a given learning rate  $\eta$  reads:

$$r_{\text{alone}} = 1 - (\textcolor{red}{1} - \textcolor{red}{\eta})^2 - (\textcolor{green}{\zeta} - 1)\eta^2. \quad (4.2)$$

The optimal learning rate  $\eta^* = \frac{1}{\zeta}$  balances the optimization term  $(\textcolor{red}{1} - \textcolor{red}{\eta})^2$  and the stochastic term  $(\textcolor{green}{\zeta} - 1)\eta^2$ . In the centralized (fully connected) setting ( $w_{ij} = \frac{1}{n} \forall i, j$ ), the rate is simple as well:

$$r_{\text{centralized}} = 1 - (\textcolor{red}{1} - \textcolor{red}{\eta})^2 - \frac{(\textcolor{green}{\zeta} - 1)\eta^2}{\textcolor{blue}{n}}. \quad (4.3)$$

Averaging between  $\textcolor{blue}{n}$  workers reduces the impact of the gradient noise, and the optimal learning rate grows to  $\eta^* = \frac{\textcolor{blue}{n}}{\textcolor{blue}{n} + \textcolor{green}{\zeta} - 1}$ . D-SGD with a general gossip matrix  $\mathbf{W}$  interpolates those results.

To quantify the reduction of the  $(\textcolor{green}{\zeta} - 1)\eta^2$  term in general, we introduce the *problem-independent* notion of *effective number of neighbors*  $n_{\mathbf{W}}(\gamma)$  of the gossip matrix  $\mathbf{W}$  and *decay parameter*  $\gamma$ .

**Definition A** The effective number of neighbors  $n_{\mathbf{W}}(\gamma) = \lim_{t \rightarrow \infty} \frac{\sum_{i=1}^n \text{Var}[\mathbf{y}_i^{(t)}]}{\sum_{i=1}^n \text{Var}[\mathbf{z}_i^{(t)}]}$  measures the ratio of the asymptotic variance of the processes

$$\mathbf{y}^{(t+1)} = \sqrt{\gamma} \cdot \mathbf{y}^{(t)} + \boldsymbol{\xi}^{(t)}, \quad \text{where } \mathbf{y}^{(t)} \in \mathbb{R}^n \text{ and } \boldsymbol{\xi}^{(t)} \sim \mathcal{N}^n(0, 1) \quad (4.4)$$

and

$$\mathbf{z}^{(t+1)} = \mathbf{W}(\sqrt{\gamma} \cdot \mathbf{z}^{(t)} + \boldsymbol{\xi}^{(t)}), \quad \text{where } \mathbf{z}^{(t)} \in \mathbb{R}^n \text{ and } \boldsymbol{\xi}^{(t)} \sim \mathcal{N}^n(0, 1). \quad (4.5)$$

We call  $\mathbf{y}$  and  $\mathbf{z}$  *random walks* because workers repeatedly add noise to their state, somewhat like SGD’s parameter updates. This should not be confused with a ‘random walk’ over nodes in the graph.

Since averaging with  $\mathbf{W}$  decreases the variance of the random walk by at most  $n$ , the effective number of neighbors is a number between 1 and  $n$ . The decay  $\gamma$  modulates the sensitivity to communication delays. If  $\gamma = 0$ , workers only benefit from averaging with their direct neighbors. As  $\gamma$  increases, multi-hop connections play an increasingly important role. As  $\gamma$  approaches 1, delayed and undelayed noise contributions become equally weighted, and the reduction tends to  $n$  for any connected topology.

For regular doubly-stochastic symmetric gossip matrices  $\mathbf{W}$  with eigenvalues  $\lambda_1, \dots, \lambda_n$ ,  $n_{\mathbf{W}}(\gamma)$  has a closed-form expression

$$n_{\mathbf{W}}(\gamma) = \frac{\frac{1}{1-\gamma}}{\frac{1}{n} \sum_{i=1}^n \frac{\lambda_i^2}{1-\lambda_i^2 \gamma}}. \quad (4.6)$$

The notion of variance reduction in random walks, however, naturally extends to infinite topologies or time-varying averaging schemes. Figure 4.2 illustrates  $n_{\mathbf{W}}$  for various topologies.

In our exact characterization of D-SGD’s convergence on the isotropic quadratic toy problem (Appendix C.3), we find that the effective number of neighbors appears in place of the number of workers  $n$  in the fully-connected rate eq. (4.3). The rate is the unique solution to

$$r = 1 - (1 - \eta)^2 - \frac{(\zeta - 1)\eta^2}{n_{\mathbf{W}}\left(\frac{(1-\eta)^2}{1-r}\right)}. \quad (4.7)$$

For fully-connected and disconnected  $\mathbf{W}$ ,  $n_{\mathbf{W}}(\gamma) = n$  or 1 respectively, irrespective of  $\gamma$ , and Equation 4.7 recovers Equations 4.2 and 4.3. For other graphs, the effective number of workers depends on the learning rate. Current theory only considers the case where  $n_{\mathbf{W}} \approx n$ , but the small learning rates this requires can make the term  $(1 - \eta)^2$  too large, defeating the purpose of collaboration.

Beyond this toy problem, the notion of effective number of neighbors also turns out to be meaningful in the analysis of general objectives (section 4.5) and deep learning (section 4.6).

## 4.5 Theoretical analysis

In the previous section, we have derived exact rates for a specific function. Now we present convergence rates for general (strongly) convex functions that are consistent with our observations in the previous section. We obtain rates that depend on the level of noise, the hardness

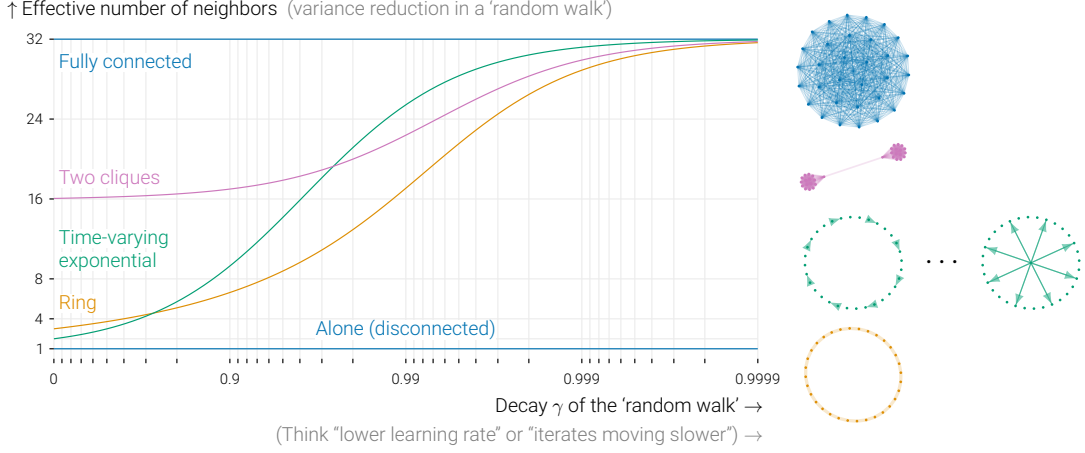


Figure 4.2: The effective number of neighbors for several topologies (appendix C.2) measured by their variance reduction in eq. (4.5). The point  $\gamma$  on the  $x$ -axis that matters depends on the learning rate and the task. The ‘best’ topology varies from problem to problem. For large decay rates  $\gamma$  (corresponding small learning rates), all connected topologies achieve variance reduction close to a fully connected graph. For small decay rates (large learning rates), workers only benefit from their direct neighbors, e.g., 3 in a ring. These curves can be computed explicitly for constant topologies, and simulated efficiently for the time-varying exponential scheme [Assran et al., 2019].

of the objective, and the topology of the graph. We will assume the following randomized model for D-SGD:

$$\mathbf{x}_i^{(t+1)} = \begin{cases} \mathbf{x}_i^{(t)} - \eta \nabla f_{\xi_i^{(t)}}(\mathbf{x}_i^{(t)}) & \text{with probability } \frac{1}{2}, \\ \sum_{j=1}^n w_{ij} \mathbf{x}_j^{(t)} & \text{otherwise,} \end{cases} \quad (4.8)$$

where  $f_{\xi_i^{(t)}}$  represent sampled data points and the gossip weights  $w_{ij}$  are elements of  $\mathbf{W}$ . This randomized model yields a clean analysis, but similar results hold for standard D-SGD as well (Appendix C.4.4).

**Assumption B** The stochastic gradients are such that: (I)  $\xi_i^{(t)}$  and  $\xi_j^{(\ell)}$  are independent for all  $t, \ell$  and  $i \neq j$ . (II)  $\mathbb{E}[f_{\xi_i^{(t)}}] = f$  for all  $t, i$  (III)  $\mathbb{E}\|\nabla f_{\xi_i^{(t)}}(\mathbf{x}^*)\|^2 \leq \sigma^2$  for all  $t, i$ , where  $\mathbf{x}^*$  is a minimizer of  $f$ . (IV)  $f_{\xi_i^{(t)}}$  is convex and  $\zeta$ -smooth for all  $t, i$ . (V)  $f$  is  $\mu$ -strongly-convex for  $\mu \geq 0$  and  $L$ -smooth.

The smoothness  $\zeta$  of the stochastic functions  $f_{\xi}$  defines the level of noise in the problem (the lower, the better). The ratio  $\zeta/L$  compares the difficulty of optimizing with stochastic gradients to the difficulty with the true global gradient (before reaching the ‘variance region’ of distance  $\mathcal{O}(\sigma^2)$  to the optimum). Assuming better smoothness for the global average objective than for the local functions is key to showing the benefit of averaging between workers. Without communication, convergence to the variance region is ensured for learning rates  $\eta \leq 1/\zeta$ . If  $\zeta \approx L$ , there is little noise and cooperation does not help before  $\|\mathbf{x}^{(t)} - \mathbf{x}^*\|^2 \approx \sigma^2$ . Yet, in noisy regimes ( $\zeta \gg L$ ), such as in section 4.4 in which  $\zeta = d + 2 \gg 1 = L$ , averaging enables larger step-sizes up to  $\min(1/L, n/\zeta)$ , greatly speeding up the initial training phase. This is precisely what we prove in Theorem III.

If the workers always remain close ( $\mathbf{x}_i \approx \frac{1}{n}(\mathbf{x}_1 + \dots + \mathbf{x}_n) \forall i$ , or equivalently  $\frac{1}{n}\mathbf{1}\mathbf{1}^\top \mathbf{x} \approx \mathbf{x}$ ), D-SGD behaves the same as SGD on the average parameter  $\frac{1}{n} \sum_{i=1}^n \mathbf{x}_i$ , and the learning rate depends on  $\max(\zeta/n, L)$ , showing a reduction of variance by  $n$ . To maintain “ $\frac{1}{n}\mathbf{1}\mathbf{1}^\top \mathbf{x} \approx \mathbf{x}$ ”, however, we require a small learning rate. This is a common starting point for the analysis of D-SGD, in particular for the proofs in Koloskova et al. [2020]. On the other extreme, if we do not assume closeness between workers, “ $\mathbf{I}\mathbf{x} \approx \mathbf{x}$ ” always holds. In this case, there is no variance reduction, but no requirement for a small learning rate either. In section 4.4, we found that, at the optimal learning rate, workers are *not* close to all other workers, but they *are* close to others that are not too far away in the graph.

We capture the concept of ‘local closeness’ by defining an averaging matrix  $\mathbf{M}$ . It allows us to consider semi-local averaging beyond direct neighbors, but without fully averaging with the whole graph. We ensure that “ $\mathbf{M}\mathbf{x} \approx \mathbf{x}$ ”, leading to some improvement in the smoothness between  $\zeta$  and  $\zeta/n$ , interpolating between the two previous cases. Each matrix  $\mathbf{M}$  implies a requirement on the learning rate, as well as an improvement in smoothness. Based on section 4.4, we therefore focus on a specific family of matrices that strike a good balance between the two: We choose  $\mathbf{M}$  as the covariance of a decay- $\gamma$  ‘random walk process’ with the graph, meaning that

$$\mathbf{M} = (1 - \gamma) \sum_{k=1}^{\infty} \gamma^{k-1} \mathbf{W}^{2k} = (1 - \gamma) \mathbf{W}^2 (1 - \gamma \mathbf{W}^2)^{-1}. \quad (4.9)$$

Varying  $\gamma$  induces a spectrum of neighborhoods from  $\mathbf{M} = \mathbf{W}^2$  ( $\gamma = 0$ ) to  $\mathbf{M} = \frac{1}{n}\mathbf{1}\mathbf{1}^\top$  ( $\gamma = 1$ ).  $\gamma$  also implies an effective number of neighbors  $n_{\mathbf{W}}(\gamma)$ : the larger  $\gamma$ , the larger  $n_{\mathbf{W}}(\gamma)$ .

Theorem III provides convergence rates for any value of  $\gamma$ , but the best rates are obtained for a specific  $\gamma$  that balances the benefit of averaging with the constraint it imposes on closeness between neighbors. In the following theorem, we assume that  $\mathbf{W}$  is regular and  $\mathbf{M}_{ii} = \mathbf{M}_{jj}$  for all  $i, j$ , so that  $\mathbf{M}_{ii}^{-1} = n_{\mathbf{W}}(\gamma)$ : the effective number of neighbors defined in (4.6) is equal to the inverse of the self-weights of matrix  $\mathbf{M}$ . For irregular graphs, all results still hold by replacing  $n_{\mathbf{W}}(\gamma)$  with  $\min_i \mathbf{M}_{ii}^{-1}$ , but they are more difficult to interpret in that case.

**Theorem III** If Assumption B holds, and the learning rate satisfies

$$\eta \leq \min \left( \frac{1}{8(\zeta/n_{\mathbf{W}}(\gamma) + L)}, \frac{1 - \gamma\lambda_2(\mathbf{W})}{2n_{\mathbf{W}}(\gamma)L} \right), \quad (4.10)$$

then the iterates obtained by (4.8) verify

$$\|\mathbf{x}^{(t)} - \mathbf{x}^*\|_{\mathbf{M}}^2 + \frac{1}{n_{\mathbf{W}}(\gamma)} \|\mathbf{x}^{(t)}\|_{\mathbf{I}-\mathbf{M}}^2 \leq \left(1 - \frac{\eta\mu}{2}\right)^t C_0 + \frac{8\eta\sigma^2}{n_{\mathbf{W}}(\gamma)}, \quad (4.11)$$

The bound on the learning rate (4.10) represents the tension between (i) reducing the noise  $\zeta$  by averaging with more people (larger  $n_{\mathbf{W}}(\gamma)$ ), which is the first term in the minimum, and (ii) staying close to all of them. A large spectral gap  $1 - \lambda_2(\mathbf{W})$  reduces the second constraint, but we allow non-trivial learning rates  $\eta > 0$  even when  $\lambda_2(\mathbf{W}) = 1$  (infinite graphs) if  $\gamma < 1$ .

Also note that the first term in the learning rate equation (4.10) corresponds well with the empirical observation that the learning rate can be scaled linearly with the number of workers as long as the noise  $\zeta$  dominates the smoothness constant  $L$ .



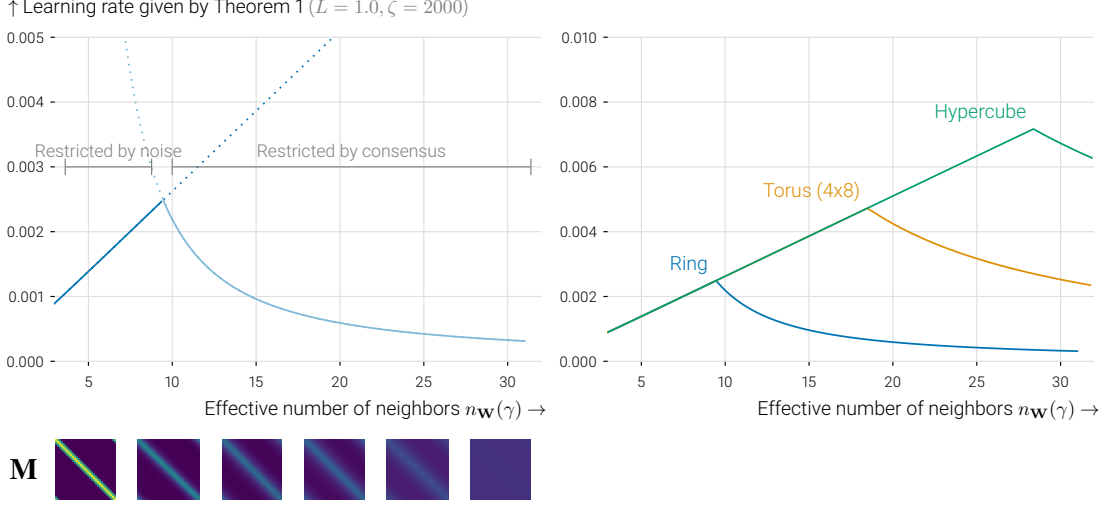


Figure 4.3: Maximum learning rates prescribed by theorem III, varying the parameter  $\gamma$  that implies an effective neighborhood size ( $x$ -axis) and an averaging matrix  $\mathbf{M}$  (drawn as *heatmaps*). On the *left*, we show the details for a 32-worker ring topology, and on the *right*, we compare it to more connected topologies. Increasing  $\gamma$  (and with it  $n_{\mathbf{W}}(\gamma)$ ) initially leads to larger learning rates thanks to noise reduction. At the optimum, the cost of consensus exceeds the benefit of further reduced noise.

Theorem III gives a rate for each parameter  $\gamma$  that controls the local neighborhood size. The task that remains is to find the  $\gamma$  parameter that gives the best convergence guarantees (the largest learning rate). As explained before, one should never reduce the learning rate in order to be close to others, because the goal of collaboration is to *increase* the learning rate. We should therefore pick  $\gamma$  such that the first term in Equation (4.10) dominates. This intuition is summarized in Corollary IV, which compares the performance of D-SGD with centralized SGD with fewer workers.

**Corollary IV** D-SGD is as fast as centralized mini-batch SGD with  $\mathcal{O}(n_{\mathbf{W}}(\gamma))$  workers, assuming that  $\zeta \geq nL$ , and that the parameter  $\gamma$  is the highest  $\gamma$  such that  $\frac{2n_{\mathbf{W}}(\gamma)^2}{1-\gamma\lambda_2(\mathbf{W})} \leq 32\frac{\zeta}{L}$ . This corresponds to a learning rate  $\eta = n_{\mathbf{W}}(\gamma)/16\zeta$ .

The typical D-SGD learning rates [Koloskova et al., 2020] are of order  $\mathcal{O}(\min(1/T, 1 - \lambda_2(W)))$ , which are much smaller than the learning rate of Corollary IV when  $\lambda_2(\mathbf{W})$  is large or the number of iterations large. We use the condition  $\zeta \geq nL$  only to present results in a simpler way. The condition  $\frac{2n_{\mathbf{W}}(\gamma)^2}{1-\gamma\lambda_2(\mathbf{W})}$  only depends on the size and topology of the graph, and can easily be computed in many cases. Thus, to obtain the best guarantees, we start from  $\gamma = 0$  and then increase it until either  $n_{\mathbf{W}}(\gamma) \approx n$ , the total size of the graph, or the two terms in the minimum match. This is how we obtain fig. 4.3.

*Proof sketch (Theorem III).* The proof relies on a simple argument: rather than bounding  $\|\mathbf{x}^{(t)} - \mathbf{x}^*\|^2$  or  $\|\frac{1}{n}\mathbf{1}\mathbf{1}^\top \mathbf{x}^{(t)} - \mathbf{x}^*\|^2$ , we analyze  $\|\mathbf{x}^{(t)} - \mathbf{x}^*\|_{\mathbf{M}}^2$ . This term better captures the benefit of averaging than  $\|\mathbf{x}^{(t)} - \mathbf{x}^*\|^2$ , thus leading to better smoothness constants, as long as  $\|\mathbf{x}^{(t)}\|_{\mathbf{I}-\mathbf{M}}^2$  is not too large. This yields fast rates without the need to guarantee that iterates between very distant workers remain close, which would be prohibitively expensive.  $\square$

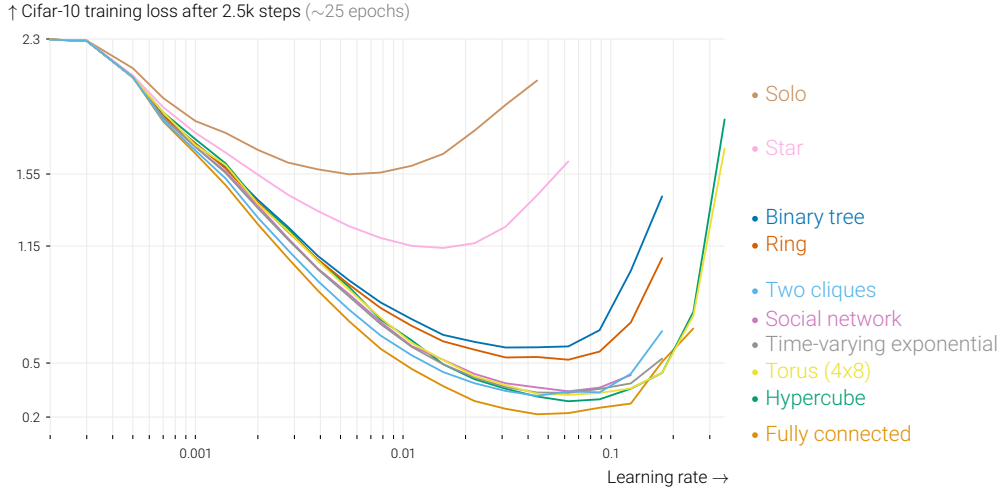


Figure 4.4: Training loss reached after 2.5k SGD steps with a variety of graph topologies. In all cases, averaging yields a small increase in speed for small learning rates, but a large gain over training alone comes from being able to increase the learning rate. While the star has a better spectral gap (0.031) than the ring (0.013), it performs worse, and does not allow large learning rates. For reference, similar curves for fully-connected graphs of varying sizes are in appendix C.6.

Theorem III is a special case of a more general theorem in Appendix C.4. That version covers, among other things, different choices of parameters, unbalanced communication and computation probabilities (thus allowing for local steps), and the convex ( $\mu = 0$ ) case.

## 4.6 Experimental analysis

While in the previous sections we have discussed isotropic quadratics or convex and smooth functions, the initial motivation for this work comes from observations in deep learning. First, it is crucial in deep learning to use a large learning rate in the initial phase of training [Li et al., 2019]. Contrary to what current theory prescribes, we do not use smaller learning rates in decentralized optimization than when training alone (even when data is heterogeneous.) And second, we find that the spectral gap of a topology is not predictive of the performance of that topology in deep learning experiments.

In this section, we experiment with several 32-worker topologies on Cifar-10 [Krizhevsky et al.] with VGG-11 [Simonyan and Zisserman, 2015]. Like other recent works [Lin et al., 2021, Vogels et al., 2021], we opt for this older model, because it does not include BatchNorm [Ioffe and Szegedy, 2015] which forms an orthogonal challenge for decentralized SGD. Please refer to appendix C.5 for full details on the experimental setup. Our set of topologies (appendix C.2) includes regular graphs like rings and toruses, but also irregular graphs such as a binary tree [Vogels et al., 2021] (chapter 5) and social network [Davis et al., 1930], and a time-varying exponential scheme [Assran et al., 2019]. We focus on the initial phase of training, 25 k steps in our case, where both train and test loss converge close-to linearly. A large learning rate in this phase is found to be important for generalization [Li et al., 2019].

Figure 4.4 shows the loss reached after the first 2.5k SGD steps for all topologies and for a dense grid of learning rates. The curves have the same global structure as those for isotropic

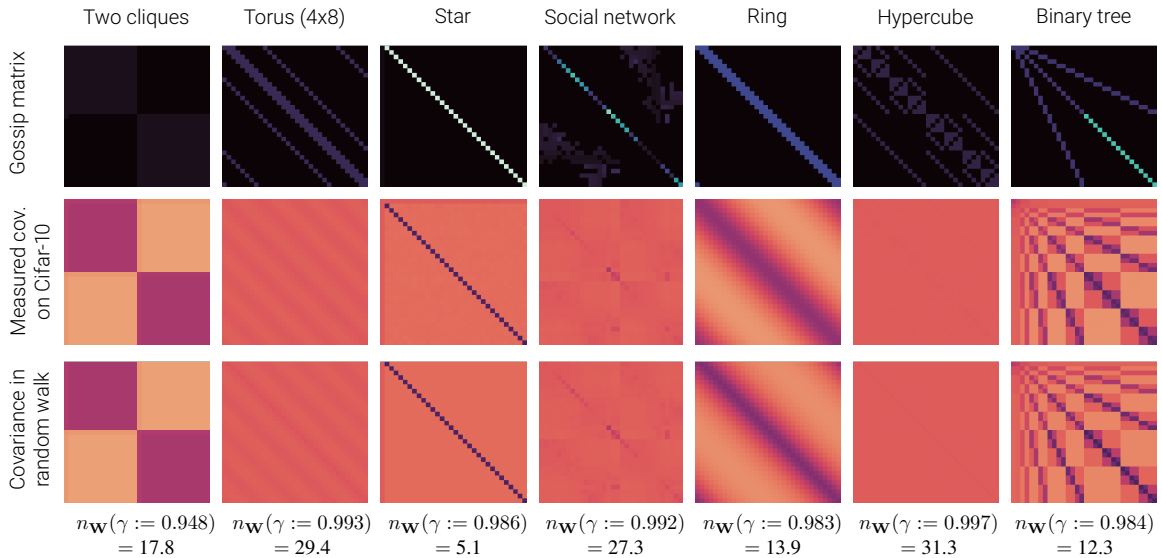


Figure 4.5: Measured covariance in Cifar-10 (second row) between workers using various graphs (top row). After 10 epochs, we store a checkpoint of the model and train repeatedly for 100 SGD steps, yielding 100 models for 32 workers. We show normalized covariance matrices between the workers. These are very well approximated by the covariance in the random walk process of section 4.4 (third row). We print the fitted decay parameters and corresponding ‘effective number of neighbors’.

quadratics fig. 4.1: (sparse) averaging yields a small increase in speed for small learning rates, but a large gain over training alone comes from being able to increase the learning rate. The best schemes support almost the same learning rate as 32 fully-connected workers, and get close in performance.

We also find that the random walks introduced in section 4.4 are a good model for variance between workers in deep learning. Figure 4.5 shows the empirical covariance between the workers after 100 SGD steps. Just like for isotropic quadratics, the covariance is accurately modeled by the covariance in the random walk process for a certain decay rate  $\gamma$ .

Finally, we observe that the effective number of neighbors computed by the variance reduction in a random walk (section 4.4) accurately describes the relative performance under tuned learning rates of graph topologies on our task, including for irregular and time-varying topologies. This is in contrast to the topology’s spectral gaps, which we find to be not predictive. We fit a decay rate  $\gamma = 0.951$  that seems to capture the specifics of our problem, and show the correlation in fig. 4.6.

In Appendix C.6.1, we replicate the same experiments in a different setting. There, we use larger graphs (of 64 workers), a different model and data set (an MLP on Fashion MNIST [Xiao et al., 2017]), and no momentum or weight decay. The results in this setting are qualitatively comparable to the ones presented above.

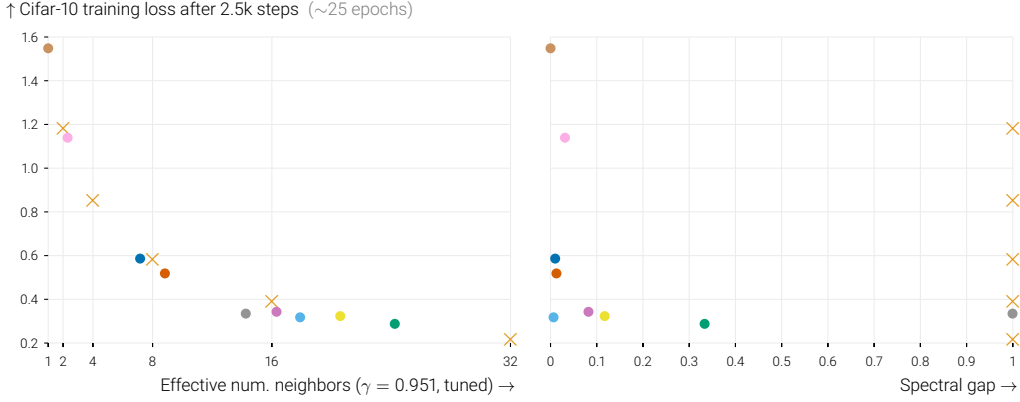


Figure 4.6: Cifar-10 training loss after 2.5k steps for all studied topologies with their optimal learning rates. Colors match fig. 4.4, and  $\times$  indicates fully-connected graphs with varying number of workers. After fitting a decay parameter  $\gamma = 0.951$  that captures problem specifics, the effective number of neighbors (left) as measured by variance reduction in a random walk (like in section 4.4) explains the relative performance of these graphs much better than the spectral gap of these topologies (right).

## 4.7 Conclusion

We have shown that the sparse averaging in decentralized learning allows larger learning rates to be used, and that it speeds up training. With the optimal large learning rate, the workers’ models are not guaranteed to remain close to their global average. Enforcing global consensus is unnecessary in the i.i.d. setting and the small learning rates it would require are counter-productive. With the optimal learning rate, models *do* remain close to some local average in a weighted neighborhood around them. The workers benefit from a number of ‘effective neighbors’, smaller than the whole graph, that allow them to use a large learning rate while retaining sufficient consensus within the ‘local neighborhood’.

Based on our insights, we encourage practitioners of sparse distributed learning to look beyond the spectral gap of graph topologies, and to investigate the actual ‘effective number of neighbors’ that is used. We also hope that our insights motivate theoreticians to be mindful of assumptions that artificially limit the learning rate.

We show experimentally that our conclusions hold in deep learning, but extending our theory to the non-convex setting is an important open direction that could reveal interesting new phenomena. Furthermore, an extension of our semi-local analysis to the heterogeneous setting where workers optimize different objectives could shed further light on the practical performance of D-SGD.

## 4.8 Acknowledgements

This project was supported by SNSF grant 200020\_200342.

We thank Lie He for valuable conversations and for identifying the discrepancy between a topology’s spectral gap and its empirical performance. We also thank Raphaël Berthier, Aditya Vardhan Varre and Yatin Dandi for their feedback on the manuscript.

## Chapter 5

# A relay mechanism for decentralized learning with heterogeneous data

### 5.1 Preface

This chapter follows [Vogels et al., 2021], with minor edits. We removed statements about the privacy benefits of decentralized learning, because such claims are difficult to substantiate [Pasquini et al., 2022] and unrelated to the topic of this thesis—communication efficiency.

**Summary** In decentralized machine learning, workers compute model updates on their local data. Because the workers only communicate with few neighbors without central coordination, these updates propagate progressively over the network. This paradigm enables distributed training on networks without all-to-all connectivity, helping to reduce the communication cost of distributed training. A key challenge, primarily in decentralized deep learning, remains the handling of differences between the workers’ local data distributions. To tackle this challenge, we study the RelaySum mechanism for information propagation in decentralized learning. RelaySum uses spanning trees to distribute information exactly uniformly across all workers with finite delays depending on the distance between nodes. In contrast, the typical gossip averaging mechanism only distributes data uniformly asymptotically while using the same communication volume per step as RelaySum. We prove that RelaySGD, based on this mechanism, is independent of data heterogeneity and scales to many workers, enabling highly accurate decentralized deep learning on heterogeneous data.

**Code** <http://github.com/epfml/relaysgd>.

**Co-authors** Lie He, Anastasia Koloskova, Tao Lin, Sai Praneeth Karimireddy, Sebastian U. Stich, Margit Jaggi.

#### Contributions

T. Vogels: methodology (60%), software (80%), visualization, writing (50%).

L. He: formal analysis (70%), methodology (40%), writing (50%).

A. Koloskova: formal analysis.

T. Lin: software.

S.P. Karimireddy: formal analysis.

S.U. Stich: formal analysis, writing – review and editing.

M. Jaggi: writing – review and editing, project administration, supervision.

## 5.2 Introduction

Ever-growing datasets lay at the foundation of the recent breakthroughs in machine learning. Learning algorithms therefore must be able to leverage data distributed over multiple devices. There are various paradigms for distributed learning, and they differ mainly in how the devices collaborate in communicating model updates with each other. In the *all-reduce* paradigm, workers average model updates with all other workers at every training step. In *federated learning* [McMahan et al., 2017], workers perform local updates before sending them to a central server that returns their global average to the workers. Finally, *decentralized learning* significantly generalizes the two previous scenarios. Here, workers communicate their updates with only few directly-connected neighbors in a network, without the help of a server.

Decentralized learning offers strong promise for new applications, allowing any group of agents to collaboratively train a model while respecting the data locality of each contributor [Nedic, 2020]. At the same time, it removes the single point of failure in centralized systems such as in federated learning [Kairouz et al., 2019], with the potential to improving robustness, security, and perhaps privacy. Even from a pure efficiency standpoint, decentralized communication patterns can speed up training in data centers [Assran et al., 2019].

In decentralized learning, workers share their local stochastic gradient updates with the others through *gossip* communication [Xiao and Boyd, 2004]. They send their updates to their neighbors, which iteratively propagate the updates further into the network. The workers typically use iterative *gossip averaging* of their models with their neighbors, using averaging weights chosen to ensure asymptotic uniform distribution of each update across the network. It will take  $\tau$  rounds of communication for an update from worker  $i$  to reach a worker  $j$  that is  $\tau$  hops away, and when it first arrives, the update is exponentially weakened by repeated averaging with weights  $< 1$ . In general networks, worker  $j$  will never exactly, but only asymptotically receive its uniform share of the update. The slow distribution of updates not only slows down training, but also makes decentralized learning sensitive to heterogeneity in workers' data distributions.

We study an alternative mechanism to gossip averaging, which we call RelaySum. RelaySum operates on spanning trees of the network, and distributes information exactly uniformly within a finite number of gossip steps equal to the diameter of the network. Rather than iteratively averaging models, each node acts as a 'router' that *relays* messages through the whole network without decaying their weight at every hop. While naive all-to-all routing requires  $n^2$  messages to be transmitted at each step, we show that on trees, only  $n$  messages (one per edge) are sufficient. This is enabled by the key observation that the routers can *merge* messages by *summation* to avoid any extra communication compared to gossip averaging. RelaySum achieves this using additional memory linear in the number of edges, and by tailoring the messages sent to different neighbors. At each time step, RelaySum workers receive a uniform average of exactly one message from each worker. Those messages just originate from different time delays depending on how many hops they travelled. The difference between gossip averaging and RelaySum is illustrated in fig. 5.1.

The RelaySum mechanism is structurally similar to Belief Propagation algorithms for inference in graphical models. This link was made by Zhang et al. [2019], who used the same mechanism for decentralized weighted average consensus in control.

We use RelaySum in the RelaySGD learning algorithm. We theoretically show that this algorithm is not affected by differences in workers' data distributions. Compared to other algorithms that have this property [Tang et al., 2018b, Pu and Nedic, 2018], RelaySGD does

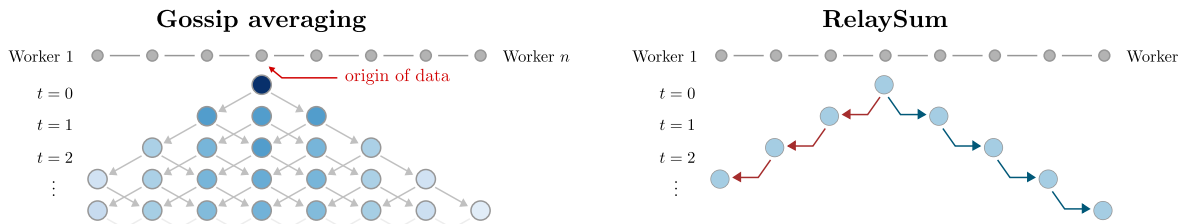


Figure 5.1: To spread information across a decentralized network, classical gossip averaging diffuses information slowly through the network. The left figure illustrates the spread of information originating from the fourth worker in a chain network. In RelaySum, the messages are *relayed* without reweighting, resulting in uniform delivery of the information to every worker. When multiple workers broadcast simultaneously (not pictured), RelaySum can *sum* their messages and use the same bandwidth as gossip averaging.

not require the selection of averaging weights, and its convergence does not depend on the spectral gap of the averaging matrix, but instead on the network diameter.

While RelaySum is formulated for trees, it can be used in any decentralized network. We use the Spanning Tree Protocol [Perlman, 1985] to construct spanning trees of any network in a decentralized fashion. RelaySGD often performs better on any such spanning tree than gossip-based methods on the original graph. When the communication network can be chosen freely, the algorithm can use double binary trees [Sanders et al., 2009]. While these trees have logarithmic diameter and scale to many workers, RelaySGD in this setup uses only constant memory equivalent to two extra copies of the model parameters and sends and receives only two models per iteration.

Surprisingly, in deep learning with highly heterogeneous data, prior methods that are theoretically independent of data heterogeneity [Tang et al., 2018b, Pu and Nedic, 2018], perform worse than heuristic methods that do not have this property, but use cleverly designed time-varying communication topologies [Assran et al., 2019]. In extensive tests on image- and text classification, RelaySGD performs better than both kinds of baselines at equal communication budget.

### 5.3 Related work

Out of the multitude of decentralized optimization methods, first-order algorithms that interleave local gradient updates with a form of gossip averaging [Nedic et al., 2017, Johansson et al., 2009] show most promise for deep learning. Such algorithms are theoretically analyzed for convex and non-convex objectives in [Nedic and Ozdaglar, 2009, Johansson et al., 2009, Nedic et al., 2017], and Lian et al. [2017], Tang et al. [2018b], Assran et al. [2019], Lin et al. [2021] demonstrate that gossip-based methods can perform well in deep learning.

In a gossip averaging step, workers average their local models with the models of their direct neighbors. The corresponding ‘mixing matrix’ is a central object of study. The matrix can be doubly-stochastic [Nedic et al., 2017, Lian et al., 2017, Koloskova et al., 2020], column-stochastic [Tsianos et al., 2012, Nedic and Olshevsky, 2016, Xi and Khan, 2017, Assran et al., 2019], row-stochastic [Xi et al., 2018, Xin et al., 2019], or a combination [Xin and Khan, 2018, 2020, Pu et al., 2021]. Column-stochastic methods use the *push-sum* consensus mechanism [Kempe et al., 2003] and can be used on directed graphs. Our analysis borrows

from the theory developed for those methods.

While gossip averages in general requires an infinite number of steps to reach exact consensus, another line of work identifies mixing schemes that yield exact consensus in finite steps [Sundaram and Hadjicostis, 2007, Sandryhaila et al., 2014, Charalambous et al., 2015]. For some graphs, this is possible with time-independent averaging weights [Ko and Gao, 2009, Georgopoulos, 2011]. One can also achieve finite-time consensus with time-varying mixing matrices. On trees, for instance, exact consensus can be achieved by routing updates to a root node and back, in exactly diameter number of steps [Ko and Gao, 2009, Georgopoulos, 2011]. On some graphs, tighter bounds can be established [Hendrickx et al., 2014]. For fully-connected networks with  $n$  workers, Assran et al. [2019] design a sparse time-varying communication scheme that yields exact consensus in a cycle of  $\log n$  averaging steps and performs well in deep learning.

The ‘relay’ mechanism of RelaySGD was previously used by Zhang et al. [2019] in the control community for the decentralized weighted average consensus problem, but they do not use it in the context of optimization. Zhang et al. also introduce a modified algorithm for loopy graphs, but this modification makes the achieved consensus inexact. The ‘relay’ mechanism effectively turns a sparse graph into a fully-connected graph with communication delays. Work on delayed consensus [Nedic and Ozdaglar, 2010] and optimization [Tsianos and Rabbat, 2011, Agarwal and Duchi, 2012] analyzes such schemes for centralized distributed algorithms. Those schemes are, however, not directly applicable to decentralized optimization.

A fundamental challenge in decentralized learning is dealing with data that is not identically distributed among workers. Because, in this case, workers pursue different optima, workers may drift [Nedic et al., 2017] and this can harm convergence. There is a large family of algorithms that use update corrections to provably mitigate such data heterogeneity. Examples applicable to non-convex problems are exact diffusion [Yuan et al., 2019], Gradient Tracking [Lorenzo and Scutari, 2016, Pu and Nedic, 2018, Zhang and You, 2019],  $D^2$  [Tang et al., 2018b], PushPull [Pu et al., 2021]. To tackle the same challenge, Lin et al. [2021], Yuan et al. [2021] propose modifications to local momentum to empirically improve performance in deep learning, but without provable guarantees. Lu and De Sa [2021] propose DeTAG which overlaps multiple consecutive gossip steps and gradient computations to accelerate information diffusion. This technique could be applied to the RelaySum mechanism, too.

## 5.4 Method

**Setup** We consider standard decentralized optimization with data on  $n \geq 1$  nodes:

$$f^* := \min_{\mathbf{x} \in \mathbb{R}^d} [f(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n [f_i(\mathbf{x}) := \mathbb{E}_{\xi \sim \mathcal{D}_i} F_i(\mathbf{x}, \xi)]] .$$

Here  $\mathcal{D}_i$  denotes the distribution of the data on node  $i$  and  $f_i: \mathbb{R}^d \rightarrow \mathbb{R}$  the local optimization objectives. Workers are connected by a network respecting a graph topology  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , where  $\mathcal{V} = \{1, \dots, n\}$  denotes the set of workers, and  $\mathcal{E}$  the set of undirected communication links between them (without self loops). Each worker  $i$  can only directly communicate with its neighbors  $\mathcal{N}_i \subset \mathcal{V}$ .

**Decentralized learning with gossip** We consider synchronous first-order algorithms that interleave local gradient-based updates

$$\mathbf{x}_i^{(t+1/2)} = \mathbf{x}_i^{(t)} + \mathbf{u}_i^{(t)}$$



with message exchange between connected workers. For SGD with typical gossip averaging (D-SGD [Lian et al., 2017]), the local updates can be written as  $\mathbf{u}_i^{(t)} = -\gamma \nabla f_i(\mathbf{x}_i^{(t)}, \xi_i^{(t)})$ , and the messages exchanged between pairs of connected workers  $(i, j)$  are  $\mathbf{m}_{i \rightarrow j}^{(t)} = \mathbf{x}_i^{(t+1/2)} \in \mathbb{R}^d$ . Each time step, the workers average their model with received messages,

$$\mathbf{x}_i^{(t+1)} = \mathbf{W}_{ii} \mathbf{x}_i^{(t+1/2)} + \sum_{j \in \mathcal{N}_i} \mathbf{W}_{ij} \mathbf{m}_{j \rightarrow i}^{(t)}, \quad (\text{DP-SGD})$$

using averaging weights defined by a *gossip matrix*  $\mathbf{W} \in \mathbb{R}^{n \times n}$ .

In this scheme, an update  $\mathbf{u}_i^{(t_1)}$  from any worker  $i$  will be linearly incorporated into the model  $\mathbf{x}_j^{(t_2)}$  at a later time step  $t_2$  with weight  $(\mathbf{W}^{t_2-t_1})_{ij}$ . The gossip matrix must be chosen such that these weights asymptotically converge to  $\frac{1}{n}$ , distributing all updates uniformly over the workers. This setup appears in, for example, [Lian et al., 2017, Koloskova et al., 2020].

**Uniform model averaging** If the graph topology is fully-connected, any worker can communicate with any other worker, and it is ideal to use ‘all-reduce averaging’,

$$\mathbf{x}_i^{(t+1)} = \frac{1}{n} \sum_{j=1}^n \mathbf{x}_j^{(t+1/2)}.$$

Contrary to the decentralized scheme (DP-SGD), this algorithm does not degrade in performance if data is distributed heterogeneously across workers. In sparsely connected networks, however, all-reduce averaging requires routing messages through the network. On arbitrary networks, such a routing protocol requires at least a number of communication steps equal to the network diameter  $\tau_{\max}$ —the minimum number of hops some messages have to travel.

**RelaySGD** In this paper, we approximate the all-reduce averaging update as

$$\mathbf{x}_i^{(t+1)} = \frac{1}{n} \sum_{j=1}^n \mathbf{x}_j^{(t-\tau_{ij}+1/2)}, \quad (\text{RelaySGD})$$

where  $\tau_{ij}$  is minimum number of network hops between workers  $i$  and  $j$  (and  $\tau_{ii} = 0$ ). Since it takes  $\tau_{ij}$  steps to route a message from worker  $i$  to  $j$ , this scheme could be implemented using a peer-to-peer routing protocol like Ethernet. Of course, this naive implementation drastically increases the bandwidth used compared to gossip averaging. The key insight of this paper is that, on tree networks, the RelaySGD update rule can be implemented while using the same communication volume per step as gossip averaging, using additional memory linear in the number of a worker’s direct neighbors.

**RelaySum** To implement RelaySGD, we require a communication mechanism that delivers sums of delayed ‘parcels’  $s_w^{(t)} = \sum_{j=1}^n p_j^{(t-\tau_{wj})}$  to each worker  $w$  in a tree network, where the parcel  $p_j^{(t)}$  is created by worker  $j$  at time  $t$ . To simplify the exposition, let us first consider the simplest type of tree network: a chain. In a chain, a worker  $w$  is connected to workers  $w-1$  and  $w+1$ , if those exist, and the delays are  $\tau_{ij} = |i-j|$ . We can then decompose

$$s_w^{(t)} = \sum_{j=1}^n p_j^{(t-\tau_{wj})} = p_w^{(t)} + \underbrace{\sum_{j=1}^{w-1} p_j^{(t-\tau_{wj})}}_{\text{parcels from the ‘left’}} + \underbrace{\sum_{j=w+1}^n p_j^{(t-\tau_{wj})}}_{\text{parcels from the ‘right’}}.$$

The sum of parcels from the ‘left’ will be sent as one message  $m_{(w-1) \rightarrow w}$  from worker  $w-1$  to  $w$ , and the sum of data from the ‘right’ will be sent as one message  $m_{(w+1) \rightarrow w}$  from  $w+1$  to

**Algorithm 5** RelaySGD

---

**Input:**  $\forall i, \mathbf{x}_i^{(0)} = \mathbf{x}^{(0)}; \forall i, j, \mathbf{m}_{i \rightarrow j}^{(-1)} = \mathbf{0}$ , counts  $c_{i \rightarrow j}^{(-1)} = 0$ , learning rate  $\gamma$ , tree network

- 1: **for**  $t = 0, 1, \dots$  **do**
- 2:   **for** node  $i$  **in parallel**
- 3:      $\mathbf{x}_i^{(t+1/2)} = \mathbf{x}_i^{(t)} - \gamma \nabla f_i(\mathbf{x}_i^{(t)})$  (or Adam/momentum)
- 4:     **for** each neighbor  $j \in \mathcal{N}_i$  **do**
- 5:       Send  $\mathbf{m}_{i \rightarrow j}^{(t)} = \mathbf{x}_i^{(t+1/2)} + \sum_{k \in \mathcal{N}_i \setminus j} \mathbf{m}_{k \rightarrow i}^{(t-1)}$  (relay messages from other neighbors)
- 6:       Send corresponding counters  $c_{i \rightarrow j}^{(t)} = 1 + \sum_{k \in \mathcal{N}_i \setminus j} c_{k \rightarrow i}^{(t-1)}$
- 7:       Receive  $(\mathbf{m}_{j \rightarrow i}^{(t)}, c_{j \rightarrow i}^{(t)})$  from node  $j$
- 8:     **end for**
- 9:      $\bar{n}_i^{(t+1)} = 1 + \sum_{j \in \mathcal{N}_i} c_{j \rightarrow i}^{(t)}$  ( $\bar{n}$  converges to the total number of workers)
- 10:     $\mathbf{x}_i^{t+1} = \frac{1}{\bar{n}_i^{(t+1)}} \left( \mathbf{x}_i^{(t+1/2)} + \sum_{j \in \mathcal{N}_i} \mathbf{m}_{j \rightarrow i}^{(t)} \right)$   $\left( = \frac{1}{n} \sum_{j=1}^n \mathbf{x}_j^{(t-\tau_{ij}+1/2)} \right)$
- 11:    **end for**
- 12: **end for**

---

$w$ . Neighboring workers can compute these messages from the messages they received from their neighbors in the previous time step. Compared to typical gossip averaging, RelaySum requires additional memory linear in the number of neighbors, but it uses the same volume of communication.

Algorithm 5 shows how this scheme is generalized to general tree networks and incorporated into RelaySGD. Along with the model parameters, we send scalar counters that are used in the first few iterations of the algorithm  $t \leq \tau_{\max}$  to correct for messages that have not yet arrived.

**Spanning trees** RelaySGD is formulated on tree networks, but it can be used on any communication graph by constructing a spanning tree. In a truly decentralized setting, we can use the Spanning Tree Protocol [Perlman, 1985] used in Ethernet to find such trees in a decentralized fashion. The protocol elects a leader as the root of the tree, after which every other node finds the fastest path to this leader.

On the other hand, when the decentralized paradigm is used in a data center to reduce communication, RelaySGD can run on double binary trees [Sanders et al., 2009] used in MPI and NCCL [Jeauguey, 2019]. The key idea of double binary trees is to use two different communication topologies for different parts of the model. We communicate odd coordinates using a balanced binary tree  $A$ , and communicate the even coordinates with a complimentary tree  $B$ . The trees  $A$  and  $B$  are chosen such that internal nodes (with 3 edges) in one tree are leaves (with only 1 edge) in the other. Using the combination of two trees, RelaySGD requires only constant extra memory equivalent to at most 2 model copies (just like the Adam optimizer [Kingma and Ba, 2015]), and it sends and receives the equivalent of 2 models (just like on a ring).

## 5.5 Theoretical analysis

Since RelaySGD updates worker's models at time step  $t+1$  using models from (at most) the past  $\tau_{\max}$  steps, we conveniently reformulate RelaySGD in the following way: Let  $\mathbf{Y}^{(t)}, \mathbf{G}^{(t)} \in$

$\mathbb{R}^{n(\tau_{\max}+1) \times d}$  denote stacked models and gradients across workers. Their row vectors at index  $n \cdot \tau + i$  represent

$$\left[\mathbf{Y}^{(t)}\right]_{n\tau+i}^\top = \begin{cases} \mathbf{x}_i^{(t-\tau)} & t \geq \tau \\ \mathbf{x}^{(0)} & \text{otherwise} \end{cases}, \quad \left[\mathbf{G}^{(t)}\right]_{n\tau+i}^\top = \begin{cases} \nabla F_i(\mathbf{x}_i^{(t-\tau)}; \xi_i^{(t-\tau)}) & t \geq \tau \\ \mathbf{x}^{(0)} & \text{otherwise} \end{cases}$$

for all  $t \geq 0$ , delay  $\tau \in [0, \tau_{\max}]$  and workers  $i \in [n]$ . Then (RelaySGD) can be written as

$$\mathbf{Y}^{(t+1)} = \mathbf{W}\mathbf{Y}^{(t)} - \gamma \tilde{\mathbf{W}}\mathbf{G}^{(t)}$$

where  $\mathbf{W}, \tilde{\mathbf{W}} \in \mathbb{R}^{n(\tau_{\max}+1) \times n(\tau_{\max}+1)}$  are non-negative matrices whose elements are

$$[\mathbf{W}]_{n\tau+i, n\tau'+j} = \begin{cases} \frac{1}{n} & \tau = 0 \text{ and } \tau' = \tau_{ij} \\ 1 & i = j \text{ and } \tau = \tau' + 1 \\ 0 & \text{otherwise} \end{cases}, \quad [\tilde{\mathbf{W}}]_{n\tau+i, n\tau'+j} = \begin{cases} \frac{1}{n} & \tau = 0 \text{ and } \tau' = \tau_{ij} \\ 0 & \text{otherwise} \end{cases}$$

for all  $\tau, \tau' \in [0, \tau_{\max}]$  and  $i, j \in [n]$ .  $\mathbf{W}$  can be interpreted as the mixing matrix of an ‘augmented graph’ [Nedic and Ozdaglar, 2010] with additional virtual ‘forwarding nodes’.  $\mathbf{W}$  is row stochastic with largest eigenvalue 1. The all-ones vector  $\mathbf{1}_{n(\tau_{\max}+1)} \in \mathbb{R}^{n(\tau_{\max}+1)}$  is a right eigenvector of  $\mathbf{W}$  and  $\boldsymbol{\pi} \in \mathbb{R}^{n(\tau_{\max}+1)}$  is the left eigenvector for which  $\boldsymbol{\pi}^\top \mathbf{1}_{n(\tau_{\max}+1)} = 1$ .

We characterize the convergence rate of the consensus distance in the following key lemma:

**Lemma 1 (Key lemma)** There exists an integer  $m = m(\mathbf{W}) > 0$  such that for any  $\mathbf{X} \in \mathbb{R}^{n(\tau_{\max}+1) \times d}$  we have

$$\|\mathbf{W}^m \mathbf{X} - \mathbf{1} \boldsymbol{\pi}^\top \mathbf{X}\|^2 \leq (1-p)^{2m} \|\mathbf{X} - \mathbf{1} \boldsymbol{\pi}^\top \mathbf{X}\|^2,$$

where  $p = \frac{1}{2}(1 - |\lambda_2(\mathbf{W})|)$  is a constant.

All the following optimization convergence results will only depend on the *effective spectral gap*  $\rho := \frac{p}{m}$  of  $\mathbf{W}$ . We empirically observe that  $\rho = \Theta(1/n)$  for a variety of network topologies (see fig. D.1 in Appendix D.1).

**Remark 2** The above key lemma is similar to [Koloskova et al., 2020, Assumption 4] for gossip-type averaging with symmetric matrices. However, in our case  $\mathbf{W}$  is just a row stochastic matrix, and its spectral norm  $\|\mathbf{W}\|_2 > 1$ . In general, the consensus distance can increase after just one single communication step (multiplication by  $\mathbf{W}$ ). That is why we need  $m > 1$ . The proof of the Lemma relies on a Perron-Frobenius type theorem, and holds over several steps  $m$  instead of a single iteration. It means RelaySum defines a consensus algorithm with linear convergence rate which pulls models closer.

Our main convergence results make the following assumptions [Koloskova et al., 2020].

**Assumption C (L-smoothness)** For each  $i \in [n]$ ,  $F_i(\mathbf{x}, \xi) : \mathbb{R}^D \times \Omega_i \rightarrow \mathbb{R}$  is differentiable for each  $\xi \in \text{supp}(\mathcal{D}_i)$  and there exists a constant  $L \geq 0$  such that for each  $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ ,  $\xi \in \text{supp}(\mathcal{D}_i)$ :

$$\|\nabla F_i(\mathbf{x}, \xi) - \nabla F_i(\mathbf{y}, \xi)\| \leq L \|\mathbf{x} - \mathbf{y}\|.$$

**Assumption D (uniform bounded noise)** There exists constant  $\bar{\sigma}$ , such that for all  $\mathbf{x} \in \mathbb{R}^d$  and  $i \in [n]$ ,

$$\mathbb{E}_\xi \|\nabla F_i(\mathbf{x}, \xi) - \nabla f_i(\mathbf{x})\|^2 \leq \bar{\sigma}^2.$$

**Assumption E ( $\mu$ -convexity)** For  $i \in [n]$ , each function  $f_i : \mathbb{R}^d \rightarrow \mathbb{R}$  is  $\mu$ -(strongly) convex for constant  $\mu \geq 0$ . That is,  $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^d$

$$f_i(\mathbf{x}) - f_i(\mathbf{y}) + \frac{\mu}{2} \|\mathbf{x} - \mathbf{y}\|_2^2 \leq \nabla f_i(\mathbf{x})^\top (\mathbf{x} - \mathbf{y}).$$

**Theorem V (RelaySGD)** For any target accuracy  $\epsilon > 0$  and an optimal solution  $\mathbf{x}^*$ ,

- **Convex** Under Assumptions C, D and E with  $\mu \geq 0$ , it holds that the average suboptimality  $\frac{1}{T+1} \sum_{t=0}^T (f(\bar{\mathbf{x}}^{(t)}) - f(\mathbf{x}^*)) \leq \epsilon$  after

$$\mathcal{O} \left( \frac{\bar{\sigma}^2}{n\epsilon^2} + \frac{C\sqrt{L}\bar{\sigma}}{\epsilon^{3/2}} + \frac{CL}{\epsilon} \right) R_0^2$$

iterations. Here  $\bar{\mathbf{x}}^{(t)} := \boldsymbol{\pi}^\top \mathbf{Y}^{(t)}$  averages past models, the constant  $R_0^2 = \|\mathbf{x}^0 - \mathbf{x}^*\|^2$ , and the constant  $C = \mathcal{O}(\frac{1}{\rho} \tau_{\max}^{3/2})$ .

- **Non-convex** Under Assumptions C and D, it holds that  $\frac{1}{T+1} \sum_{t=0}^T \|\nabla f(\bar{\mathbf{x}}^{(t)})\|^2 \leq \epsilon$  after

$$\mathcal{O} \left( \frac{\bar{\sigma}^2}{n\epsilon^2} + \frac{C\bar{\sigma}}{\epsilon^{3/2}} + \frac{C}{\epsilon} \right) LF_0$$

iterations, where  $F_0 := f(\bar{\mathbf{x}}^{(0)}) - f(\mathbf{x}^*)$ .

The dominant term in our convergence result,  $\mathcal{O}(\frac{\bar{\sigma}^2}{n\epsilon^2})$  matches with the dominant term in the convergence rate of centralized (‘all-reduce’) SGD, and thus cannot be improved. In contrast to other methods, the presented convergence result of RelaySGD is independent of the data heterogeneity  $\zeta^2$  in [Koloskova et al., 2020, Assumption 3b].

**Definition F (data heterogeneity)** There exists a constant  $\zeta^2$  such that  $\forall i \in [n], \mathbf{x} \in \mathbb{R}^d$

$$\|\nabla f_i(\mathbf{x}) - \nabla f(\mathbf{x})\|_2^2 \leq \zeta^2.$$

**Remark 3** For convex objectives, Assumptions D and F can be relaxed to only hold at the optimum  $\mathbf{x}^*$ . A weaker variant of assumption C only uses  $L$ -smoothness of  $f_i$  [Koloskova et al., 2020, Assumption 1b].

Comparing to gossip averaging for convex  $f_i$  which has complexity  $\mathcal{O}(\frac{\bar{\sigma}^2}{n\epsilon^2} + (\frac{\zeta}{\rho} + \frac{\bar{\sigma}}{\sqrt{\rho}}) \frac{\sqrt{L}}{\epsilon^{3/2}} + \frac{L}{\rho\epsilon}) R_0^2$ , our rate for RelaySGD is independent of  $\zeta^2$  and has same leading term  $\mathcal{O}(\frac{\bar{\sigma}^2}{n\epsilon^2})$  as  $D^2$ .

## 5.6 Experimental analysis and practical properties

### 5.6.1 Effect of network topology

**Random quadratics** To efficiently investigate the scalability of RelaySGD with respect to the number of workers, and to study the benefits of binary tree topologies over chains, we introduce a family of synthetic functions. We study *random quadratics* with local cost functions  $f_i(\mathbf{x}) = \|\mathbf{A}_i \mathbf{x} - \mathbf{b}_i\|_2^2$  to precisely control all constants that appear in our theoretical analysis. The Hessians  $\mathbf{A}_i$  are initialized randomly, and their spectrum is scaled to achieve a desired smoothness  $L$  and strong convexity  $\mu$ . The offsets  $\mathbf{b}_i$  ensure a desired level of heterogeneity  $\zeta^2$  and distance between optimum and initialization  $r_0$ . Appendix D.2.4 describes the generation of these quadratics in detail.

**Scalability on rings and trees** Using these quadratics, fig. 5.2 studies the number of steps required to reach a suboptimality  $f(\bar{\mathbf{x}}) - f(\mathbf{x}^*) \leq \epsilon$  with tuned constant learning rates. On *ring* topologies with uniform  $(1/3)$  gossip weights (and chains for RelaySum), all compared methods require steps at least linear in the number of workers to reach the target quality. RelaySGD and  $D^2$  empirically scale significantly better than Gradient Tracking, these methods are all independent of data heterogeneity. On a *balanced binary tree network* with Metropolis-Hastings weights [Xiao and Boyd, 2004], both  $D^2$  and Gradient Tracking notably do not scale better than on a ring, while RelaySGD on these trees requires only a number of steps logarithmic in the number of workers. SGP with their time-varying exponential topology scales well, too, but it requires more steps on more heterogeneously distributed data.

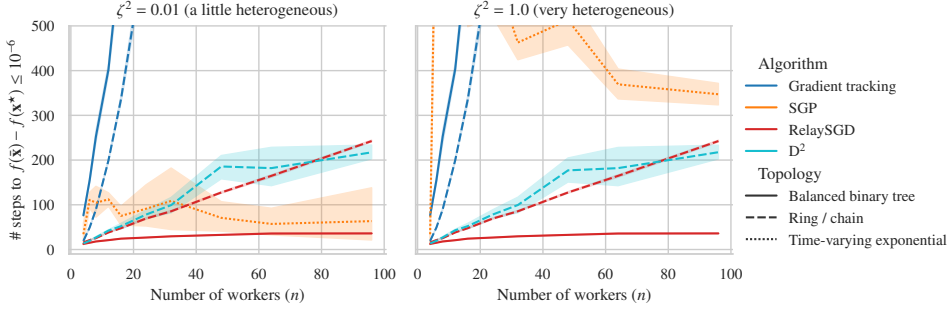


Figure 5.2: Time required to optimize random quadratics ( $\sigma^2 = 0, r_0 = 10, L = 1, \mu = 0.5$ ) to suboptimality  $\leq 10^{-6}$  with varying numbers of workers with tuned constant learning rates. On a ring (---),  $\blacksquare$   $D^2$  and  $\blacksquare$  RelaySGD require steps linear in the number of workers, and this number is *independent of the data heterogeneity*. RelaySGD reduces this to  $\log n$  on a balanced tree topology (—), but trees do not improve  $\blacksquare$   $D^2$  or  $\blacksquare$  Gradient Tracking. For  $\blacksquare$  SGP with time-varying exponential topology (.....), the number of steps does not consistently grow with more workers, but it increases with more heterogeneity (left v.s. right plot).

### 5.6.2 Spanning trees compared to other topologies

RelaySGD cannot utilize all available edges in arbitrary networks to communicate, but is restricted to a spanning tree of the graph. We empirically find that this restriction is not limiting. In fig. 5.3, we take an organic social network topology based on the Davis Southern Women graph [Davis et al., 1930] from NetworkX [Hagberg et al., 2008], and construct random spanning trees found by the Spanning Tree Protocol [Perlman, 1985]. On any such spanning tree, RelaySGD optimizes random heterogeneous quadratics as fast as  $D^2$  on the full graph with Metropolis-Hastings weights [Xiao and Boyd, 2004], significantly faster than D-SGD.

For decentralized learning used in a fully-connected data center for communication efficiency, the deep learning experiments below show that RelaySGD on double binary trees outperforms the most popular non-tree-based communication scheme used in decentralized deep learning [Assran et al., 2019].

### 5.6.3 Effect of data heterogeneity in decentralized deep learning

We study the performance of RelaySGD in deep-learning based image- and text classification. While the algorithm is theoretically independent of dissimilarities in training data, other

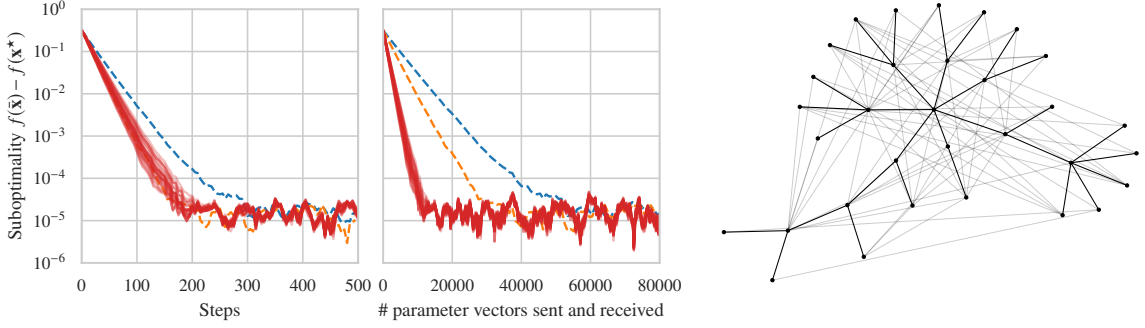


Figure 5.3: Performance of ■ RelaySGD on spanning trees of the Social Network graph (32 nodes) found using Spanning Tree Protocol, compared to ■ D-SGD and ■ D<sup>2</sup> on the full network. Solid lines (—) indicate spanning trees while dashed lines (---) indicate the full graph. The right figure shows one spanning tree on top of the original network. Learning rates are tuned to reach suboptimality  $\leq 10^{-5}$  on random quadratics ( $\zeta^2 = 0.1, \sigma^2 = 0.1, r_0 = 1, L = 1, \mu = 0.5$ ). ■ RelaySGD on spanning trees converges as fast as ■ D<sup>2</sup> on the full network, while the total communication on spanning trees is smaller than on the full graph.

methods (D<sup>2</sup>, RelaySGD/Grad) that have the same property often lose accuracy in the presence of high data heterogeneity [Lin et al., 2021]. To study the dependence of RelaySGD in practical deep learning, we partition training data strictly across 16 workers and distribute the classes using a Dirichlet process [Yurochkin et al., 2019, Lin et al., 2021]. The Dirichlet parameter  $\alpha$  controls the heterogeneity of the data across workers.

We compare RelaySGD against a variety of other algorithms. D-SGD [Lian et al., 2017] is the most natural combination of SGD with gossip averaging, and we chose D<sup>2</sup> [Tang et al., 2018b] to represent the class of previous work that is theoretically robust to heterogeneity. We extend D<sup>2</sup> to allow varying step sizes and local momentum, according to Appendix D.4.4, and make it suitable for practical deep learning. Although Stochastic Gradient Push [Assran et al., 2019] is not theoretically independent of data heterogeneity, it is a popular choice in the data center setting, where they use a time-varying exponential scheme on  $2^d$  workers that mixes exactly uniformly in  $d$  rounds (Appendix D.4.6). We also compare to D-SGD with quasi-global momentum [Lin et al., 2021], a practical method recently introduced to increase robustness to heterogeneous data.

Table 5.1 evaluates RelaySGD in the fully-connected data center setting where we limit the communication budget per iteration to two models. We use 16-workers on Cifar-10, following the experimental details outlined in Appendix D.2 and hyperparameter tuning procedure from Appendix D.3. For this experiment, we consider three topologies: (1) double binary trees as described in section 5.4, (2) rings, and (3) the time-varying exponential scheme of Stochastic Gradient Push (SGP) [Assran et al., 2019]. Because SGP normally sends/receives only one model per communication round, we execute two synchronous communication steps per gradient update, increasing its latency. The various algorithms compared have different optimal topology choices. In table 5.1 we only include the optimal choice for each algorithm. Table 5.2 qualitatively compares the possible combinations. We opt for the VGG-11 architecture because it does not feature BatchNorm [Ioffe and Szegedy, 2015]. BatchNorm poses particular challenges to data heterogeneity, and the search for alternatives is an active, and orthogonal, area of research [Liu et al., 2020].

Table 5.1: Cifar-10 [Krizhevsky and Hinton, 2009] test accuracy with the VGG-11 architecture. We vary the data heterogeneity  $\alpha$  [Lin et al., 2021] between 16 workers. Each method sends/receives 2 models per iteration. We use a ring topology for D-SGD [Lian et al., 2017] and  $D^2$  [Tang et al., 2018b] because they perform better on rings than on trees. For D-SGD, we use the quasi-global momentum [Lin et al., 2021] to increase robustness to heterogeneous data. RelaySum with momentum achieves the best results across all levels of data heterogeneity.

Algorithm	Topology (optimal c.f. table 5.2)	$\alpha = 1.00$ (most homogeneous)	$\alpha = 0.1$	$\alpha = .01$ (most heterogeneous)
All-reduce (baseline)	fully connected	87.0% $\rightarrow$	87.0% $\rightarrow$	87.0% $\rightarrow$
+momentum		90.2% $\rightarrow$	90.2% $\rightarrow$	90.2% $\rightarrow$
<b>RelaySGD</b>	binary trees	87.4% $\rightarrow$	86.9% $\rightarrow$	84.6% $\rightarrow$
+ <b>local momentum</b>		90.2% $\rightarrow$	89.5% $\rightarrow$	89.1% $\rightarrow$
D-SGD	ring	87.4% $\rightarrow$	79.9% $\rightarrow$	53.9% $\rightarrow$
+quasi-global mom.		89.5% $\rightarrow$	84.8% $\rightarrow$	63.3% $\rightarrow$
$D^2$	ring	87.2% $\rightarrow$	84.0% $\rightarrow$	38.2% $\rightarrow$
+local momentum		88.2% $\rightarrow$	88.5% $\rightarrow$	61.0% $\rightarrow$
Stochastic gradient push	time-varying exponential	87.4% $\rightarrow$	86.7% $\rightarrow$	86.7% $\rightarrow$
+local momentum		89.5% $\rightarrow$	89.2% $\rightarrow$	87.5% $\rightarrow$

Table 5.2: Motivation of topology choices. For each algorithm, we compare 4 topologies configured to send/receive 2 models at each SGD iteration. The algorithms have different optimal topologies.

Algorithm	Ring	Chain (= spanning tree of ring)	Double binary trees	Time-varying exp.
RelaySGD	Unsupported	Suboptimal (D.5.1)	<b>Best result</b>	Unsupported
D-SGD	<b>Best result</b>	Suboptimal	Suboptimal (D.5.1)	Unsupported
$D^2$	<b>Best result</b>	Suboptimal	Suboptimal (D.5.1)	Unsupported
SGP	Equiv. to D-SGD	Equiv. to D-SGD	Equiv. to D-SGD	<b>Best result</b>

Even though RelaySGD does not use a time-varying topology, it performs as well as or better than SGP, and RelaySGD with momentum suffers minimal accuracy loss up to heterogeneity  $\alpha = 0.01$ , a level higher than considered in previous work [Lin et al., 2021]. While  $D^2$  is theoretically independent of data heterogeneity, and while some of its random repetitions yield good results, it is unstable in the very heterogeneous setting. Moreover, fig. 5.4 shows that workers with RelaySGD achieve high test accuracies quicker during training than with other algorithms.

These findings are confirmed on ImageNet [Deng et al., 2009] with the ResNet-20-EvoNorm architecture [Liu et al., 2020] in table 5.3. On the BERT fine-tuning task from [Lin et al., 2021], table 5.4 demonstrates that RelaySGD with the Adam optimizer, customary for such NLP tasks, outperforms all compared algorithms.

#### 5.6.4 Robustness to unreliable communication

Peer-to-peer applications are a central use case for decentralized learning. Decentralized learning algorithms must therefore be robust to workers joining and leaving, and to unreliable communication between workers. Gossip averaging naturally features such robustness, but

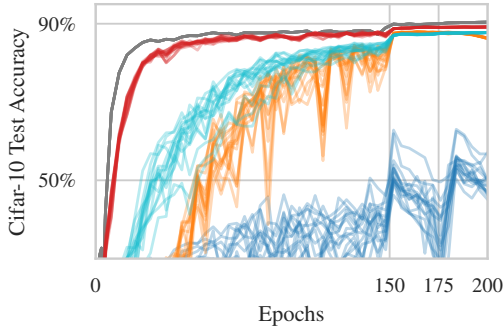


Figure 5.4: Test accuracy during training of 16 workers with heterogeneous data ( $\alpha = 0.01$ ) on Cifar-10. Like, with the  $\blacksquare$  all-reduce baseline, all workers in  $\blacksquare$  RelaySGD on double binary trees quickly reach good accuracy, while this takes longer for  $\blacksquare$  SGP with time-varying exponential topology and  $\blacksquare$  D<sup>2</sup> on a ring.  $\blacksquare$  D-SGD does not reach good accuracy with such heterogeneous data.

Table 5.3: Test accuracies on ImageNet, using 16 workers with heterogeneous data ( $\alpha = 0.1$ ). Even when communicating over a simple chain network, RelaySGD performs similarly to SGP [Assran et al., 2019] with their time-varying exponential communicating scheme. Methods use default learning rates (Appendix D.3.2).

Algorithm	Topology	Top-1 Accuracy
Centralized (baseline)	fully-connected	69.7%
<b>RelaySGD w/ momentum</b>	double binary trees	60.0%
D-SGD w/ quasi-global momentum	ring	55.8%
D <sup>2</sup> w/ momentum	ring	diverged at epoch 65, at 49.5%
SGP w/ momentum	time-varying exponential	58.5%

Algorithm	Topology	Top-1 Accuracy
Centralized Adam	fully-connected	94.2% $\pm$ 0.1%
<b>Relay-Adam</b>	double binary trees	93.2% $\pm$ 0.6%
D-SGD Adam	ring	87.3% $\pm$ 0.6%
Quasi-global Adam	ring	88.3% $\pm$ 0.7%
SGP Adam	time-varying exp.	88.3% $\pm$ 0.3%

Table 5.4: DistilBERT [Sanh et al., 2019] fine-tuning on AG news data [Zhang et al., 2015] using 16 nodes with heterogeneous data ( $\alpha = 0.1$ ). Transformers are usually trained with Adam, and RelaySGD naturally supports Adam updates without the need to synchronize local optimizer state between workers. (Appendix D.2.3).

Table 5.5: Robustness to unreliable networks. On Cifar-10/VGG-11 with 16 workers and heterogeneous data ( $\alpha = 0.01$ ), we compare momentum versions of the best-performing algorithms from table 5.1. Like gossip-based algorithms, RelaySGD with the robust update rule 5.1 can tolerate up to 10% dropped messages and converge to full test accuracy. Without modification, D<sup>2</sup> [Tang et al., 2018b] does not share this property.

Algorithm	Topology	Reliable network	1% dropped messages	10% dropped messages
RelaySGD w/ momentum	trees	89.2%	89.3%	89.3%
D-SGD w/ quasi-global m.	ring	78.3%	76.2%	76.9%
D <sup>2</sup> w/ momentum	ring	87.4%	diverges	diverges
SGP w/ momentum	time-varying	88.5%	88.6%	88.1%



for methods like  $D^2$ , that correct for local data biases, achieving such robustness is non-trivial. As a proxy for these challenges, in table 5.5, we verify that RelaySGD can tolerate randomly dropped messages. The algorithm achieves this by reliably counting the number of models summed up in each message. For this experiment, we use an extended version of Algorithm 5, where line 10 is replaced by

$$\mathbf{x}_i^{(t+1)} = \frac{1}{n} \left( \mathbf{x}_i^{(t+1/2)} + \sum_{j \in \mathcal{N}_i} \mathbf{m}_{j \rightarrow i}^{(t)} + (n - \bar{n}_i^{(t+1)}) \mathbf{x}_i^{(t)} \right). \quad (5.1)$$

We count the number of models received as  $\bar{n}$ , and substitute any missing models ( $< n$ ) by the previous state  $\mathbf{x}_i^{(t)}$ . RelaySGD trains reliably to good test accuracy with up to 10% deleted messages. This behavior is on par with a similarly modified SGP [Assran et al., 2019] that corrects for missing energy. In contrast,  $D^2$  becomes unstable with undelivered messages.

## 5.7 Conclusion

Decentralized learning has great promise as a building block in the democratization of deep learning. Deep learning relies on large datasets, and while large companies can afford those, many individuals together can, too. Of course, their data does not follow the exact same distribution, calling for robustness of decentralized learning algorithms to data heterogeneity. Algorithms with this property have been proposed and analyzed theoretically, but they do not always perform well in deep learning.

In this paper, we propose RelaySGD for distributed optimization over decentralized networks with heterogeneous data. Unlike algorithms based on gossip averaging, RelaySGD *relays* models through spanning trees of a network without decaying their magnitude. This yields an algorithm that is both theoretically independent of data heterogeneity, but also high performing in actual deep learning tasks. With its demonstrated robustness to unreliable communication, RelaySGD makes an attractive choice for peer-to-peer deep learning and applications in large-scale data centers.

### 5.7.1 Applicability to data center training

This thesis focuses on communication-efficient distributed training in a data center setting. The RelaySGD algorithm introduced in this chapter is most useful when workers have heterogeneous data, and it was primarily discussed in this context, but the heterogeneous setting is probably not so relevant in data centers.

We do also observe benefits for RelaySGD in data center training, due to the potential for fast mixing between all workers in the network. Table 5.1, for example, shows that, even in the absence of high heterogeneity, RelaySGD can be a better choice than gossip SGD.

## 5.8 Acknowledgements

This project was supported by SNSF grant 200020\_200342, as well as the DIGIPREDICT project from the European Union, and a Google PhD Fellowship.

We thank Yatin Dandi and Lenka Zdeborová for pointing out the similarities between this algorithm and Belief Propagation during a poster session. This discussion helped us find the strongly related article by Zhang et al. [2019] that we missed initially.

We thank Renee Vogels for proofreading of the manuscript.



## Chapter 6

# Conclusion

As the success of deep learning today largely depends on the size of the models and of the datasets on which these models are trained, training efficiently is more important than ever. Distributed training helps to speed up the process, but communication can be a bottleneck in the scalability of distributed learning systems. This thesis has examined solutions to this communication bottleneck in the form of lossy communication compression and sparse connectivity between workers.

The proposed PowerSGD algorithm for communication compression made strong compression (up to  $\sim 100\times$ ) possible in high-performance training setups based on all-reduce, using compression operations that are efficient on GPU hardware. We then extended this compression scheme to the decentralized learning setting, where the compression algorithm is generalized as a way to partially synchronize models between pairs of connected workers. The key selling point for this algorithm in the decentralized setting is the absence of communication-specific hyperparameters beyond the compression rate.

Our study of the sparse communication paradigm from decentralized learning continued by evaluating the effect of the communication topology on the convergence of the training process. We introduced a framework to reason about the ‘effective number of neighbors’ in a sparse topology, as an alternative to the typical ‘spectral gap’ metric. Apart from a better explanation of the empirical merits topologies, this framework offers a natural extension to time-varying topologies, which are popular in practice [Assran et al., 2019].

Finally, we investigated an alternative communication mechanism to gossip communication for sparsely connected (decentralized) learning. Instead of diffusing gradient updates slowly through a graph, this mechanism relies on efficiently relaying gradients through a spanning tree of the graph. This algorithm is mainly useful when workers have very heterogeneous data distributions, because it ensures that all workers have equal influence on the updates of other workers.

### 6.1 Discussion and future work

Tools for communication efficiency, such as the ones discussed in this thesis, are finding their way into the mainstream of deep learning. The PowerSGD algorithm introduced in chapter 2, for example, was used to train the impressive DALL-E text-to-image model [Ramesh et al., 2021], and sparse communication can reduce the training time in large-scale distributed training [Assran et al., 2019]. PowerSGD is now available as a ‘communication hook’ in Py-

Torch [Contributors, 2020], but these methods are far from becoming being a practical default for distributed training. In practice, the most popular tools for communication efficiency are low- or mixed-precision training [Micikevicius et al., 2018, Kalamkar et al., 2019]. These methods do not reach the same levels of communication compression, but they are easy to implement and unlikely to significantly alter the training process. To enable widespread adoption of the communication-efficient training methods like the ones in this thesis, I believe the following issues are critical to address.

**Implementation efficiency** If the goal of communication compression is to reduce the communication bottleneck in distributed training, it is imperative that the compression algorithm is faster than the communication itself. The PowerSGD algorithm is fast for strong compression rates (low ranks), but the computations can become too expensive when lower compression rates are required to achieve a desired model accuracy [Agarwal et al., 2022, Markov et al., 2021]. Ramesh et al. [2021], who used PowerSGD in production, made significant improvements to the speed of the orthogonalization operation required in PowerSGD, and such low-level optimizations are crucial to make any compression algorithm practical. It is also worth exploring variations on PowerSGD that avoid using a large compression rank, such as replacing few large messages with many small ones, or by cutting up layer gradients into smaller blocks before compression.

Similarly, to make sparse connectivity practical as a replacement for all-reduce averaging, these algorithms should be implemented at the same level of all-reduce primitives [NVIDIA, 2019, Markov et al., 2021] to offer competitive performance.

**Adaptivity without hyperparameters** While the proposed methods for communication compression were designed to require fewer hyperparameters than some baseline methods, these methods still fundamentally require the user to choose the strength of the communication compression. Compressing too much can result in a degradation in model accuracy, while compressing too little can be wasteful and suboptimal in terms of speed. The use of these methods would be much more attractive if this choice was made automatically by the algorithm. There has been significant progress into adaptive communication compression [Agarwal et al., 2021, Alimohammadi et al., 2022], but none of these methods completely removes the need for a user-specified compression rate.

**Guaranteed accuracy preservation** Tying into the previous point, training deep learning models is a process surrounded by uncertainty. In optimization research, we often train models for which we already know what ‘good performance’ is, while in practice, we often do not know what limits the performance. Is it the dataset, the model architecture, or the training procedure? Using non-standard training algorithms that improve communication efficiency would *only* be attractive for the majority of research and exploration in deep learning, if they can guarantee that no model quality is lost by using the method. It should always be safe for a user to ‘turn on’ a communication efficient training algorithm.

## Appendix A

# Appendix for PowerSGD

### A.1 Discussion of convergence

The proof of convergence of EF-SGD with momentum can be derived by incorporating a few key changes to the proof of [Karimireddy et al., 2019b]: i) we are in a multi-worker setting, and ii) we incorporate the techniques introduced by [Ghadimi and Lan, 2016] to handle the additional *momentum*. Further,  $\|\cdot\|^2$  unless otherwise specified is always the standard euclidean norm for vectors, and is the *Frobenius* norm for matrices.

Suppose that we want to minimize a continuous and (possibly) non-convex function  $f: \mathbb{R}^d \rightarrow \mathbb{R}$ :

$$f^* = \min_{\mathbf{x} \in \mathbb{R}^d} f(\mathbf{x}).$$

The classic stochastic gradient algorithm (SGD) [Robbins and Monro, 1951] when adapted to the distributed optimization setting performs iterations of the form

$$\mathbf{x}_{t+1} := \mathbf{x}_t - \gamma \mathbf{g}_t, \text{ where} \tag{A.1}$$

$$\mathbf{g}_t = \frac{1}{W} \sum_{w=1}^W \mathbf{g}_{t,w} \quad \text{and} \quad \mathbb{E}[\mathbf{g}_t] = \nabla f(\mathbf{x}_t).$$

Here  $\gamma \in \mathbb{R}$  is the step-size (or learning-rate) and  $\mathbf{g}_{t,w}$  is the stochastic gradient computed by the  $w$ th worker for  $w \in \{1, \dots, W\}$  workers.

Now EF-SGD (Algorithm 2) when run on the  $W$  workers with step-size  $\gamma$  and momentum parameter  $\lambda$  can be rewritten making the dependence on iteration  $t$  explicit as follows:

$$\begin{aligned} \Delta'_t &= \text{DECOMPRESS}(\text{COMPRESS}(\mathbf{g}_t + \mathbf{e}_t)), \\ \mathbf{m}_{t+1} &= \Delta'_t + \lambda \mathbf{m}_t, \\ \mathbf{x}_{t+1} &= \mathbf{x}_t - \gamma(\Delta'_t + \mathbf{m}_{t+1}), \text{ and} \\ \mathbf{e}_{t+1} &= (\mathbf{g}_t + \mathbf{e}_t) - \Delta'_t. \end{aligned} \tag{A.2}$$

#### A.1.1 Eigen compression

**Assumption G (eigen compression)** Consider any matrix  $M = g_t + e_t$  encountered during the run of Algorithm 2 such that  $M$  is of rank  $R$ . Further, suppose that  $\mathcal{C}_r(M)$  is the best rank- $r$

approximation of  $M$  i.e.,

$$\mathcal{C}_r(M) = \arg \min_C \|M - C\|^2.$$

Then we assume that there exists a  $\delta_{\mathbf{e},r} > 0$  such that

$$\|M - \mathcal{C}_r(M)\|^2 \leq (1 - \delta_{\mathbf{e},r})\|M\|^2 \text{ a.s.}$$

We state the below standard fact from linear algebra.

**Remark 4 (best rank- $r$  approximation)** Suppose we are given a matrix  $M$  of rank  $n$  whose singular value decomposition is

$$M = \sum_{i=1}^n \sigma_i \mathbf{u}_i \mathbf{v}_i^\top,$$

where the singular-values  $(\sigma_i)$  are sorted in descending order. Then the best rank- $r$  approximation of  $M$  for  $r \leq n$  is

$$\mathcal{C}_r(M) = \left( \sum_{i=1}^r \sigma_i \mathbf{u}_i \mathbf{v}_i^\top \right) Q,$$

where  $Q \in \mathbb{R}^{r \times r}$  is an orthogonal matrix, and further the quality of its approximation is bounded by

$$\|M - \mathcal{C}_r(M)\|^2 = \left( 1 - \frac{\sum_{i=1}^r \sigma_i^2}{\sum_{i=1}^n \sigma_i^2} \right) \|M\|^2.$$

Thus, if we used Algorithm 2 with exact rank- $r$  approximation of the gradients, we would converge at rate dictated by the eigenspectrum of the gradients. If the singular values are ‘top-heavy’, i.e., the largest  $r$  values are significantly larger than the rest, then a rank- $r$  approximation is quite accurate. As demonstrated in [Wang et al., 2018], the eigenspectrum of stochastic gradients in common deep learning tasks is indeed ‘top-heavy’. Thus, we can expect  $\delta_{\mathbf{e},r}$  to be bounded away from 0 even for very small  $r$  (e.g., 1 or 2). Of course computing the actual top eigenvectors of the stochastic gradients is very computationally expensive, and more-over is not linear (and hence does not support *reduce*).

### A.1.2 Subspace iteration

The key innovation in PowerSGD is to use only a *single* step of subspace (or power) iteration to give a fast low rank approximation [Stewart and Miller, 1975] to the given matrix, which in our case is a stochastic gradient. However, a single step of subspace iteration in general does not result in an adequate low-rank approximation of the input matrix. To combat this, and to at the same time reduce the variance of the stochastic gradient approximation compared to the full (deterministic) gradient, we propose the *reuse* of the low-rank approximation from the previous iteration as the starting point for the current iteration. This is in spite of the target matrices which are trying to approximate are *changing*, as the parameters evolve. Nevertheless, reuse here is justified because the full gradient does not change very fast (the gradient is Lipschitz by assumption) and we only perform a tiny update at each step, so can be assumed to be stationary within few steps. Intuitively, by linearity of the subspace operation, the sequence of subspace steps with the reuse then is converging to the eigenvector of the averaged stochastic gradients over these steps, thus having a lower variance than the analogue without re-use, which has no such averaging effect.

For simplicity, we assume all matrices to be square and symmetric in this subsection. These insights can be generalized to arbitrary matrices but with a substantial increase in complexity of exposition. Here, we simply note that for any non-square matrix  $A$ , we can instead consider

$$\tilde{A} = \begin{bmatrix} 0 & A \\ A^\top & 0 \end{bmatrix}$$

which is symmetric and has the same eigenvectors and eigenvalues as the original matrix  $A$ —see [Stewart, 1976] for more details on handling such cases.

We can now state an informal theorem about the convergence of subspace iteration.

**Theorem VI** Suppose that we run subspace iteration as in (A.3) on a fixed matrix  $A_t = M$ . Also let  $M = \sum_{i=1}^n \sigma_i \mathbf{u}_i \mathbf{u}_i^\top$  be the eigendecomposition of  $M$  with  $\sigma_1 \geq \dots \sigma_r > \sigma_{r+1} \geq \dots \geq \sigma_n$ . Then there exists an orthogonal matrix  $Q \in \mathbb{R}^{r \times r}$  such that

$$\lim_{t \rightarrow \infty} X_t = [\mathbf{u}_1, \dots, \mathbf{u}_r] Q.$$

In other words, (A.3) recovers the best rank- $r$  approximation of  $M$  as long as there is a gap between the  $\sigma_r$  and  $\sigma_{r+1}$  eigenvalues.

Suppose that at each iteration we receive a matrix  $A_t \in \mathbb{R}^{n \times n}$  whose expectation is the same fixed matrix  $M \in \mathbb{R}^{n \times n}$ . Starting from an orthonormalized  $X_0 \in \mathbb{R}^{n \times r}$ , i.e.,  $X_0^\top X_0 = I_r$ , the rank- $r$  subspace iteration algorithm performs the following update:

$$X_{t+1} = \text{ORTHOGONALIZE}(A_t X_t). \quad (\text{A.3})$$

The final output of the algorithm (the matrix approximation) is  $(A_{T+1} X_T) X_T^\top$ . This closely resembles the method of PowerSGD as outlines in Algorithm 1. We recommend [Arbenz, 2016] for an in-depth analysis of the (non-varying) subspace iteration algorithm.

**Remark 5 (orthogonalization is a linear operation)** We recall some more facts from linear algebra. For any square matrix  $B$ , there exists an orthogonal matrix  $Q$  and a triangular matrix  $R$  such that  $QQ^\top = I$  and  $B = QR$ . This is true, e.g., if we use Gram–Schmidt procedure to orthonormalize  $B$ : Suppose  $\text{ORTHOGONALIZE}(B)$  uses the Gram–Schmidt procedure to orthogonalize  $B$ . Then there exists a triangular matrix  $R$  such that

$$\text{ORTHOGONALIZE}(B) = BR^{-1}.$$

*Proof.* It is easy to see that for any orthogonal matrix  $Q$ , the matrix  $[\mathbf{u}_1, \dots, \mathbf{u}_r]Q$  is also orthogonal, and further is the fixed point of (A.3). In fact all rank- $r$  matrices which are fixed points of (A.3) are of this form.

We will use the observation in Remark 5 to rewrite the update (A.3) in a more convenient fashion. There exist triangular matrices  $R_0, \dots, R_t$  such that

$$X_{t+1} = \text{ORTHOGONALIZE}(A_t X_t) = A_t X_t R_t^{-1} = (A_t A_{t-1} \dots A_0) X_0 (R_0^{-1} R_1^{-1} \dots R_t^{-1}).$$

Thus,  $X_{t+1}$  can alternatively be written as

$$X_{t+1} = \text{ORTHOGONALIZE}((A_t A_{t-1} \dots A_0) X_0) = \text{ORTHOGONALIZE}(M^{t+1} X_0).$$

Here we assumed that the matrix was fixed, i.e.,  $A_t = M$ . Let us further assume that  $X_0$  has a non-zero support on the first  $r$  eigenvectors of  $M$ . Then, a gap in the eigenvalues  $\sigma_r > \sigma_{r+1}$  implies that  $\text{ORTHOGONALIZE}(M^{t+1} X_0)$  converges to  $[\mathbf{u}_1, \dots, \mathbf{u}_r]Q$ . We refer to Chapter 7.2 of [Arbenz, 2016] for the actual proof of this fact.  $\square$

### A.1.3 Single/multi worker equivalence

The difference between the update as written in (A.2) and Algorithm 2 is that the error computation and compression is performed on the *aggregated* gradient  $\mathbf{g}_t$  instead of on the individual workers' gradients  $\mathbf{g}_{t,w}$ . While in general these are not equivalent, the linearity of PowerSGD ensures that these are indeed equivalent. This implies that PowerSGD has the neat property that the algorithm is equivalent if run on  $W$  workers or a single worker with a larger batch-size. This does not hold for most other schemes (e.g., sign based compression schemes, QSGD, etc.).

**Lemma 6 (equivalence of single worker and multi worker updates)** The updates in PowerSGD (i.e., Algorithm 2 using Compressor 1) are equivalent to the updates (A.2).

*Proof.* Consider the update performed by PowerSGD for arbitrary vectors  $\{\mathbf{v}_w\}$ . Let  $\mathcal{C}(\mathbf{v}_w)$  be the compressed version of  $\mathbf{v}_w$  for  $w \in \{1, \dots, W\}$ . Then, by design of PowerSGD, the following holds:

$$\text{DECOMPRESS}(\text{AGGREGATE}(\mathcal{C}(\mathbf{v}_1), \dots, \mathcal{C}(\mathbf{v}_W))) = \text{DECOMPRESS}(\mathcal{C}(\frac{1}{W} \sum_w \mathbf{v}_w)).$$

This implies that running the algorithm on multiple workers, or running it on a single worker with a larger batch-size is identical. In particular,

$$\begin{aligned} & \text{DECOMPRESS}(\text{AGGREGATE}(\mathcal{C}(\mathbf{g}_{t,1} + \mathbf{e}_{t,1}), \dots, \mathcal{C}(\mathbf{g}_{t,W} + \mathbf{e}_{t,W}))) \\ &= \text{DECOMPRESS}(\mathcal{C}(\frac{1}{W} \sum_w \mathbf{g}_{t,w} + \mathbf{e}_{t,w})) \\ &= \text{DECOMPRESS}(\frac{1}{W} \mathcal{C}(\mathbf{g}_t + \mathbf{e}_t)). \end{aligned}$$

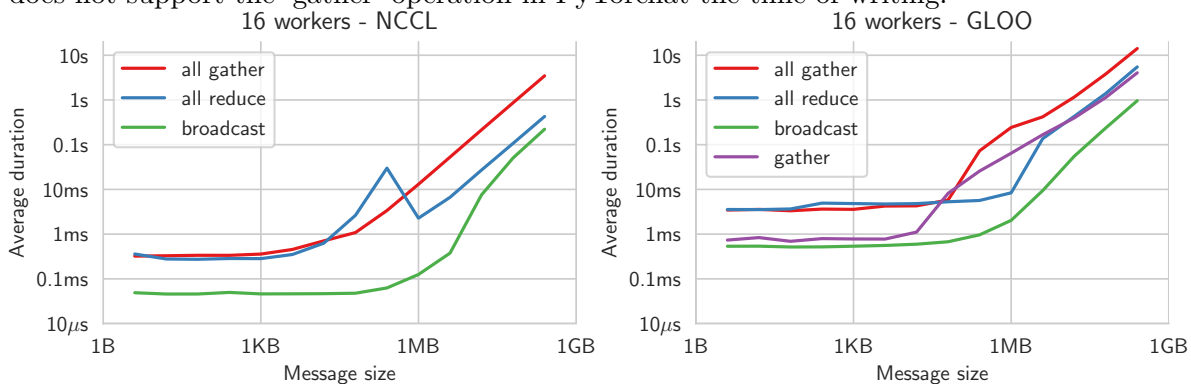
□



## A.2 Cluster specifications

- 8 nodes
- GPUs:  $2 \times$  Nvidia GeForce GTX Titan X with 12 GB memory per node
- GPU connection: traversing PCIe and the SMP interconnect between NUMA nodes
- CPU: Intel Xeon E5-2680 v3 @ 2.50Ghz, 48 cores
- System memory: 251 GiB
- Ethernet: 10Gbit/s SFI/SFP+
- *Fat tree* network topology
- Running PyTorch 1.1 on Anaconda Python 3.7

**Timings of collective communication operations** The figure below shows timings for the NCCL backend, which is the default in our experiments, and the GLOO backend. Note that NCCL does not support the ‘gather’ operation in PyTorch at the time of writing.



### A.3 Convergence curves

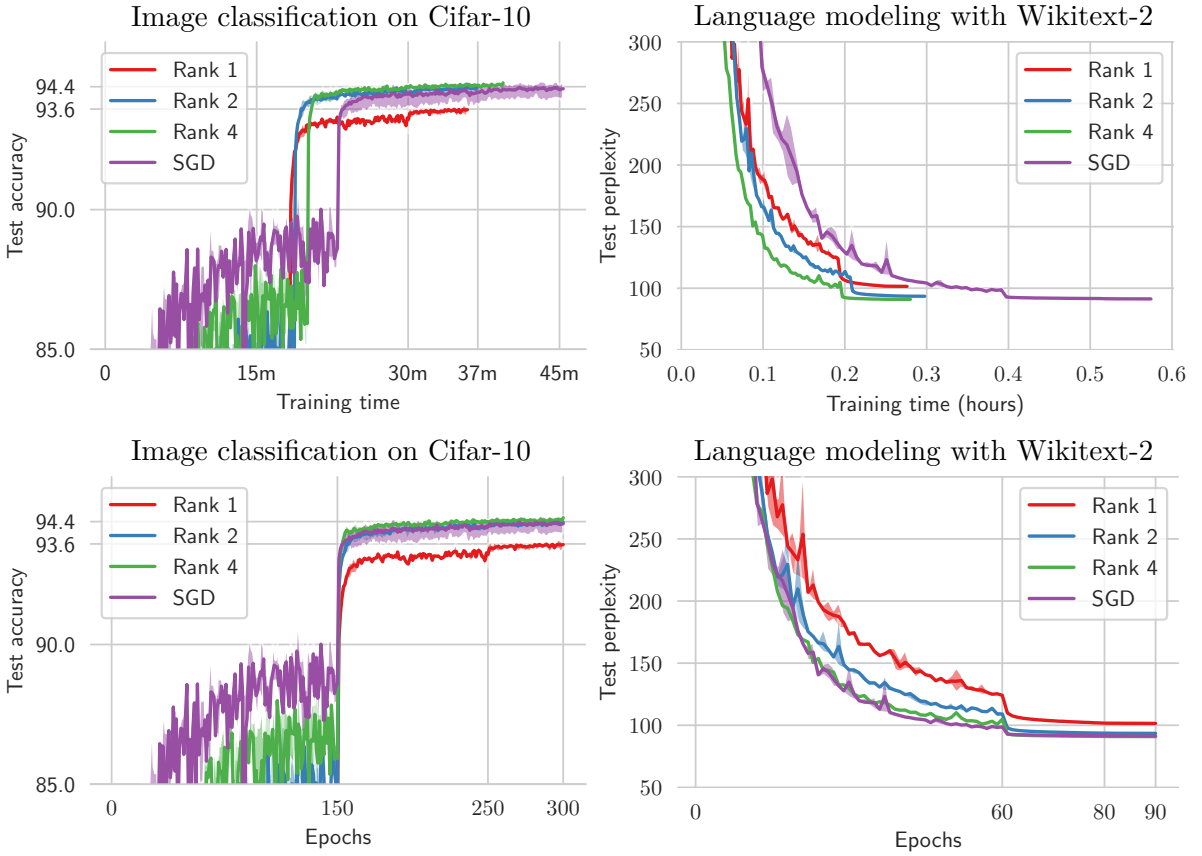


Figure A.1: Convergence curves of PowerSGD with varying rank. This figure is meant to give context to the final results and timings presented in Table 2.3. In two different tasks, PowerSGD with high enough rank can achieve the test quality of full-precision SGD with lower wall-clock duration. Contrary to Table 2.3, these timings include testing overhead at the end of each epoch, checkpointing, and other bookkeeping. Shaded areas show the min—max values over 3 replications of the experiments.

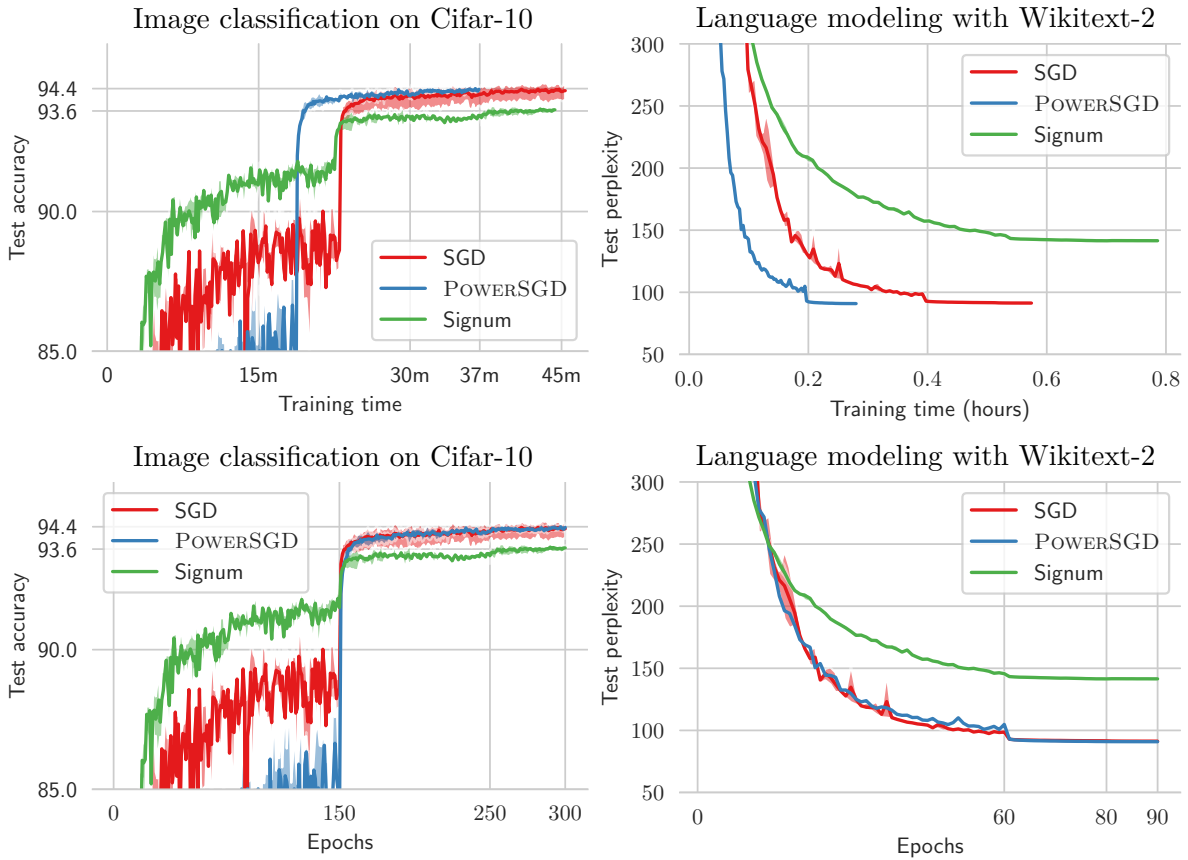


Figure A.2: Convergence curves comparing PowerSGD to the Signum optimizer Bernstein et al. [2019] (with tuned learning rate). Out of the compared methods, Signum came out as the most competitive. This figure is meant to give context to the final results and timings presented in Table 2.6. Contrary to Table 2.3, these timings include testing overhead at the end of each epoch, checkpointing, and other bookkeeping. Shaded areas show the min—max values over 3 replications of the experiments.

## A.4 Language modeling with transformers

In this case study, we assess PowerSGD’s universality and ease of tuning. We implemented PowerSGD communication in Facebook AI Research’s `fairseq` library [Ott et al., 2019]. We trained fairseq’s language modeling example<sup>1</sup> with transformers [Baevski and Auli, 2019] on Google’s public cloud. The communication infrastructure, hardware, number of workers (32), and model architecture are all different from any experiments we have conducted before. See Table A.1 for details.

The results of our experiments for various ranks are shown in fig. A.3 and Table A.2. For this task, we need a higher rank than previously (32 vs 4) to achieve a validation loss competitive to uncompressed SGD. We hypothesize this may be due differences in architecture to the cosine learning rate schedule. Nevertheless, even at this higher rank, we achieve a time-to-accuracy (to loss = 5) of around  $1.5\times$  and a compression ratio of  $14\times$ . These numbers could probably be further improved by re-tuning learning-rate-related hyperparameters.

Table A.1: Experimental setting for the experiments in Appendix A.4

Dataset	WikiText-103
Architecture	Transformer-based [Baevski and Auli, 2019]
Framework & defaults	<a href="https://github.com/pytorch/fairseq/tree/920b85d4bd39e181229db5639c701c854c83ec5c/examples/language_model">https://github.com/pytorch/fairseq/tree/920b85d4bd39e181229db5639c701c854c83ec5c/examples/language_model</a>
Number of workers	32
Backend	NCCL (fastest in PyTorch)
Hardware	n1-standard-8 nodes on Google Cloud with 1 Nvidia Tesla K80 GPU
Hyperparameters	Taken from the example, not re-tuned, with minor changes for the higher number of workers and different GPU memory:
<code>lr period updates</code>	16875
<code>max update</code>	17875
<code>max tokens (valid)</code>	1536 (to fit on a K80 GPU)
<code>tokens per sample</code>	1536 (to fit on a K80 GPU)
<code>warm-up updates</code>	1000
<code>update freq</code>	[1] — don’t aggregate multiple mini-batches locally
Optimizer	original: Nesterov accelerated gradient, we just added PowerSGD for communication
Learning rate	original cosine schedule from the example
Float precision	32-bit (16-bit is unavailable on the K80)
Repetitions	1

<sup>1</sup>[https://github.com/pytorch/fairseq/tree/920b85d4bd39e181229db5639c701c854c83ec5c/examples/language\\_model](https://github.com/pytorch/fairseq/tree/920b85d4bd39e181229db5639c701c854c83ec5c/examples/language_model)

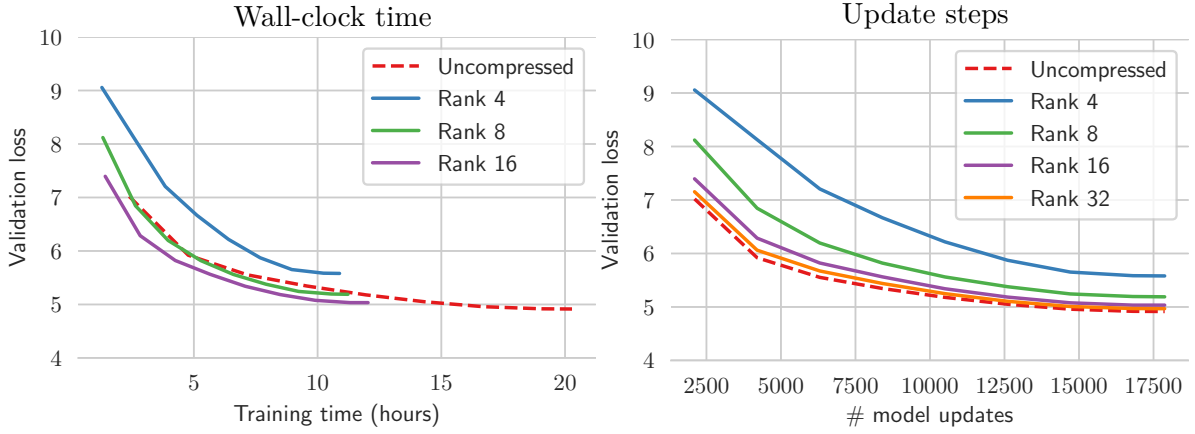





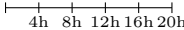





Figure A.3: Language Modeling on Wikitext-2 with Transformers. With a large enough rank, PowerSGD can roughly match the validation loss of full-precision SGD in the same number of iterations. A speedup of  $1.5\times$  in time-to-accuracy (loss=5) is achieved with a rank of 16.

Table A.2: PowerSGD for Language Modeling with Transformers. With rank 32, PowerSGD achieves similar validation loss to uncompressed SGD in the same number of update steps. At this rank, the compression ratio is  $14\times$ , and we can train the model in 12h compared to 20h for the baseline.

Compression	Total training time for 17875 updates	Compression ratio	Validation loss at 17875 updates
Uncompressed	 20h	$1\times$	4.92
Rank 4	 11h	$105\times$	5.58
Rank 8	 11h	$55\times$	5.19
Rank 16	 12h	$28\times$	5.03
Rank 32	 13h	$14\times$	4.97
			

 Forward pass
  Backward pass
  Gradient exchange including computation

## A.5 The need for error feedback

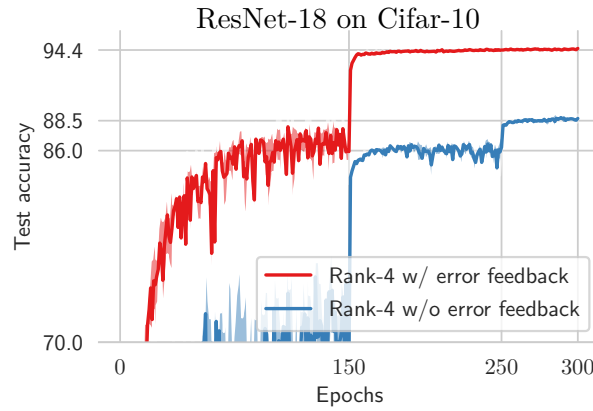


Figure A.4: PowerSGD with and without error feedback compared. While rank-4 PowerSGD achieves the same test accuracy as full-precision SGD, the same method without error feedback does not converge to a good accuracy at all. Both experiments use the same learning rate that was tuned for full-precision SGD.

## A.6 Network parameters

See Table A.3 and Table A.4 for an overview of parameters in the models used.

Table A.3: Parameters in the ResNet-18 architecture and their shapes. The table shows the per-tensor compression ratio achieved by rank- $r$  PowerSGD.

Parameter	Gradient tensor shape	Matrix shape	Uncompressed	Compression
layer4.1.conv2	$512 \times 512 \times 3 \times 3$	$512 \times 4608$	9216 KB	$461/r \times$
layer4.0.conv2	$512 \times 512 \times 3 \times 3$	$512 \times 4608$	9216 KB	$461/r \times$
layer4.1.conv1	$512 \times 512 \times 3 \times 3$	$512 \times 4608$	9216 KB	$461/r \times$
layer4.0.conv1	$512 \times 256 \times 3 \times 3$	$512 \times 2304$	4608 KB	$419/r \times$
layer3.1.conv2	$256 \times 256 \times 3 \times 3$	$256 \times 2304$	2304 KB	$230/r \times$
layer3.1.conv1	$256 \times 256 \times 3 \times 3$	$256 \times 2304$	2304 KB	$230/r \times$
layer3.0.conv2	$256 \times 256 \times 3 \times 3$	$256 \times 2304$	2304 KB	$230/r \times$
layer3.0.conv1	$256 \times 128 \times 3 \times 3$	$256 \times 1152$	1152 KB	$209/r \times$
layer2.1.conv2	$128 \times 128 \times 3 \times 3$	$128 \times 1152$	576 KB	$115/r \times$
layer2.1.conv1	$128 \times 128 \times 3 \times 3$	$128 \times 1152$	576 KB	$115/r \times$
layer2.0.conv2	$128 \times 128 \times 3 \times 3$	$128 \times 1152$	576 KB	$115/r \times$
layer4.0.shortcut.0	$512 \times 256 \times 1 \times 1$	$512 \times 256$	512 KB	$171/r \times$
layer2.0.conv1	$128 \times 64 \times 3 \times 3$	$128 \times 576$	288 KB	$105/r \times$
layer1.1.conv1	$64 \times 64 \times 3 \times 3$	$64 \times 576$	144 KB	$58/r \times$
layer1.1.conv2	$64 \times 64 \times 3 \times 3$	$64 \times 576$	144 KB	$58/r \times$
layer1.0.conv2	$64 \times 64 \times 3 \times 3$	$64 \times 576$	144 KB	$58/r \times$
layer1.0.conv1	$64 \times 64 \times 3 \times 3$	$64 \times 576$	144 KB	$58/r \times$
layer3.0.shortcut.0	$256 \times 128 \times 1 \times 1$	$256 \times 128$	128 KB	$85/r \times$
layer2.0.shortcut.0	$128 \times 64 \times 1 \times 1$	$128 \times 64$	32 KB	$43/r \times$
linear	$10 \times 512$	$10 \times 512$	20 KB	$10/r \times$
conv1	$64 \times 3 \times 3 \times 3$	$64 \times 27$	7 KB	$19/r \times$
Bias vectors (total)			38 KB	None
<b>Total</b>			43 MB	$243/r \times$

Table A.4: Parameters in the LSTM architecture and their shapes. The table shows the per-tensor compression ratio achieved by rank- $r$  PowerSGD.

Parameter	Gradient tensor shape	Matrix shape	Uncompressed	Compression
encoder	$28869 \times 650$	$28869 \times 650$	73300 KB	$636/r \times$
rnn-ih-l0	$2600 \times 650$	$2600 \times 650$	6602 KB	$520/r \times$
rnn-hh-l0	$2600 \times 650$	$2600 \times 650$	6602 KB	$520/r \times$
rnn-ih-l1	$2600 \times 650$	$2600 \times 650$	6602 KB	$520/r \times$
rnn-hh-l1	$2600 \times 650$	$2600 \times 650$	6602 KB	$520/r \times$
rnn-ih-l2	$2600 \times 650$	$2600 \times 650$	6602 KB	$520/r \times$
rnn-hh-l2	$2600 \times 650$	$2600 \times 650$	6602 KB	$520/r \times$
Bias vectors (total)			174 KB	None
<b>Total</b>			110 MB	$310/r \times$

## A.7 Compressor implementation details

### A.7.1 Random Block

This implements compression for error feedback with momentum (Algorithm 2).

---

**Algorithm 6** Random Block compression

---

```

1: function COMPRESS(update matrix  $M \in \mathbb{R}^{n \times m}$ )
2:   Treat  $M$  as a vector of length  $nm$ .
3:   Sample an index  $s$  uniformly between 0 and  $nm - 1$ , using the same seed on all workers.
4:   The block length  $b$  is set to  $(m + n)r$  to match rank- $r$  PowerSGD.
5:   return A consecutive memory slice  $S = M(s : s + b)$ .
6: end function
7: function AGGREGATE+DECOMPRESS(worker's slices  $S_1 \dots S_W$ )
8:    $\hat{M} \leftarrow \mathbf{0} \in \mathbb{R}^{n \times m}$ 
9:    $\hat{M}(s : s + b) \leftarrow \frac{1}{W} \sum_{i=1}^W S_i$  ▷ using all-reduce
10:  return  $\hat{M}$ 
11: end function

```

---

### A.7.2 Random K

This implements compression for error feedback with momentum (Algorithm 2).

---

**Algorithm 7** Random  $K$  compression

---

```

1: function COMPRESS(update matrix  $M \in \mathbb{R}^{n \times m}$ )
2:   Treat  $M$  as a vector of length  $nm$ .
3:   The number of samples  $b$  is set to  $(m + n)r$  to match rank- $r$  PowerSGD.
4:   Sample a set of  $b$  indices  $I$  without replacement, using the same seed on all workers.
5:   return Looked up values  $S = M(I)$ .
6: end function
7: function AGGREGATE+DECOMPRESS(worker's values  $S_1 \dots S_W$ )
8:    $\hat{M} \leftarrow \mathbf{0} \in \mathbb{R}^{n \times m}$ 
9:    $\hat{M}(I) \leftarrow \frac{1}{W} \sum_{i=1}^W S_i$  ▷ using all-reduce
10:  return  $\hat{M}$ 
11: end function

```

---

**Sampling of indices** We sample random indices on the CPU using NumPy. This operation is relatively expensive. Together with the many random lookups, this explains why Random  $K$  compression is significantly slower than Random Block compression.



### A.7.3 Sign+Norm

This implements compression for error feedback with momentum (Algorithm 2).

---

**Algorithm 8** Sign+Norm compression
 

---

```

1: function COMPRESS(update matrix  $M \in \mathbb{R}^{n \times m}$ )
2:   Compute the signs  $S \in \{-1, 1\}^{n \times m}$  of  $M$ 
3:   Compute the  $L_1$  norm  $\ell$  of  $M$ .
4:   return  $(\ell, S)$ 
5: end function
6: function AGGREGATE+DECOMPRESS(worker's norms  $\ell_1 \dots \ell_W$  and signs  $S_1 \dots S_W$ )
7:   return  $\frac{1}{W} \sum_{i=1}^W \frac{\ell_i}{nm} S_i$  ▷ Executed on all workers using NCCL's all-gather
8: end function

```

---

Because PyTorch does not natively support data types smaller than 8 bits per scalar, we use a C++ extension [Bernstein et al., 2019] to actually send single bits to other workers. The employed all-gather operation from NCCL is faster than aggregation using a parameter server using GLOO. We cannot implement a parameter server in NCCL due to lack of a ‘gather’ operation.

### A.7.4 Top K

This implements compression for error feedback with momentum (Algorithm 2).

---

**Algorithm 9** Top  $K$  compression
 

---

```

1: function COMPRESS(update matrix  $M \in \mathbb{R}^{n \times m}$ )
2:   Treat  $M$  as a vector of length  $nm$ .
3:   The number of samples  $b$  is set to  $(m+n)r$  to match rank- $r$  PowerSGD.
4:   Construct a list of  $b$  indices  $I$  corresponding to the top absolute values in  $M$ .
5:   return Looked up values  $S = M(I)$  and indices  $I$ .
6: end function
7: function AGGREGATE+DECOMPRESS(worker's values  $S_1 \dots S_W$  and indices  $I_1 \dots I_W$ )
8:    $\hat{M} \leftarrow \mathbf{0} \in \mathbb{R}^{n \times m}$ 
9:   for worker index  $i$  in  $1, \dots, W$  do
10:     $\hat{M}(I_i) \leftarrow \frac{1}{W} S_i$  ▷ using all-gather in NCCL
11:   end for
12:   return  $\hat{M}$ 
13: end function

```

---

The employed all-gather operation from NCCL is faster than aggregation using a parameter server using GLOO. We cannot implement a parameter server in NCCL due to lack of a ‘gather’ operation.

### A.7.5 Signum

This is our implementation of the Signum compression algorithm by Bernstein et al. [2019]. We run it in its original form, without error feedback, with momentum of 0.9, and a learning rate tuned based on 5 experiments in the 16-worker setting.

---

**Algorithm 10** Signum compression

---

```

1: function COMPRESS(update matrix  $M \in \mathbb{R}^{n \times m}$ )
2:   Compute the signs  $S \in \{-1, 1\}^{n \times m}$  of  $M$ 
3:   return  $S$ 
4: end function
5: function AGGREGATE+DECOMPRESS(worker's signs  $S_1 \dots S_W$ )
6:   return  $\text{SIGN}(\sum_{i=1}^W S_i)$   $\triangleright$  Majority vote, on all workers using NCCL's all-gather
7: end function

```

---

Because PyTorch does not natively support data types smaller than 8 bits per number, we use a C++ extension [Zhao, 2019] to actually send single bits to other workers. The employed all-gather operation from NCCL is faster than aggregation using a parameter server using GLOO. We cannot implement a parameter server in NCCL due to lack of a ‘gather’ operation.

### A.7.6 Atomo

This is our implementation of the Spectral Atomo algorithm presented by Wang et al. [2018]. We run it in its original form, without error feedback, with momentum of 0.9, and a learning rate tuned based on 4 experiments in the 16-worker setting.

**Matrix shape** Atomo differs from PowerSGD in how it treats tensors as matrices. This results in lower compression at the same rank.

**Number of sampled components** Atomo decomposes gradient matrices  $M$  using a Singular Value Decomposition into  $M \sim \sum_i U_i S_{ii} V_i^\top$  and importance-samples components from this summation based on probabilities derived from the absolute singular values  $S_{ii}$ . The probabilities are such, that the expected number of samples components is equal to the target rank  $r$ , but there is no guarantee. We modify the algorithm to always use exactly  $r$  components, to allow for faster communication. We achieve this by repeating the sampling procedure until the number of selected components is  $r$ . This has no significant impact on the runtime performance.

**Algorithm 11** Rank- $r$  Spectral-Atomo compression

---

```

1: function COMPRESS(update matrix  $M \in \mathbb{R}^{n \times m}$ )
2:    $U, S, V \leftarrow \text{SVD}(M)$ . ▷ on CPU using NumPy, faster than PyTorch
3:   Compute Atomo probabilities  $p_1 \dots p_k$  from  $S_{11}, \dots, S_{kk}$ . ▷ see [Wang et al., 2018].
4:   Sampling: include index  $i$  independently with probability  $p_i$ .
5:   Repeat sampling until a set of  $r$  indices  $C$  is selected. ▷ our modification (see above)
6:   return  $\{(U_{i:} \cdot S_{ii}/p_i, V_{i:}) \mid i \in C\}$  as two matrices  $U' \in \mathbb{R}^{n \times r}$  and  $V' \in \mathbb{R}^{m \times r}$ .
7: end function
8: function AGGREGATE+DECOMPRESS(rank- $r$  approximations  $(U'_1, V'_1) \dots (U'_W, V'_W)$  for
   each worker)
9:   return  $\sum_{i=1}^W U'_i V'^{\top}_i$  ▷ using all-gather in NCCL
10: end function

```

---

The employed all-gather operation from NCCL is faster than aggregation using a parameter server using GLOO. We cannot implement a parameter server in NCCL due to lack of a ‘gather’ operation.

**A.7.7 Best-approximation PowerSGD**

This variant is the same as PowerSGD (Algorithm 1), but with more steps of subspace iteration, and without reuse of previous steps. We find that 4 steps of subspace iterations (8 matrix multiplications) is enough to converge to the best low-rank approximation of gradient matrices, when measuring final test accuracy achieved by PowerSGD.

## A.8 Performance optimizations

Because we compare timings, we have aimed to optimize all compared optimizers to a similar level. For sign-based methods, we used a publicly available C++ library by Bernstein et al. [2019] to efficiently pack the signs into bitmaps, an operation which is not supported by PyTorch natively. For Atomo, we have benchmarked the SVD operation on the GPU and CPU, and chose the faster CPU implementation. For all methods, we pack all gradient tensors into one flat buffer to reduce the number of communications. Where possible, we overlay communication with computation. Algorithms that do not support all-reduce are implemented using NCCL’s all-gather, which is faster than a parameter server with GLOO.<sup>2</sup>

## A.9 Learning rate tuning

For each task and each optimization algorithm without error feedback, learning rates were tuned separately. For algorithms based on error feedback with momentum, we use the learning rate tuned for SGD.

Learning rates are defined as rates for 1 worker, and scaled linearly with 5-epoch warm-up to the number of workers (16 by default). We tune them in the 16-worker setting.

We determine the best learning rate by comparing test accuracy of one replication after running the full number of epochs. We start training with 3 different learning rates, a factor 2 apart, based on commonly used rates for the optimizer, and if the best learning rate is either the lower or higher end, we extended the range.

For Cifar-10, the rates considered for SGD were [0.05, 0.1, 0.2], we chose 0.1. For rank-2 Spectral Atomo, we considered [0.025, 0.05, 0.1, 0.2] and chose 0.1. For Signum, we considered [2e-5, 5e-5, 1e-4, 2e-4] and chose 5e-5.

For Wikitext-2, the rates considered for SGD were [0.6, 1.25, 2.5, 5, 10], we chose 1.25. For Signum, we considered [2e-4, 1e-1, 5e-5, 1e-5, 1e-6], and chose 1e-5.

We have not tuned the momentum parameter or  $L_2$ , weight decay parameters or learning rate schedule for any experiment.

---

<sup>2</sup>‘reduce’+‘gather’ (parameter server communication) with GLOO takes longer than all-gather with NCCL, as shown in Appendix A.2. NCCL in PyTorch currently lacks support for a ‘gather’ operator.

## Appendix B

# Appendix for PowerGossip

### B.1 Compressed Consensus

(Proof of Theorem I) Recall that the consensus update for each node  $i$  performs (3.4):

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

**Lemma 7 (preserves average)** For every step of (3.4),  $\bar{\mathbf{X}}^{(t)} = \bar{\mathbf{X}}^{(0)}$ .

*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:

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

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

**Lemma 8 (effect of compression)** Assuming (A4) and (A5) hold, the iteration (3.4) satisfies

$$\|\Delta_i^{(t)}\|_F^2 \leq (1 - \delta) \|\Delta_i^{(t-1)}\|_F^2 + \delta \left\| \sum_{j \in [N]} W_{ij} \Delta_j^{(t-1)} \right\|_F^2,$$

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

*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  $\Pi_{ijt}$ . Recall that  $\mathcal{C}_{ijt}$  is a  $\delta$ -approximate linear projection which implies that  $\Pi_{ijt}$  satisfies

$$\mathbb{E}[\Pi_{ijt}] = \mathbb{E}[\Pi_{ijt}^\top] = \mathbb{E}[\Pi_{ijt}^\top \Pi_{ijt}] = \delta I. \quad (\text{B.1})$$

Further, since  $\Pi_{ijt}$  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  $\Delta_i^{(t)} := \mathbf{X}_i^{(t)} - \bar{\mathbf{X}}^{(0)}$ . Our previous expression for  $\mathbf{X}_i^{(t)}$  implies that

$$\Delta_i^{(t)} = \Delta_i^{(t-1)} + \sum_{j \in [n]} W_{ij} \Pi_{ijt} (\Delta_j^{(t-1)} - \Delta_i^{(t-1)}).$$

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

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

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  $\text{Tr}(AB) = \text{Tr}(BA)$  we can simplify the expression as

$$\mathbb{E}[\text{Tr}(\Delta_i^{(t)\top} \Delta_i^{(t)})] \leq (1 - \delta) \text{Tr}(\Delta_i^{(t-1)\top} \Delta_i^{(t-1)}) + \delta \text{Tr}((\sum_{j \in [n]} W_{ij} \Delta_j)^\top (\sum_{j \in [n]} W_{ij} \Delta_j))$$

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

**Lemma 9 (effect of mixing)** Assuming that  $\mathbf{W}$  has a spectral gap  $\rho$  as in (A4) and  $\Delta_i^{(t)} := \mathbf{X}_i^{(t)} - \bar{X}^{(0)}$ , we have

$$\frac{1}{n} \sum_{i \in [n]} \left\| \sum_{j \in [n]} W_{ij} \Delta_j^{(t-1)} \right\|_F^2 \leq (1 - \rho) \frac{1}{n} \sum_{i \in [n]} \|\Delta_i^{(t-1)}\|_F^2.$$

*Proof.* Follows from standard mixing arguments such as in [Xiao and Boyd, 2004].  $\square$

Averaging lemma 8 over the nodes  $i$  and then applying Lemma 9 gives

$$\begin{aligned} \frac{1}{n} \sum_{i \in [n]} \|\Delta_i^{(t)}\|_F^2 &\leq (1 - \delta) \frac{1}{n} \sum_{i \in [n]} \|\Delta_i^{(t-1)}\|_F^2 + \delta \frac{1}{n} \sum_{i \in [n]} \left\| \sum_{j \in [n]} W_{ij} \Delta_j^{(t-1)} \right\|_F^2 \\ &\leq (1 - \delta + \delta(1 - \rho)) \frac{1}{n} \sum_{i \in [n]} \|\Delta_i^{(t-1)}\|_F^2 \\ &= (1 - \rho\delta) \frac{1}{n} \sum_{i \in [n]} \|\Delta_i^{(t-1)}\|_F^2. \end{aligned}$$

This proves the statement of Theorem I.  $\square$

## B.2 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 7), 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.  $\square$

## B.3 Experimental settings

Tables B.1, B.2 and B.3 describe the implementation details of our experiments.

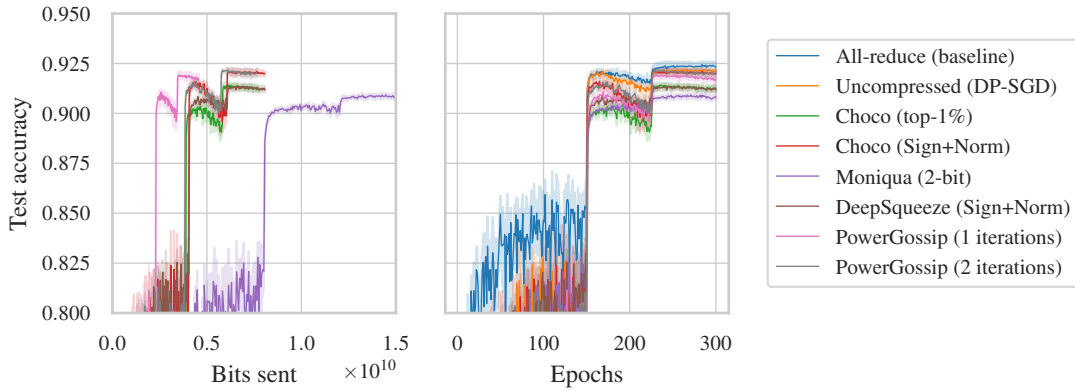
## B.4 Convergence curves

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.

Table B.1: Default experimental settings for Cifar-10/ResNet-20 [based on Koloskova et al., 2019a]

Dataset	Cifar-10
Data augmentation	random horizontal flip and random $32 \times 32$ 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 warm-up	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

#### B.4.1 ResNet-20 on Cifar-10



#### B.4.2 LSTM on WikiText-2

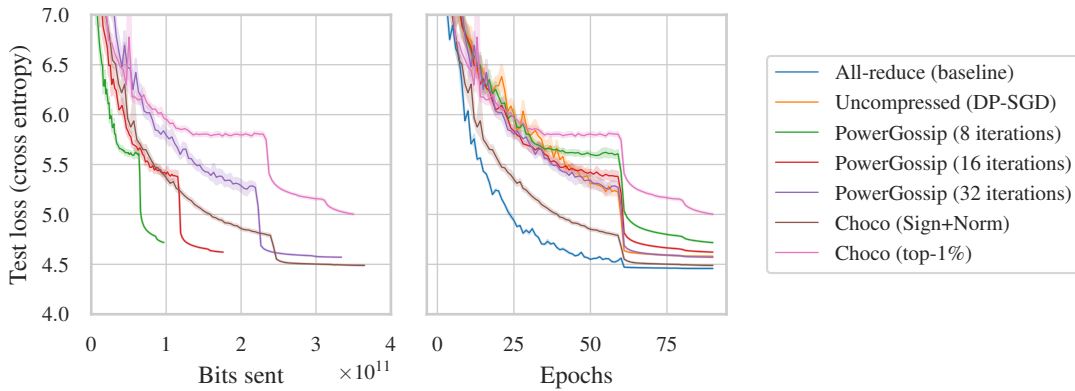




Table B.2: Default experimental settings for WikiText-2 [based on Vogels et al., 2019]

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 warm-up	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 B.3: Experimental settings for Consensus

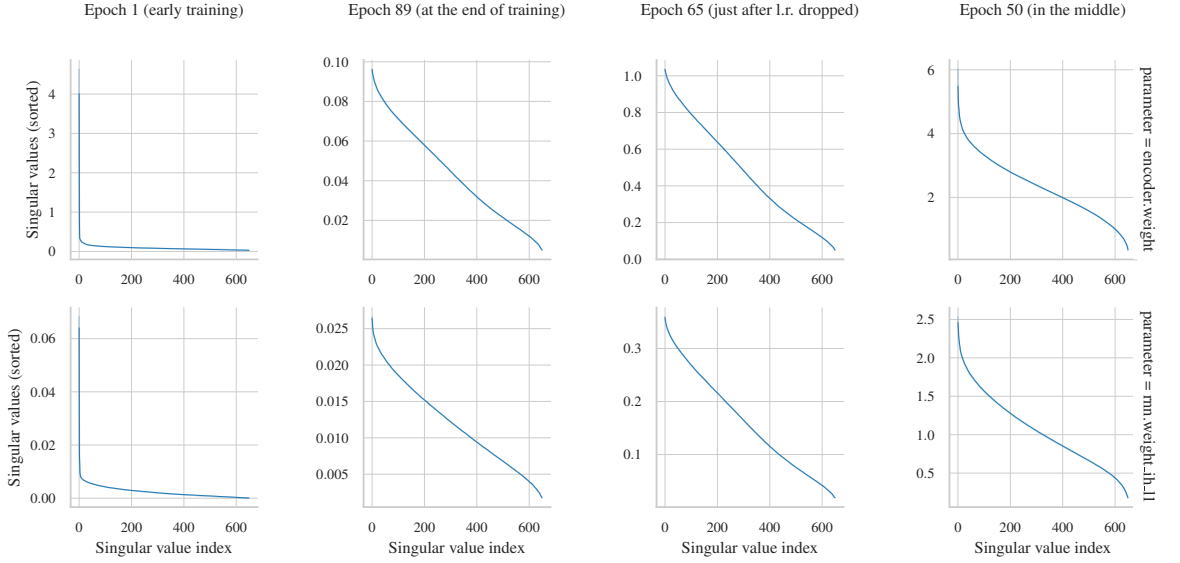
Number of workers	8
Topology	ring
Network $W_{ij}$	0.436 for neighbors $i, j$ , 0.128 if $i = j$ , 0 otherwise (optimized for largest spectral gap)
Data	$100 \times 100$ random normal data or 8 randomly selected $64 \times 64$ faces from [AT&T Laboratories Cambridge]
Objective	minimize $\frac{1}{8} \sum_{i=1}^8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

## B.5 The power spectrum of parameter differences

### B.5.1 LSTM Training

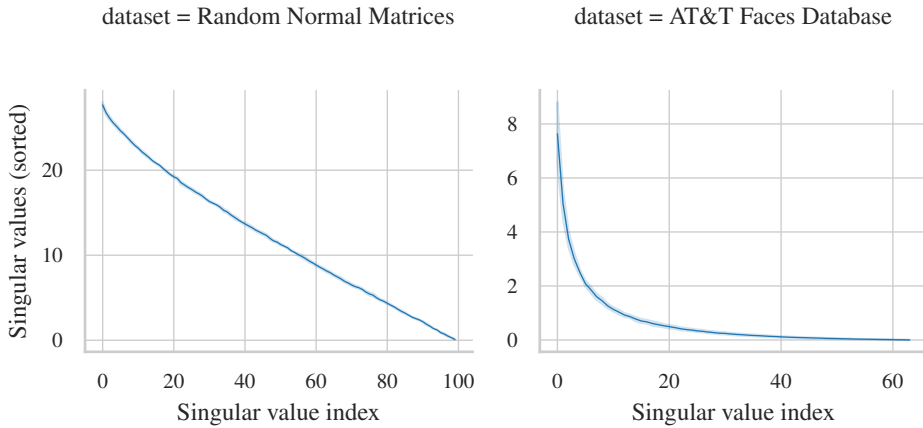
The plots below show the power spectra of parameter differences observed while training the LSTM (Appendix B.9). We train with 16 workers connected in a ring, using PowerGossip with 32 power iterations per gradient update. During training, we record the power spectra of the differences between the parameters 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 carry 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].



### B.5.2 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 fig. 3.1, PowerGossip with power iterations is more efficient per-bit than uncompressed gossip for this dataset.



### B.6 Changing rank vs changing # power iterations

PowerSGD [Vogels et al., 2019] (chapter 2), 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

## B.7 Hyperparameters

### B.7.1 Consensus

In fig. 3.1, we plot results obtained with two compressors in ChocoGossip [Koloskova et al., 2019b], using 20 consensus step size parameters  $\gamma$  ranging from  $7.6 \times 10^{-5}$  to 1 on an exponential grid. The optimal hyperparameter depends on the compressor used.

### B.7.2 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].

Method	Learning rate $\eta$		Consensus rate $\gamma$		Modulo parameter $\theta$	
	Tested	Used	Tested	Used	Tested	Used
All-reduce (baseline)	{0.8, 1.13, 1.6}	1.13				
Uncompressed D-SGD	{0.8, 1.13, 1.6}	1.13				
Choco (top-1%)	{0.96, 1.2, 1.6}*	1.13	{0.025, 0.0375, 0.075, 0.15}*	0.0375		
Choco (Sign+Norm)	{1.2, 1.6, 2.4}*	1.6	{0.15, 0.2, 0.45, 1}*	0.45		
Moniqua (2-bit)	{0.1, 0.2, 0.4, 0.8}	0.4	{0.01, 0.005, 0.0025, 0.0012}†	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}*	0.01		
PowerGossip (1 iteration)		11.3				
PowerGossip (2 iterations)		11.3				

★: based on published parameters and the tuning strategy from Koloskova et al. [2019a].

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

### B.7.3 LSTM on WikiText-2

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

Method	Learning rate $\eta$		Consensus rate $\gamma$		Modulo parameter $\theta$	
	Tested	Used	Tested	Used	Tested	Used
All-reduce (baseline)	{15, 20, 27.5, 35, 47.5}	47.5				
Uncompressed D-SGD	{15, 20, 27.5, 35, 47.5}	47.5				
Choco (top-1%) <sup>†</sup>	{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				

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

## B.8 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 de-anonymization).

### B.8.1 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 D-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).

### B.8.2 DeepSqueeze

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

### B.8.3 Moniqua

Because the 1-bit version of Moniqua [Lu and 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 and Sa, 2020]. We use the same step size schedule  $\{\alpha_k\}$

as for the optimizers we evaluated, and tune the a priori bound  $\theta$  as a global 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  $B_\theta$ ’ in the algorithm yields values between  $-\frac{1}{2}B_\theta$  and  $\frac{1}{2}B_\theta$ .

## B.9 Parameters in architectures

See Table B.4 and Table B.5 for an overview of parameters in the models used.

Table B.4: 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	$64 \times 64 \times 3 \times 3$	$64 \times 576$	144 KB	$115/r \times$
layer3.2.conv1	$64 \times 64 \times 3 \times 3$	$64 \times 576$	144 KB	$115/r \times$
layer3.0.conv2	$64 \times 64 \times 3 \times 3$	$64 \times 576$	144 KB	$115/r \times$
layer3.1.conv2	$64 \times 64 \times 3 \times 3$	$64 \times 576$	144 KB	$115/r \times$
layer3.2.conv2	$64 \times 64 \times 3 \times 3$	$64 \times 576$	144 KB	$115/r \times$
layer3.0.conv1	$64 \times 32 \times 3 \times 3$	$64 \times 288$	72 KB	$105/r \times$
layer2.2.conv2	$32 \times 32 \times 3 \times 3$	$32 \times 288$	36 KB	$58/r \times$
layer2.1.conv1	$32 \times 32 \times 3 \times 3$	$32 \times 288$	36 KB	$58/r \times$
layer2.0.conv2	$32 \times 32 \times 3 \times 3$	$32 \times 288$	36 KB	$58/r \times$
layer2.1.conv2	$32 \times 32 \times 3 \times 3$	$32 \times 288$	36 KB	$58/r \times$
layer2.2.conv1	$32 \times 32 \times 3 \times 3$	$32 \times 288$	36 KB	$58/r \times$
layer2.0.conv1	$32 \times 16 \times 3 \times 3$	$32 \times 144$	18 KB	$52/r \times$
layer1.1.conv1	$16 \times 16 \times 3 \times 3$	$16 \times 144$	9 KB	$29/r \times$
layer1.1.conv2	$16 \times 16 \times 3 \times 3$	$16 \times 144$	9 KB	$29/r \times$
layer1.0.conv2	$16 \times 16 \times 3 \times 3$	$16 \times 144$	9 KB	$29/r \times$
layer1.2.conv1	$16 \times 16 \times 3 \times 3$	$16 \times 144$	9 KB	$29/r \times$
layer1.0.conv1	$16 \times 16 \times 3 \times 3$	$16 \times 144$	9 KB	$29/r \times$
layer1.2.conv2	$16 \times 16 \times 3 \times 3$	$16 \times 144$	9 KB	$29/r \times$
layer3.0.downsample.0	$64 \times 32 \times 1 \times 1$	$64 \times 32$	8 KB	$43/r \times$
fc	$10 \times 64$	$10 \times 64$	2 KB	$17/r \times$
layer2.0.downsample.0	$32 \times 16 \times 1 \times 1$	$32 \times 16$	2 KB	$21/r \times$
conv1	$16 \times 3 \times 3 \times 3$	$16 \times 27$	2 KB	$20/r \times$
Bias vectors (total)			6 KB	None

Table B.5: 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	$28869 \times 650$	$28869 \times 650$	73300 KB	$1271/r \times$
rnn-ih-l0	$2600 \times 650$	$2600 \times 650$	6602 KB	$1040/r \times$
rnn-hh-l0	$2600 \times 650$	$2600 \times 650$	6602 KB	$1040/r \times$
rnn-ih-l1	$2600 \times 650$	$2600 \times 650$	6602 KB	$1040/r \times$
rnn-hh-l1	$2600 \times 650$	$2600 \times 650$	6602 KB	$1040/r \times$
rnn-ih-l2	$2600 \times 650$	$2600 \times 650$	6602 KB	$1040/r \times$
rnn-hh-l2	$2600 \times 650$	$2600 \times 650$	6602 KB	$1040/r \times$
Bias vectors (total)			174 KB	None

## **B.10 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.

## Appendix C

# Appendix for Beyond Spectral gap

### C.1 Notation

Table C.1 defines some notation and conventions used throughout this paper and in the appendix.

Table C.1: Notation

Bold symbol $\mathbf{v}$	Vector
Bold uppercase $\mathbf{M}$	Matrix
$\mathcal{N}^d(0, 1)$	Standard normal distribution with $d$ independent dimensions
$\langle \mathbf{x}, \mathbf{y} \rangle$	Inner product $\mathbf{x}^\top \mathbf{y}$
$\ \mathbf{T}\ _2$	Spectral norm
$\ \mathbf{T}\ _F$	Frobenius norm
$\mathbf{P} \otimes \mathbf{Q}$	Kronecker product
$\mathbf{1}$	Vector of all ones

## C.2 Topologies

The static topologies that we consider in this work are drawn in fig. C.1. Figures C.2 and C.3 show the gossip matrices we use in detail.

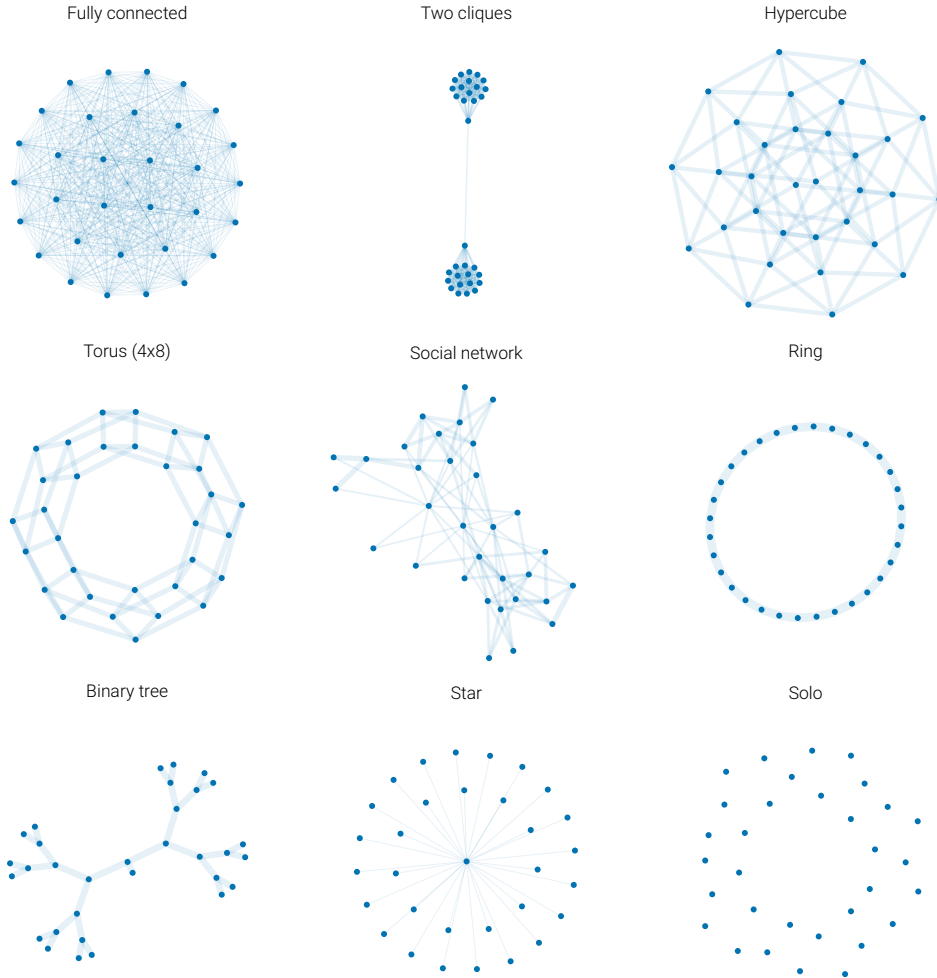


Figure C.1: Spring-layout drawings of the static graph topologies considered used this paper. The nodes represent workers, and an edge between two workers indicates that they are connected. The thickness of an edge is proportional to its averaging weight.



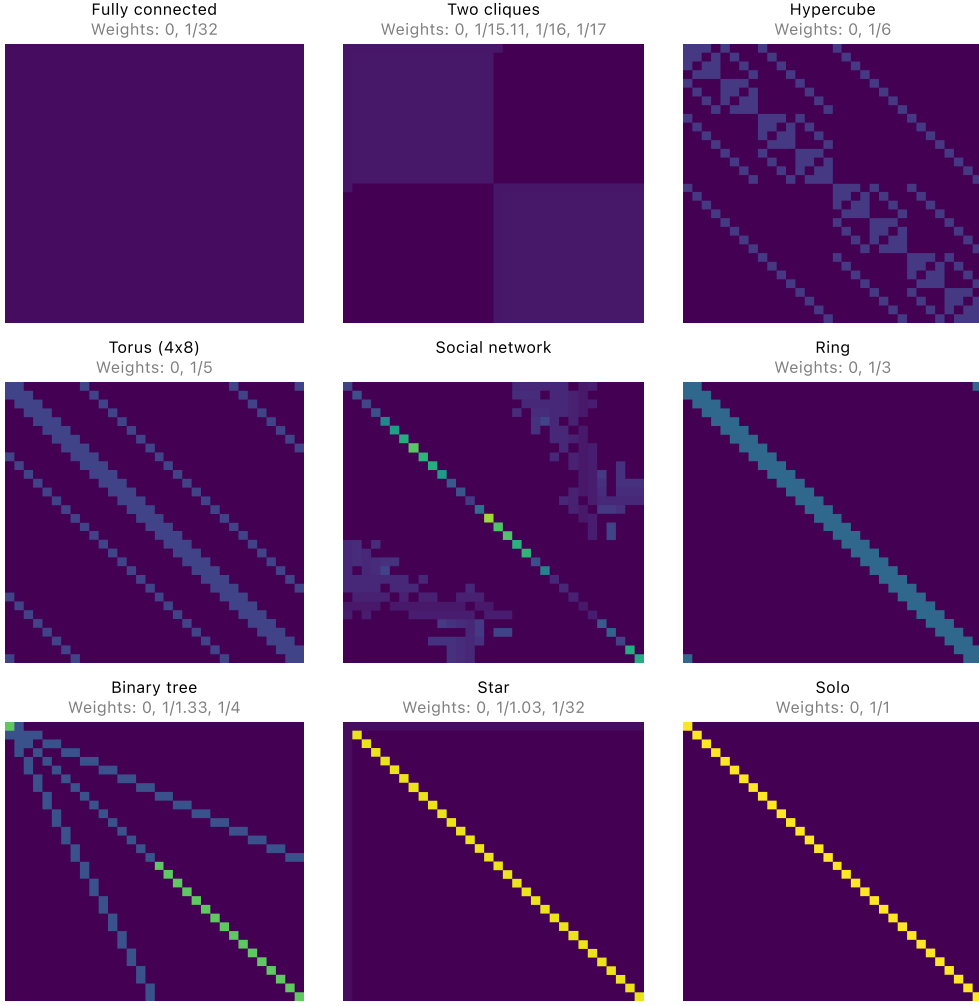


Figure C.2: Gossip matrices corresponding to the graph topologies drawn in fig. C.1.  $x$  and  $y$  axes represent workers, and the color of each coordinate in the plots indicates the gossip weight between each pair of workers. The brighter, the higher the weight.

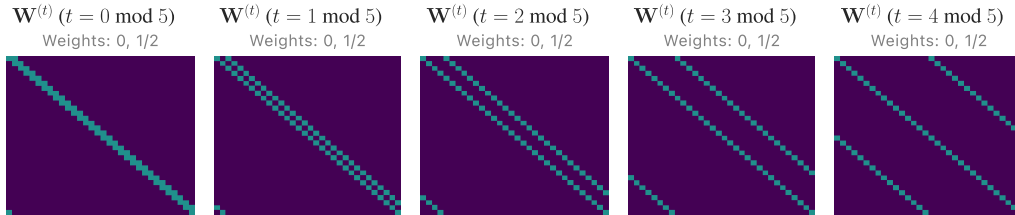


Figure C.3: Gossip matrices for the time-varying exponential graph [Assran et al., 2019, Ying et al., 2021]. The product of  $\log n$  consecutive gossip matrices equals to the fully-connected averaging matrix with  $w_{ij} = 1/n \forall i, j$ .

### C.3 Random quadratics

#### C.3.1 Objective

We study the simple problem of minimizing an isotropic  $d$ -dimensional quadratic,

$$\mathbf{x}^* = \arg \min_{\mathbf{x} \in \mathbb{R}^d} f(\mathbf{x})$$

where the objective function  $f(x) = \frac{1}{2} \|\mathbf{x}\|^2$  is considered to be the expectation over an infinite dataset with random normal features and labels 0:

$$f(\mathbf{x}) = \mathbb{E}_{\mathbf{d} \sim \mathcal{N}^d(0,1)} \frac{1}{2} \langle \mathbf{d}, \mathbf{x} \rangle^2. \quad (\text{C.1})$$

The optimum of this objective is at  $\mathbf{x}^* = \mathbf{0}$  without loss of generality, because any shifted quadratic would behave the same in the algorithm studied. We will access this objective function through stochastic gradients of the form  $\mathbf{g}(\mathbf{x}) = \mathbf{d}\mathbf{d}^\top \mathbf{x}$ . The stochasticity of these gradients disappears at the optimum, like in an over-parameterized model.

The difficulty of this problem depends on the dimensionality  $d$ . For a lower-dimensional problem, the ‘stochastic Hessian’  $\mathbf{d}\mathbf{d}^\top$  is closer to the true hessian  $\mathbf{I}$  than for a high dimensional one. This level of stochasticity is captured by the following quantity:

**Definition H (noise level)**  $\zeta = \sup_{\mathbf{x}} \frac{\mathbb{E}_{\mathbf{d}} \|\mathbf{d}\mathbf{d}^\top \mathbf{x}\|^2}{\|\mathbf{x}\|^2}$ .

For our random normal data with batch size 1, this notion of noise level corresponds directly to the dimensionality of the data as  $\zeta = d + 2$ .

#### C.3.2 Algorithm

The objective (C.1) is collaboratively optimized by  $n$  workers. At every time step  $t$ , each worker  $i$  has its own copy of the ‘model’  $\mathbf{x}_i^{(t)} \in \mathbb{R}^d$ . In the D-SGD algorithm, workers iteratively compute stochastic gradient estimates  $\mathbf{g}_i^{(t)} = \mathbf{d}_i^{(t)} \mathbf{d}_i^{(t)\top} \mathbf{x}_i^{(t)}$ , where  $\mathbf{d}_i^{(t)}$  are i.i.d. from  $\mathcal{N}^d(0, 1)$ . The stochastic gradients are unbiased:  $\mathbb{E} \mathbf{g}_i^{(t)} = \nabla f(\mathbf{x}_i^{(t)}) = \mathbf{x}_i^{(t)}$ .

Workers interleave stochastic gradient updates with gossip averaging:

$$\begin{aligned} \mathbf{x}_i^{(0)} &= \mathbf{x}^{(0)} \quad \forall i \\ \mathbf{x}_i^{(t+1)} &= \mathbf{W}(\mathbf{x}_i^{(t)} - \eta \mathbf{g}_i^{(t)}), \end{aligned}$$

where  $\eta$  is the learning rate and

$$\mathbf{W}(\mathbf{x}_i) = \sum_{j=1}^n w_{ij} \mathbf{x}_j.$$

This linear operation can be interpreted as matrix multiplication, but one operating on each coordinate of the model independently.  $\mathbf{W}$  is an  $n \times n$  matrix, and *not* a  $d \times d$  matrix as the notation may suggest. The averaging weights  $w_{ij}$  encode the connectivity of the communication topology: non-zero  $w_{ij}$  implies that workers  $i$  and  $j$  are directly connected. We make several assumptions about the gossip weights in this analysis:

**Assumption I** Constant gossip weights: The weights  $w_{ij}$  are constant between D-SGD steps.

**Assumption J** Symmetric gossip weights:  $w_{ij} = w_{ji}$ .

**Assumption K** Doubly stochastic gossip weights:  $w_{ij} \geq 0 \forall i, j$ ,  $\sum_j w_{ij} = 1 \forall i$ ,  $\sum_i w_{ij} = 1 \forall j$ .

**Assumption L** Regular topology: all workers have  $k$  directly-connected neighbors, and  $w_{ij} = c$  for some constant  $c$ , and for each edge where  $i \neq j$ .

**Definition M** Spectrum of  $\mathbf{W}$ . Let the eigenvalues of  $\mathbf{W}$  be  $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$ . We call the corresponding eigenvectors  $\mathbf{v}_1, \dots, \mathbf{v}_n$ . Under assumption K,  $\lambda_1 = 1$ , and we call  $1 - \lambda_2$  the *spectral gap* of  $\mathbf{W}$ .

The assumptions on constant gossip weights and regular topologies are mainly here to ease the analysis. We experimentally observe that our findings hold for time-varying topologies and infinite graphs, too, and that they approximately hold for irregular graphs.

### C.3.3 Linear convergence of an unrolled error vector

We will study the convergence of the algorithm by tracking the *error matrix*  $\mathbf{E} \in \mathbb{R}^{n \times n}$ . The coordinates of this matrix are the expected covariance between each pair of workers.

$$\mathbf{E}_{ij}^{(t)} = \mathbb{E} \langle \mathbf{x}_i^{(t)}, \mathbf{x}_j^{(t)} \rangle.$$

We sometimes flatten the error matrix into a vector  $\mathbf{e} \in \mathbb{R}^{n^2}$ , such that  $\mathbf{e}_{ni+j} = \mathbf{E}_{ij}$ . The diagonal entries of this matrix describe the worker's error on the objective, and as all workers converge to the optimum at zero, each entry of the matrix will converge to zero. Our analysis of  $\mathbf{E}$  quantity starts with a key observation:

**Lemma 10** There exists an  $n^2 \times n^2$  ‘transition’ matrix  $\mathbf{T}$  such that  $\mathbf{e}^{(t+1)} = \mathbf{T}\mathbf{e}^{(t)} \forall t$ .

*Proof.* Because both gossip averaging and the gradient updates are linear, this follows from expanding the inner product.  $\square$

The transition matrix  $\mathbf{T}$  depends on the gossip matrix  $\mathbf{W}$  and on the learning rate  $\eta$ . Its spectral gap describes the convergence of the algorithm. D-SGD converges linearly if the norm  $\|\mathbf{T}\|_2 < 1$ .

We separate  $\mathbf{T}$  into a product  $\mathbf{T} = \mathbf{T}^{\text{gossip}} \mathbf{T}^{\text{grad}}$ , where  $\mathbf{T}^{\text{grad}}$  and  $\mathbf{T}^{\text{gossip}}$  respectively capture the gradient update and gossip steps of the algorithm. We find that

$$\mathbf{T}^{\text{gossip}} = \mathbf{W} \otimes \mathbf{W}$$

and that  $\mathbf{T}^{\text{grad}}$  is diagonal. It only operates element-wise, such that

$$\left[ \mathbf{T}^{\text{grad}} \mathbf{e} \right]_{ni+j} = \begin{cases} (1 - \eta)^2 \mathbf{e}_{ni+j} + (\zeta - 1) \eta^2 \mathbf{e}_{ni+j} & i = j \text{ (same worker)}, \\ (1 - \eta)^2 \mathbf{e}_{ni+j} & i \neq j \text{ (different workers)}. \end{cases} \quad (\text{C.2})$$

This follows directly from expanding the inner product  $\langle \mathbf{x}_i - \eta \mathbf{g}_i, \mathbf{x}_j - \eta \mathbf{g}_j \rangle$ . The terms with  $i = j$  behave differently than the ones where  $i \neq j$ , because the noise cancels if  $i \neq j$ .

### C.3.4 Random walks with gossip averaging

Before we study the convergence of D-SGD on the random quadratic objective, we first take a step back and inspect a particular random walk process, where workers average their random walk iterates through gossip averaging.

Let  $\mathbf{z}^{(t)} \in \mathbb{R}^n$  be a vector containing (scalar) iterates of  $n$  workers in the following process:

$$\mathbf{z}^{(0)} = \mathbf{0}, \quad (\text{C.3})$$

$$\mathbf{z}^{(t+1)} = \mathbf{W} \left( \sqrt{\gamma} \mathbf{z}^{(t)} + \boldsymbol{\xi}^{(t)} \right) \quad \text{where } \boldsymbol{\xi}^{(t)} \sim \mathcal{N}^n(0, 1). \quad (\text{C.4})$$

We call the parameter  $0 < \gamma \leq 1$  the ‘decay rate’. Note that the name *random walk* refers to iterative addition of random noise to the workers iterates, and not to a ‘random walk’ between nodes of the graph.

For this random walk, we will track the covariance matrix  $\mathbf{C} \in \mathbb{R}^{n \times n}$  across workers (and its flattened version  $\mathbf{c} \in \mathbb{R}^{n^2}$ ). Its coordinates are

$$\mathbf{C}_{ij}^{(t)} = \mathbb{E}[\mathbf{z}_i^{(t)} \mathbf{z}_j^{(t)}].$$

**Lemma 11** For static, symmetric and doubly-stochastic topologies (Assumptions I, J and K), the Eigen decomposition of the covariance is

$$\mathbf{C}^{(t)} = \sum_{i=1}^n c_i^{(t)} \mathbf{v}_i \mathbf{v}_i^\top,$$

with  $0 \leq c_i^{(t)} \leq \frac{\lambda_i^2}{1-\gamma\lambda_i^2}$ . Here  $(\lambda_i, \mathbf{v}_i)$  are the eigenvalue/eigenvector pairs of  $\mathbf{W}$ . As  $t \rightarrow \infty$ ,  $c_i^{(t)} = \frac{\lambda_i^2}{1-\gamma\lambda_i^2}$  with equality.

*Proof.* We can unroll the iterations:

$$\mathbf{z}^{(t)} = \sum_{k=1}^t \mathbf{W}^k \gamma^{(k-1)/2} \boldsymbol{\xi}^{(t-k)}$$

and use the temporal independence of  $\boldsymbol{\xi}^{(t)}$  to write

$$\mathbf{C}^{(t)} = \mathbb{E}[\mathbf{z}^{(t)} \mathbf{z}^{(t)\top}] = \sum_{k=1}^t \gamma^k \mathbf{W}^{2k} \mathbb{E}[\boldsymbol{\xi}^{(t-k)} \boldsymbol{\xi}^{(t-k)\top}] = \sum_{k=1}^t \gamma^k \mathbf{W}^{2k}.$$

Using commutativity of  $\mathbf{W}$  and its Eigen decomposition (Assumptions J, K), we can decompose it as

$$\mathbf{C}^{(t)} = \sum_{i=1}^n \mathbf{v}_i \mathbf{v}_i^\top \underbrace{\left( \sum_{k=1}^t \gamma^{k-1} \lambda_i^{2k} \right)}_{c_i^{(t)}}$$

Because all terms of parenthesized expression are non-negative, and its limit equals  $\frac{\lambda_i^2}{1-\gamma\lambda_i^2}$ , this proves the Lemma.  $\square$

**Lemma 12** When the topology is regular (Assumption L) in addition to the assumptions of Lemma 11, workers in the random walk process have equal variance:

$$\text{Var}[\mathbf{z}_i^{(t)}] = \frac{1}{n} \text{Tr}[\mathbf{C}^{(t)}] = \frac{1}{n} \sum_{i=1}^n c_i^{(t)}.$$

*Proof.* The variances of  $\mathbf{z}_i$  are the diagonal entries of the covariance matrix. By regularity, and because workers are initialized equally, all workers should have the same variance.  $\text{Var}[\mathbf{z}_i^{(t)}]$  is therefore equal to the average diagonal entry of  $\mathbf{C}_i^{(t)}$ . The second equality is a standard property of the trace.  $\square$

**Lemma 13** Under the assumptions of Lemma 12, the variance  $\text{Var}[\mathbf{z}_i^{(t)}]$  increases over time:

$$\text{Var}[\mathbf{z}_i^{(t)}] \leq \text{Var}[\mathbf{z}_i^{(t+1)}] \leq \lim_{t' \rightarrow \infty} \text{Var}[\mathbf{z}_i^{(t')}] \quad \forall t.$$

*Proof.* If we write  $\text{Var}[\mathbf{z}_i^{(t)}]$  as  $\frac{1}{n} \sum_{i=1}^n c_i^{(t)}$  using Lemma 12, the statement of this Lemma follows from the realization in Lemma 11 that  $c_i^{(t)}$  increases over time to the limit  $\frac{\lambda_i^2}{1-\gamma\lambda_i^2}$  for all  $i$ .  $\square$

Note that while the results above are for static gossip matrices, random walks and these variance quantities can be analogously defined time-varying topologies. Those just lack a simple exact form. The stronger the averaging of the gossip process, the lower the variance. We capture this in the following quantity:

**Definition N (effective number of neighbors)**

$$n_{\mathbf{W}}(\gamma) = \frac{\frac{1}{1-\gamma}}{\lim_{t \rightarrow \infty} \frac{1}{n} \sum_{i=1}^n \text{Var}[\mathbf{z}_i^{(t)}]},$$

where  $\mathbf{z}$  are the iterates from a random walk with gossip averaging, with decay parameter  $\gamma$ . The numerator is the variance of the random walk process without any gossip averaging ( $\mathbf{W} = \mathbf{I}$ ).

### C.3.5 Converging random walk

The covariance of the random walk process  $\mathbf{C}$  and the error matrix of D-SGD iterates  $\mathbf{E}$  share clear similarities. The quantities are both iteratively updated by an affine transformation. The main difference between them, however, is that  $\mathbf{C}$  converges to a non-zero constant while  $\mathbf{E}$  converges linearly to zero (or it diverges.)

In the next section, we draw a clear connection between the two processes, but first, we define a modified version of the random walk process that further highlights their similarity.

**Definition O (scaled random walk)** Let  $0 < r < 1$  be a scalar. We define a scaled version of the random walk iterates, such that

$$\begin{aligned} \mathbf{y}^{(t)} &= (1-r)^{t/2} \mathbf{z}^{(t)}, \\ \mathbf{B}^{(t)} &= (1-r)^t \mathbf{C}^{(t)}, \text{ and} \\ \text{Var}[\mathbf{y}_i^{(t)}] &= (1-r)^t \text{Var}[\mathbf{z}_i^{(t)}] \end{aligned}$$

Because the sequence  $\mathbf{z}^{(t)}$  converges to a non-zero stationary point, the scaled sequence  $\mathbf{y}^{(t)}$  converges to zero with a linear rate  $r$ .

**Lemma 14** Under the assumptions of Lemma 12, the variance  $\text{Var}[\mathbf{y}_i^{(t)}]$  is bounded as

$$\text{Var}[\mathbf{y}_i^{(t)}] \leq \frac{(1-r)^t}{(1-\gamma)n_{\mathbf{W}}(\gamma)},$$

with equality as  $t \rightarrow \infty$ .

*Proof.* From Lemma 13, we know that  $(1-r)^t \text{Var}[\mathbf{z}_i^{(t)}] \leq (1-r)^t \lim_{t' \rightarrow \infty} \text{Var}[\mathbf{z}_i^{(t')}]$ , with equality as  $t \rightarrow \infty$ . Because the variance  $\text{Var}[\mathbf{z}_i^{(t)}]$  is equal across workers  $i$  (Lemma 12), the Lemma follows from rearranging Definition N.  $\square$

**Lemma 15** The covariance vector  $\mathbf{b}$  (the flattened version of  $\mathbf{B}$ ) of this scaled random walk process follows the recursion  $\mathbf{b}^{(t+1)} = \mathbf{T}^{\text{gossip}} u_t(\mathbf{b}^{(t)})$ , where

$$u_t(\mathbf{b}^{(t)})_{ni+j} = \begin{cases} \gamma(1-r) \mathbf{b}_{ni+j}^{(t)} + (1-r)^{t+1} & i = j \text{ (same worker)}, \\ \gamma(1-r) \mathbf{b}_{ni+j}^{(t)} & i \neq j \text{ (different workers)}. \end{cases}$$

*Proof.* The entries  $u_t(\mathbf{b}^{(t)})_{ni+j}$  are inner products:

$$\begin{aligned} u_t(\mathbf{b}^{(t)})_{ni+j} &= (1-r)^{t+1} \left\langle \sqrt{\gamma} \mathbf{y}_i^{(t)} + \boldsymbol{\xi}_i^{(t)}, \sqrt{\gamma} \mathbf{y}_j^{(t)} + \boldsymbol{\xi}_j^{(t)} \right\rangle \\ &= \gamma(1-r) \mathbf{b}_{ni+j}^{(t)} + (1-r)^{t+1} \mathbb{E} \left\langle \boldsymbol{\xi}_i^{(t)}, \boldsymbol{\xi}_j^{(t)} \right\rangle. \end{aligned}$$

The inner product between noise contributions  $\boldsymbol{\xi}_i^{(t)}$  and  $\boldsymbol{\xi}_j^{(t)}$  are 1 if  $i = j$  and 0 otherwise.  $\square$

**Lemma 16** The covariance  $\mathbf{b}$  follows the recursion  $\mathbf{b}^{(t+1)} \geq \mathbf{T}^{\text{gossip}} \mathbf{T}^{\text{r.w.}} \mathbf{b}^{(t)}$  (element-wise), where

$$[\mathbf{T}^{\text{r.w.}} \mathbf{b}^{(t)}]_{ni+j} = \begin{cases} \gamma(1-r) \mathbf{b}_{ni+j}^{(t)} + (1-r)(1-\gamma)n_{\mathbf{W}}(\gamma) \mathbf{b}_{ni+j}^{(t)} & i = j, \\ \gamma(1-r) \mathbf{b}_{ni+j}^{(t)} & i \neq j. \end{cases} \quad (\text{C.5})$$

In the limit of  $t \rightarrow \infty$ , this is true with equality.

*Proof.* From Lemma 14, we have that  $\mathbf{b}_{ni+i}^{(t)} = \text{Var}[\mathbf{y}_i^{(t)}] \leq \frac{(1-r)^t}{(1-\gamma)n_{\mathbf{W}}(\gamma)}$ , with equality as  $t \rightarrow \infty$ . The entries of  $\mathbf{T}^{\text{r.w.}} \mathbf{b}^{(t)}$  are therefore all smaller than or equal to the entries of  $u_t(\mathbf{b}^{(t)})$  from Lemma 15, which proves the Lemma.  $\square$

### C.3.6 The rate for D-SGD

**Theorem VII (D-SGD on random quadratics)** Under assumptions I, J, K, and L, if the pair of the learning rate  $\eta$  and  $r$  satisfy

$$r = 1 - (1 - \eta)^2 - \frac{(\zeta - 1)\eta^2}{n_{\mathbf{W}} \left( \frac{(1-\eta)^2}{1-r} \right)}, \quad (\text{C.6})$$

the error of D-SGD with learning rate  $\eta$  on the random quadratic objective with noise parameter  $\zeta$  converges with rate  $r$ :

$$\sum_{i=1}^n \mathbb{E} \|\mathbf{x}_i^{(t)}\|^2 \leq (1 - r)^t \sum_{i=1}^n \mathbb{E} \|\mathbf{x}_i^{(0)}\|^2.$$

This rate becomes exact as  $t \rightarrow \infty$ .

*Proof.* If the condition (C.6) is satisfied, the expected error iterates  $\mathbf{E}$  (Equation C.2) of the D-SGD algorithm follow the transition matrix (C.5) of Lemma 16 with  $\gamma = \frac{(1-\eta)^2}{1-r}$ . The choice of  $\gamma$  ensures that  $\gamma(1-r) = (1-\eta)^2$ , and the condition (C.6) that  $(\zeta - 1)\eta^2 = (1-r)(1-\gamma)n_{\mathbf{W}}(\gamma)$ .

From Lemma 16, we know that a sequence that has this transition matrix  $\mathbf{T}^{\text{gossip}} \mathbf{T}^{\text{r.w.}} \mathbf{b}^{(t)}$  (C.5) converges at least as fast as the iterates of the corresponding scaled random walk process  $\mathbf{B}$ , with equality in the limit as  $t \rightarrow \infty$ .

Since  $\mathbf{B}$  converges to zero with a rate  $r$ , this now implies the same rate for the error matrix  $\mathbf{E}$ . The sum  $\sum_{i=1}^n \mathbb{E} \|\mathbf{x}_i^{(t)}\|^2$  in the statement of this theorem is the trace of the matrix  $\mathbf{E}^{(t)}$ , and therefore it converges to zero with the same rate. This completes the proof.  $\square$

## C.4 (Strongly)-Convex case, missing proofs and additional results

### C.4.1 Preliminaries on Bregman divergences

Throughout this section, we will use Bregman divergences, which are defined for a differentiable function  $h$  and two points  $x, y \in \mathbb{R}^d$  as:

$$D_h(x, y) = h(x) - h(y) - \nabla h(y)^\top (x - y). \quad (\text{C.7})$$

We assume throughout this paper that the functions we consider are twice continuously differentiable and strictly convex on  $\text{dom } h$ , and that  $\nabla h(x) = \min_y h(y) - x^\top y$  is uniquely defined (although milder assumptions could be used). Among the many properties of these divergences, an important one is that if  $h$  is  $L$  smooth and  $\mu$  strongly-convex, then

$$\frac{\mu}{2} \|x - y\|^2 \leq D_h(x, y) \leq \frac{L}{2} \|x - y\|^2 \quad (\text{C.8})$$

Another important property is called duality, which states that:

$$D_h(x, y) = D_{h^*}(\nabla h(y), \nabla h(x)), \quad (\text{C.9})$$

where  $h^*$  is the convex conjugate of  $h$ .

### C.4.2 Main result

This section is devoted to proving Theorem VIII, from which Theorem III can be deduced directly by taking  $\omega = M_0$  and  $p = 1/2$ . We recall Assumption B, which is at the heart of Theorem VIII.

**Assumption B** The stochastic gradients are such that: (I)  $\xi_i^{(t)}$  and  $\xi_j^{(\ell)}$  are independent for all  $t, \ell$  and  $i \neq j$ . (II)  $\mathbb{E}[f_{\xi_i^{(t)}}] = f$  for all  $t, i$ . (III)  $\mathbb{E}\|\nabla f_{\xi_i^{(t)}}(\mathbf{x}^*)\|^2 \leq \sigma^2$  for all  $t, i$ , where  $\mathbf{x}^*$  is a minimizer of  $f$ . (IV)  $f_{\xi_i^{(t)}}$  is convex and  $\zeta$ -smooth for all  $t, i$ . (V)  $f$  is  $\mu$ -strongly-convex for  $\mu \geq 0$  and  $L$ -smooth.

Note that Assumption B (IV) is stated in this form for simplicity, but it can be relaxed by asking directly that  $\mathbb{E}[\|\nabla f_{\xi_i^{(t)}}(x^{(t)}) - \nabla f_{\xi_i^{(t)}}(x^*)\|^2] \leq 2\zeta D_f(x^*, x^{(t)})$ , which can also be implied by assuming that each  $f_{\xi}$  is  $\zeta_{\xi}$ -smooth, with  $\mathbb{E}[\zeta_{\xi} D_{f_{\xi}}(x^*, x^{(t)})] \leq \zeta D_f(x^*, x^{(t)})$  (see Equation (C.17)). These weaker forms would be satisfied by the toy problem of Section 4.4.

**Theorem VIII** Denote  $\mathbf{x}^{(t)}$  the iterates obtained by D-SGD,  $\mathbf{L}_{\mathbf{M}} = \mathbf{I} - \mathbf{M}$ , and  $p$  the probability to perform a communication step ( $\mathbf{x}_{t+1} = \mathbf{W}\mathbf{x}_t$ ). Parameter  $\beta$  is such that  $\mathbf{I} - \mathbf{W} \succcurlyeq \beta \mathbf{L}_{\mathbf{M}}$ . For some  $\omega > 0$ , denote:

$$\mathcal{L}_t = \|\mathbf{x}^{(t)} - \mathbf{x}^*\|_{\mathbf{M}}^2 + \omega \|\mathbf{x}^{(t)}\|_{\mathbf{L}_{\mathbf{M}}}^2. \quad (\text{C.10})$$

Then, if  $\eta$  is such that:

$$\eta \leq \frac{M_0 \beta}{L} \frac{p}{1-p}, \quad (\text{C.11})$$

$$\eta \leq \frac{1}{4(M_0 \zeta + L)} \quad (\text{C.12})$$

we have that:

$$\mathcal{L}_t \leq [1 - (1-p)\eta\mu]^t \mathcal{L}_0 + \frac{\eta \tilde{\sigma}^2}{\mu}, \quad (\text{C.13})$$

with  $\tilde{\sigma}^2 = \sigma_{\mathbf{M}}^2 + \omega \sigma_{\mathbf{L}_{\mathbf{M}}}^2$ , where  $\mathbb{E}\|\nabla f_{\xi_i^{(t)}}(\mathbf{x}^*)\|_{\mathbf{M}}^2 \leq \sigma_{\mathbf{M}}^2$  (and similarly for  $\mathbf{L}_{\mathbf{M}}$ ).

In the convex case ( $\mu = 0$ ), we have:

$$\mathbb{E} \left[ \frac{1}{T} \sum_{t=0}^{T-1} D_f(\mathbf{M}\mathbf{x}^{(t)}, \mathbf{x}^*) \right] \leq \frac{1}{1-p} \frac{\mathcal{L}_0}{\eta T} + \eta \tilde{\sigma}^2 \quad (\text{C.14})$$

Note that the factors 2 in Equation (C.12) are simplifications to make the result more readable but could be improved.

*Proof.* We now proceed to the proof of the theorem. To show that the Lyapunov  $\mathcal{L}_t$  decreases over iterations, we will study how each quantity  $\|\mathbf{x}^{(t)} - \mathbf{x}^*\|_{\mathbf{M}}^2$  and  $\|\mathbf{x}^{(t)}\|_{\mathbf{L}_{\mathbf{M}}}^2$  evolves through time. In particular, we will first consider the case of computation updates (so, local gradient updates), and then the case of gossip updates.

**1 - Computation updates** In this case, we assume that the update is of the form

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \eta \nabla f_{\xi_t}(\mathbf{x}^{(t)}). \quad (\text{C.15})$$

This happens with probability  $1-p$ , and expectations are taken with respect to  $\xi_t$ . To avoid notation clutter, we use notations  $\nabla f_{\xi}$  and  $\nabla f_{\xi,i}$ , which are such that  $\nabla f_{\xi,i}(\mathbf{x}^{(t)}) = (\nabla f_{\xi}(\mathbf{x}^{(t)}))_i = \nabla f_{\xi_i^{(t)}}(\mathbf{x}_i^{(t)})$ .



**Distance to optimum** We bound the distance to optimum as follows, using that  $\mathbf{M}\mathbf{x}^\star = \mathbf{x}^\star$ , and  $\mathbb{E} [\nabla f_\xi(\mathbf{x}^{(t)})] = \nabla f(\mathbf{x}^{(t)})$ :

$$\begin{aligned}\mathbb{E} [\|\mathbf{x}^{(t+1)} - \mathbf{x}^\star\|_{\mathbf{M}}^2] &= \|\mathbf{x}^{(t)} - \mathbf{x}^\star\|_{\mathbf{M}}^2 - 2\eta \mathbb{E} [(\mathbf{x}^{(t)} - \mathbf{x}^\star)^\top \mathbf{M} \nabla f_\xi(\mathbf{x}^{(t)})] + \eta^2 \|\nabla f_\xi(\mathbf{x}^{(t)})\|_{\mathbf{M}}^2 \\ &= \|\mathbf{x}^{(t)} - \mathbf{x}^\star\|_{\mathbf{M}}^2 - 2\eta (\mathbf{M}\mathbf{x}^{(t)} - \mathbf{x}^\star)^\top \nabla f(\mathbf{x}^{(t)}) + \eta^2 \mathbb{E} [\|\nabla f_\xi(\mathbf{x}^{(t)})\|_{\mathbf{M}}^2].\end{aligned}$$

Then, we expand the middle term in the following way:

$$\begin{aligned}-\nabla f(\mathbf{x}^{(t)})^\top (\mathbf{M}\mathbf{x}^{(t)} - \mathbf{x}^\star) &= -\nabla f(\mathbf{x}^{(t)})^\top (\mathbf{x}^{(t)} - \mathbf{x}^\star) - \nabla f(\mathbf{x}^{(t)})^\top (\mathbf{M}\mathbf{x}^{(t)} - \mathbf{x}^{(t)}) \\ &= -D_f(\mathbf{x}^{(t)}, \mathbf{x}^\star) - D_f(\mathbf{x}^\star, \mathbf{x}^{(t)}) + D_f(\mathbf{M}\mathbf{x}^{(t)}, \mathbf{x}^{(t)}) - f(\mathbf{M}\mathbf{x}^{(t)}) + f(\mathbf{x}^{(t)}) \\ &= -D_f(\mathbf{M}\mathbf{x}^{(t)}, \mathbf{x}^\star) - D_f(\mathbf{x}^\star, \mathbf{x}^{(t)}) + D_f(\mathbf{M}\mathbf{x}^{(t)}, \mathbf{x}^{(t)}) \\ &\leq -\frac{\mu}{2} \|\mathbf{x}^{(t)} - \mathbf{x}^\star\|_{\mathbf{M}^2}^2 - D_f(\mathbf{x}^\star, \mathbf{x}^{(t)}) + \frac{L}{2} \|\mathbf{M}\mathbf{x}^{(t)} - \mathbf{x}^{(t)}\|^2,\end{aligned}\tag{C.16}$$

where in the last time we used the  $\mu$ -strong convexity and  $L$ -smoothness of  $f$ . For the noise term, we use that fact that  $(\nabla f_\xi(\mathbf{x}^{(t)}))_i$  and  $(\nabla f_\xi(\mathbf{x}^{(t)}))_j$  are independent for  $i \neq j$ , so that

$$\begin{aligned}\frac{1}{2} \mathbb{E} [\|\nabla f_\xi(\mathbf{x}^{(t)})\|_{\mathbf{M}}^2] &= \mathbb{E} [\|\nabla f_\xi(\mathbf{x}^{(t)}) - \nabla f_\xi(\mathbf{x}^\star)\|_{\mathbf{M}}^2] + \mathbb{E} [\|\nabla f_\xi(\mathbf{x}^\star)\|_{\mathbf{M}}^2] \\ &= \sum_{i=1}^n \mathbf{M}_{ii} \mathbb{E} [\|\nabla f_{\xi,i}(\mathbf{x}^{(t)}) - \nabla f_{\xi,i}(\mathbf{x}^\star)\|^2] + \mathbb{E} [\|\nabla f_\xi(\mathbf{x}^\star)\|_{\mathbf{M}}^2] \\ &\quad + \sum_{i=1}^n \sum_{j \neq i} \mathbf{M}_{ij} \mathbb{E} [(\nabla f_{\xi,i}(\mathbf{x}^{(t)}) - \nabla f_{\xi,i}(\mathbf{x}^\star))^\top (\nabla f_{\xi,j}(\mathbf{x}^{(t)}) - \nabla f_{\xi,j}(\mathbf{x}^\star))] \\ &= \sum_{i=1}^n \mathbf{M}_{ii} \mathbb{E} [\|\nabla f_{\xi,i}(\mathbf{x}^{(t)}) - \nabla f_{\xi,i}(\mathbf{x}^\star)\|^2] + \|\nabla f(\mathbf{x}^{(t)})\|_{\mathbf{M}}^2 + \mathbb{E} [\|\nabla f_\xi(\mathbf{x}^\star)\|_{\mathbf{M}}^2].\end{aligned}$$

We now use for all  $i \in \{1, \dots, n\}$  the  $\zeta$ -smoothness of  $f_{\xi,i}$ , which implies the  $\zeta^{-1}$ -strong convexity of  $f_{\xi,i}^*$  [Kakade et al., 2009], so that:

$$\begin{aligned}\mathbb{E} [\|\nabla f_{\xi,i}(\mathbf{x}^{(t)}) - \nabla f_{\xi,i}(\mathbf{x}^\star)\|^2] &= 2 \mathbb{E} [D_{\frac{1}{2}\|\cdot\|^2}(\nabla f_{\xi,i}(\mathbf{x}^{(t)}), \nabla f_{\xi,i}(\mathbf{x}^\star))] \\ &\leq \frac{2}{\zeta^{-1}} \mathbb{E} [D_{f_{\xi,i}^*}(\nabla f_{\xi,i}(\mathbf{x}^{(t)}), \nabla f_{\xi,i}(\mathbf{x}^\star))] \\ &\leq 2\zeta \mathbb{E} [D_{f_{\xi,i}}(\mathbf{x}^\star, \mathbf{x}^{(t)})] \\ &= 2\zeta D_f(\mathbf{x}^\star, \mathbf{x}^{(t)})\end{aligned}\tag{C.17}$$

For the expected gradient term, we can use that:

$$\|\nabla f(\mathbf{x}^{(t)})\|_{\mathbf{M}}^2 \leq \|\nabla f(\mathbf{x}^{(t)}) - \nabla f(\mathbf{x}^\star)\|^2 \leq 2LD_f(\mathbf{x}^\star, \mathbf{x}^{(t)}),$$

so that in the end,

$$\mathbb{E} [\|\nabla f_\xi(\mathbf{x}^{(t)})\|_{\mathbf{M}}^2] \leq 4(\zeta M_0 + L)D_f(\mathbf{x}^\star, \mathbf{x}^{(t)}) + 2\sigma_{\mathbf{M}}^2,\tag{C.18}$$

where  $M_0 = \max_i \mathbf{M}_{ii}$ , and  $\mathbb{E} [\|\nabla f_\xi(\mathbf{x}^*)\|_{\mathbf{M}}^2] \leq \sigma_{\mathbf{M}}^2$ , the locally averaged variance at optimum. Plugging this into the main equation, we obtain that:

$$\begin{aligned} \mathbb{E} [\|\mathbf{x}^{(t+1)} - \mathbf{x}^*\|_{\mathbf{M}}^2] &\leq \|\mathbf{x}^{(t)} - \mathbf{x}^*\|_{\mathbf{M}}^2 - \eta\mu\|\mathbf{x}^{(t)} - \mathbf{x}^*\|_{\mathbf{M}^2}^2 + 2\eta^2\sigma_{\mathbf{M}}^2 \\ &\quad - 2\eta(1 - 2\eta[\zeta M_0 + L]) D_f(\mathbf{x}^*, \mathbf{x}^{(t)}) + \frac{L}{2}\|\mathbf{M}\mathbf{x}^{(t)} - \mathbf{x}^{(t)}\|^2 \end{aligned}$$

The last step is to write that  $\mathbf{M}^2 = \mathbf{M} - \mathbf{M}\mathbf{L}_{\mathbf{M}}$ , so that:

$$\begin{aligned} \mathbb{E} [\|\mathbf{x}^{(t+1)} - \mathbf{x}^*\|_{\mathbf{M}}^2] &\leq (1 - \eta\mu)\|\mathbf{x}^{(t)} - \mathbf{x}^*\|_{\mathbf{M}}^2 + \eta\mu\|\mathbf{x}^{(t)} - \mathbf{x}^*\|_{\mathbf{M}\mathbf{L}_{\mathbf{M}}}^2 + 2\eta^2\sigma_{\mathbf{M}}^2 \\ &\quad - 2\eta(1 - 2\eta[\zeta M_0 + L]) D_f(\mathbf{x}^*, \mathbf{x}^{(t)}) + \eta L\|\mathbf{M}\mathbf{x}^{(t)} - \mathbf{x}^{(t)}\|^2 \end{aligned}$$

At this point, we can use that  $\mathbf{M}\mathbf{L}_{\mathbf{M}} \leq \mathbf{I}/4$ ,

$$\mu\|\mathbf{x}^{(t)} - \mathbf{x}^*\|_{\mathbf{M}\mathbf{L}_{\mathbf{M}}}^2 \leq \frac{\mu}{4}\|\mathbf{x}^{(t)} - \mathbf{x}^*\|^2 \leq \frac{1}{2}D_f(\mathbf{x}^*, \mathbf{x}^{(t)}), \quad (\text{C.19})$$

so that

$$\begin{aligned} \mathbb{E} [\|\mathbf{x}^{(t+1)} - \mathbf{x}^*\|_{\mathbf{M}}^2] &\leq (1 - \eta\mu)\|\mathbf{x}^{(t)} - \mathbf{x}^*\|_{\mathbf{M}}^2 + \eta L\|\mathbf{M}\mathbf{x}^{(t)} - \mathbf{x}^{(t)}\|^2 \\ &\quad - 2\eta(3/4 - 2\eta[\zeta M_0 + L]) D_f(\mathbf{x}^*, \mathbf{x}^{(t)}) + 2\eta^2\sigma_{\mathbf{M}}^2. \end{aligned} \quad (\text{C.20})$$

**Distance to consensus** We now bound the distance to consensus in the case of a communication update. More specifically, we write that:

$$\mathbb{E} [\|\mathbf{x}^{(t+1)}\|_{\mathbf{L}_{\mathbf{M}}}^2] = \|\mathbf{x}^{(t)}\|_{\mathbf{L}_{\mathbf{M}}}^2 - 2\eta\nabla f(\mathbf{x}^{(t)})^\top \mathbf{L}_{\mathbf{M}}\mathbf{x}^{(t)} + \eta^2 \mathbb{E} [\|\nabla f_{\xi_t}(\mathbf{x}^{(t)})\|_{\mathbf{L}_{\mathbf{M}}}^2]$$

Then, we develop the middle term as:

$$\begin{aligned} -\nabla f(\mathbf{x}^{(t)})^\top \mathbf{L}_{\mathbf{M}}\mathbf{x}^{(t)} &= -\nabla f(\mathbf{x}^{(t)})^\top (\mathbf{I} - \mathbf{M})\mathbf{x}^{(t)} \\ &= \nabla f(\mathbf{x}^{(t)})^\top (\mathbf{M}\mathbf{x}^{(t)} - \mathbf{x}^{(t)}) \\ &= -D_f(\mathbf{M}\mathbf{x}^{(t)}, \mathbf{x}^{(t)}) + f(\mathbf{M}\mathbf{x}^{(t)}) - f(\mathbf{x}^{(t)}) \\ &\leq -\frac{\mu}{2}\|\mathbf{M}\mathbf{x}^{(t)} - \mathbf{x}^{(t)}\|^2 + f(\mathbf{M}\mathbf{x}^{(t)}) - f(\mathbf{x}^{(t)}) \end{aligned}$$

By convexity of  $f$  (since the expected function is the same for all workers), we have that

$$f(\mathbf{M}\mathbf{x}^{(t)}) \leq f(\mathbf{x}^{(t)}). \quad (\text{C.21})$$

We finally decompose  $\mathbf{L}_{\mathbf{M}}^2 = \mathbf{L}_{\mathbf{M}}(\mathbf{I} - \mathbf{M})$ , so that:

$$-2\eta\nabla f(\mathbf{x}^{(t)})^\top \mathbf{L}_{\mathbf{M}}\mathbf{x}^{(t)} \leq -\eta\mu\|\mathbf{x}^{(t)} - \mathbf{x}^*\|_{\mathbf{L}_{\mathbf{M}}}^2 + \eta\mu\|\mathbf{x}^{(t)}\|_{\mathbf{M}\mathbf{L}_{\mathbf{M}}}^2$$

For the noise term, we obtain exactly the same derivations as in the previous setting, but this time with matrix  $\mathbf{L}_{\mathbf{M}} = \mathbf{I} - \mathbf{M}$  instead. Using the same bounding, and  $M_{\min} = \min_i (\mathbf{M})_{ii}$ , we thus obtain:

$$\mathbb{E} [\|\nabla f_\xi(\mathbf{x}^{(t)})\|_{\mathbf{L}_{\mathbf{M}}}^2] \leq 4(\zeta(1 - M_{\min}) + L)D_f(\mathbf{x}^*, \mathbf{x}^{(t)}) + 2\sigma_{\mathbf{L}_{\mathbf{M}}}^2. \quad (\text{C.22})$$

In particular, we have that:

$$\begin{aligned} \mathbb{E} \left[ \|\mathbf{x}^{(t+1)} - \mathbf{x}^*\|_{\mathbf{L}_M}^2 \right] &\leq (1 - \eta\mu) \|\mathbf{x}^{(t)} - \mathbf{x}^*\|_{\mathbf{L}_M}^2 + \eta\mu \|\mathbf{x}^{(t)}\|_{\mathbf{M}\mathbf{L}_M}^2 + 2\eta^2 \sigma_{\mathbf{L}_M}^2 \\ &\quad + 4\eta^2 (\zeta(1 - M_{\min}) + L) D_f(\mathbf{x}^*, \mathbf{x}^{(t)}). \end{aligned}$$

Similarly to before, we use that

$$\mu \|\mathbf{x}^{(t)}\|_{\mathbf{M}\mathbf{L}_M}^2 = \mu \|\mathbf{x}^{(t)} - \mathbf{x}^*\|_{\mathbf{M}\mathbf{L}_M}^2 \leq \frac{\mu}{4} \|\mathbf{x}^{(t)} - \mathbf{x}^*\|^2 \leq \frac{1}{2} D_f(\mathbf{x}^*, \mathbf{x}^{(t)}), \quad (\text{C.23})$$

so that for computation updates, the distance to consensus evolves as:

$$\mathbb{E} \left[ \|\mathbf{x}^{(t+1)} - \mathbf{x}^*\|_{\mathbf{L}_M}^2 \right] \leq (1 - \eta\mu) \|\mathbf{x}^{(t)} - \mathbf{x}^*\|_{\mathbf{L}_M}^2 + 2\eta \left[ \frac{1}{4} + 2\eta(\zeta(1 - M_{\min}) + L) \right] D_f(\mathbf{x}^*, \mathbf{x}^{(t)}) + 2\eta^2 \sigma_{\mathbf{L}_M}^2 \quad (\text{C.24})$$

Combining Equation (C.24) with Equation (C.20) leads to:

$$\begin{aligned} \mathcal{L}^{(t+1)} &\leq (1 - \eta\mu) \mathcal{L}_t + \eta L \|\mathbf{M}\mathbf{x}^{(t)} - \mathbf{x}^{(t)}\|^2 + 2\eta^2 \tilde{\sigma}^2 \\ &\quad - \eta(1 - 4\eta[\zeta(M_0 + \omega(1 - M_{\min})) + (1 + \omega)L]) D_f(\mathbf{x}^*, \mathbf{x}^{(t)}), \end{aligned} \quad (\text{C.25})$$

with  $\tilde{\sigma}^2 = \sigma_{\mathbf{M}}^2 + \omega\sigma_{\mathbf{L}_M}^2$ .

**2 - Communication updates** We write:

$$\begin{aligned} \|\mathbf{x}^{(t+1)} - \mathbf{x}^*\|_{\mathbf{L}_M}^2 &= \|\mathbf{x}^{(t)} - \mathbf{x}^*\|_{\mathbf{W}\mathbf{L}_M\mathbf{W}}^2 \\ &\leq \|\mathbf{x}^{(t)} - \mathbf{x}^*\|_{\mathbf{W}\mathbf{L}_M}^2 \\ &= \|\mathbf{x}^{(t)} - \mathbf{x}^*\|_{\mathbf{L}_M}^2 - \|\mathbf{x}^{(t)} - \mathbf{x}^*\|_{\mathbf{L}_M\mathbf{W}}^2 \end{aligned}$$

For distance to optimum part in communication update, we obtain:

$$\|\mathbf{x}^{(t+1)} - \mathbf{x}^*\|_{\mathbf{M}}^2 = \|\mathbf{x}^{(t)} - \mathbf{x}^*\|_{\mathbf{W}\mathbf{M}\mathbf{W}}^2 \leq \|\mathbf{x}^{(t)} - \mathbf{x}^*\|_{\mathbf{M}}^2 \quad (\text{C.26})$$

We now introduce  $\beta$ , the strong convexity of  $\mathbf{L}_W = \mathbf{I} - \mathbf{W}$  relative to  $\mathbf{L}_M$ :

$$\mathbf{L}_W \geq \beta \mathbf{L}_M. \quad (\text{C.27})$$

Therefore, we obtain that for communication updates,

$$\mathcal{L}^{(t+1)} \leq \mathcal{L}_t - \omega\beta \|\mathbf{x}^{(t)} - \mathbf{x}^*\|_{\mathbf{L}_M}^2. \quad (\text{C.28})$$

**Putting terms back together** We now put everything together, assuming that communication steps happen with probability  $p$  (and so computations steps with probability  $1 - p$ ). Thus, we mix Equations (C.25) and (C.28) to obtain:

$$\begin{aligned} \mathbb{E} \left[ \mathcal{L}^{(t+1)} \right] &\leq (1 - (1 - p)\eta\mu) \mathcal{L}_t + 2(1 - p)\eta^2 \tilde{\sigma}^2 \\ &\quad + [(1 - p)\eta L - \omega p\beta] \|\mathbf{x}^{(t)}\|_{\mathbf{L}_M}^2 \\ &\quad - \eta(1 - p)(1 - 4\eta[\zeta(M_0 + \omega(1 - M_{\min})) + (1 + \omega)L]) D_f(\mathbf{x}^*, \mathbf{x}^{(t)}). \end{aligned}$$

In particular, we obtain the linear decrease of the Lyapunov  $\mathcal{L}_t$  under the following conditions:

$$\eta \leq \frac{\omega\beta}{L} \frac{p}{1-p}$$

$$\eta \leq \frac{1}{4(\zeta[M_0 + \omega(1 - M_{\min})] + (1 + \omega)L)}$$

Under these conditions, we have that

$$\mathbb{E} [\mathcal{L}^{(t+1)}] \leq (1 - (1 - p)\eta\mu)\mathcal{L}_t + (1 - p)\eta^2\tilde{\sigma}^2, \quad (\text{C.29})$$

and we can simply chain this relation to finish the proof of the theorem.  $\square$

**Convex case** In the convex case ( $\mu = 0$ ), the proof is very similar, except that we keep the  $D_f(\mathbf{M}\mathbf{x}^{(t)}, \mathbf{x}^*)$  term from Equation (C.16). In particular, under the same step-size conditions as the strongly convex case, this leads to:

$$\mathbb{E} [\mathcal{L}^{(t+1)}] \leq \mathcal{L}_t + 2(1 - p)\eta^2\tilde{\sigma}^2 - \eta(1 - p)D_f(\mathbf{M}\mathbf{x}^{(t)}, \mathbf{x}^*). \quad (\text{C.30})$$

This leads to:

$$\mathbb{E} \left[ \frac{1}{T} \sum_{t=0}^{T-1} D_f(\mathbf{M}\mathbf{x}^{(t)}, \mathbf{x}^*) \right] \leq \frac{1}{1-p} \frac{\mathcal{L}_0}{\eta T} + 2\eta\tilde{\sigma}^2, \quad (\text{C.31})$$

which finishes the proof of the theorem.

**Evaluating  $\beta$**  There are two important graph quantities:  $M_0$  and  $\beta$ . If we choose  $M$  as in Equation (4.9), then its eigenvalues are equal to  $\frac{(1-\gamma)\lambda_i^2}{1-\gamma\lambda_i^2}$ , where  $\lambda_i$  is the  $i$ -th eigenvalue of  $\mathbf{W}$ . Therefore,

$$\lambda_i^{\mathbf{L}_M} = \frac{1 - \lambda_i^2}{1 - \gamma\lambda_i^2}. \quad (\text{C.32})$$

In particular, we have that for all  $i$ ,

$$1 - \lambda_i \geq \beta \frac{1 - \lambda_i^2}{1 - \gamma\lambda_i^2}, \quad (\text{C.33})$$

so that we can take

$$\beta = \frac{1 - \gamma\lambda_2^2}{1 + \lambda_2} \geq \frac{1 - \gamma\lambda_2}{2}, \quad (\text{C.34})$$

where we use  $\lambda_2 \leq 1$  to simplify the results. In particular,  $\beta$  does not depend on the spectral gap of  $\mathbf{W}$  (which is equal to  $1 - \lambda_2$ ) as long as  $\gamma$  is not too large. Yet, an interesting phenomenon happens: *a larger graph also implies more effective neighbors for a given  $\gamma$ .*

**Choice of  $\omega$**  A reasonable value for  $\omega$  is to simply take it as  $\omega = M_0$ . Indeed,

- The second condition almost does not benefit from  $\omega \leq M_0$  (factor 2 at most).
- If the first condition dominates, such that taking  $\omega \geq M_0$  would loosen it, then instead one can reduce  $\gamma$ . This will lead to a higher value for both  $M_0$  (and so for  $\omega$ ) and  $\beta$ . Note that, again, increasing  $M_0$  does not make the second condition stronger than what it would have been with just increasing  $\omega$  by more than a factor 2.

With this choice, we thus obtain that:

$$\eta \leq \min \left( \frac{M_0 \beta}{L} \frac{p}{1-p}, \frac{1}{4(M_0 \zeta (2 - M_{\min}) + (1 + M_0)L)} \right), \quad (\text{C.35})$$

and Theorem VIII is obtained by taking  $M_0 \leq 1$  and  $M_{\min} \geq 0$ .

### C.4.3 Obtaining Corollary IV

In this section, we discuss the derivations leading to Corollary IV. To do so, we start by making the simplifying assumption that

$$\frac{\zeta}{n} \geq L. \quad (\text{C.36})$$

Using this, and writing  $n_{\mathbf{W}}(\gamma) = 1/M_0$ , the condition from Equation (C.12) simplifies to:

$$\eta \leq \frac{Ln_{\mathbf{W}}(\gamma)}{16\zeta}. \quad (\text{C.37})$$

We always want this condition to be tight, and not Equation (C.11) the communication one, which is only there to allow us to use larger values of  $n_{\mathbf{W}}(\gamma)$ . In particular, we want that:

$$\frac{Ln_{\mathbf{W}}(\gamma)}{16\zeta} \leq \frac{\beta}{n_{\mathbf{W}}(\gamma)L}. \quad (\text{C.38})$$

When we increase  $\gamma$ ,  $n_{\mathbf{W}}(\gamma)$  increases and  $\beta$  decreases. We thus want to take the highest  $\gamma$  such that (C.38) is verified (potentially with an equality if  $n_{\mathbf{W}}(\gamma) < n$ ).

### C.4.4 Deterministic algorithm

So far, we have analyzed the randomized variant of D-SGD, in which at each step, there is a coin flip to decide whether to perform a communication or computation step. We now show how to extend the analysis to the case in which:

$$\mathbf{x}^{(t+1)} = \mathbf{W}\mathbf{x}^{(t)} - \eta \nabla f_{\xi}(\mathbf{W}\mathbf{x}^{(t)}) \quad (\text{C.39})$$

Note that D-SGD is often presented as  $\mathbf{x}_{t+1} = \mathbf{W}(\mathbf{x}_t - \eta \nabla f_{\xi}(\mathbf{x}_t))$ , but it turns out that the analysis is easier when considering it in the form of Equation (C.39). Yet, it comes down to the same algorithm (alternating communication and computation steps), and the difference simply is whether the error is evaluated after a communication step or a local gradient step. The results in the previous section did not depend on the value of  $\mathbf{x}_t$ , so we can perform the same derivations with  $\mathbf{W}\mathbf{x}_t$  instead of  $\mathbf{x}_t$ , so that Equation (C.25) now writes:

$$\begin{aligned} \mathcal{L}(\mathbf{x}^{(t+1)}) &= (1 - \eta\mu)\mathcal{L}(\mathbf{W}\mathbf{x}^{(t)}) + \eta L \|\mathbf{M}\mathbf{W}\mathbf{x}^{(t)} - \mathbf{W}\mathbf{x}^{(t)}\|^2 + 2\eta^2 \tilde{\sigma}^2 \\ &\quad - \eta(1 - 4\eta[\zeta(M_0 + \omega(1 - M_{\min})) + (1 + \omega)L]) D_f(\mathbf{x}^*, \mathbf{W}\mathbf{x}^{(t)}), \end{aligned} \quad (\text{C.40})$$

where  $\mathcal{L}(\mathbf{x}) = \|\mathbf{x} - \mathbf{x}^*\|_{\mathbf{M}}^2 + \omega\|\mathbf{x}\|_{\mathbf{L}_{\mathbf{M}}}^2$ , so that  $\mathcal{L}^{(t)} = \mathcal{L}(\mathbf{x}^{(t)})$ . In particular, choosing  $\eta$  such that the second line is always negative (as before) leads to:

$$\mathcal{L}(\mathbf{x}^{(t+1)}) = (1 - \eta\mu)\mathcal{L}(\mathbf{W}\mathbf{x}^{(t)}) + \eta L \|\mathbf{x}^{(t)}\|_{\mathbf{W}\mathbf{L}_{\mathbf{M}}^2\mathbf{W}}^2 + 2\eta^2 \tilde{\sigma}^2. \quad (\text{C.41})$$

Similarly, using Equation (C.28), we obtain that

$$\mathcal{L}(\mathbf{W}\mathbf{x}^{(t)}) \leq \mathcal{L}(\mathbf{x}^{(t)}) - \omega\beta\|\mathbf{x}^{(t)} - \mathbf{x}^*\|_{\mathbf{L}_M^2}^2. \quad (\text{C.42})$$

Combining Equations (C.41) and (C.42) and using that  $\mathbf{W}\mathbf{L}_M^2\mathbf{W} \preccurlyeq \mathbf{L}_M^2$ , we obtain:

$$\mathcal{L}^{(t+1)} \leq (1 - \eta\mu)\mathcal{L}^{(t)} + (\eta L - (1 - \eta\mu)\omega\beta)\|\mathbf{x}^{(t)}\|_{\mathbf{L}_M^2}^2 + 2\eta^2\tilde{\sigma}^2. \quad (\text{C.43})$$

Thus, we obtain similar guarantees (up to a factor  $1 - \eta\mu$  which is small) for the deterministic and randomized algorithms. Note that in this case, constant  $\beta$  can be replaced by a slightly better constant  $\tilde{\beta}$  which would be such that:

$$\mathbf{L}_M\mathbf{L}_W \geq \tilde{\beta} \mathbf{W}\mathbf{L}_M^2\mathbf{W}. \quad (\text{C.44})$$

## C.5 Cifar-10 experimental setup

Table C.2 describes the details of our experiments with D-SGD with VGG-11 on Cifar-10.

Table C.2: Default experimental settings for Cifar-10/VGG-11

Dataset	Cifar-10 [Krizhevsky et al.]
Data augmentation	Random horizontal flip and random $32 \times 32$ cropping
Data normalization	Subtract mean (0.4914, 0.4822, 0.4465) and divide standard deviation (0.2023, 0.1994, 0.2010)
Architecture	VGG-11 [Simonyan and Zisserman, 2015]
Training objective	Cross entropy
Evaluation objective	Top-1 accuracy
Number of workers	32 (unless otherwise specified)
Topology	Ring (unless otherwise specified)
Gossip weights	Metropolis-Hastings (1/3 for ring, $w_{ij} = 1/(\max(n_i, n_j) + 1)$ , worker $i$ has $n_i$ direct neighbors)
Data distribution	Identical: workers can sample from the whole dataset
Sampling	With replacement (i.i.d.), <i>no</i> shuffled passes
Batch size	16 patches per worker
Momentum	0.9 (heavy ball / PyTorch default)
Learning rate	Exponential grid or tuned for lowest training loss after 25 epochs
LR decay	Step-wise, $\times 0.1$ at epoch 75% and 90% of training
LR warm-up	None
# Epochs	100 (full training) or only 25 (initial phase), based on total number of gradient accesses across workers
Weight decay	$10^{-4}$
Normalization scheme	no normalization layers
Exponential moving average	$\mathbf{x}_{\text{ema}}^{(t)} = 0.95 \mathbf{x}_{\text{ema}}^{(t-1)} + 0.05 \mathbf{x}^{(t)}$ . This influences evaluation, not training
Repetitions per training	Just 1 per learning rate, but experiments are very consistent across similar learning rates
Reported metrics	<i>Loss after 2.5k steps:</i> to reduce noise, we take two measures: (i) we use exponential moving average of the model parameters, and (ii) we fit a parametric model $\log(l) = at + b$ to the 25 loss evaluations $(t, l)$ closest to $t = 2500$ . We then evaluate this function at $t = 2500$ .

## C.6 Additional experiments

In the main paper, we have focussed on the training loss in the initial phase of training of Cifar-10. We do find that our findings there do correlate with test accuracy after a complete training with 100 epochs. Figure C.4 shows the test accuracy as training progresses, for plots ordered by improving training loss after 2.5k steps.

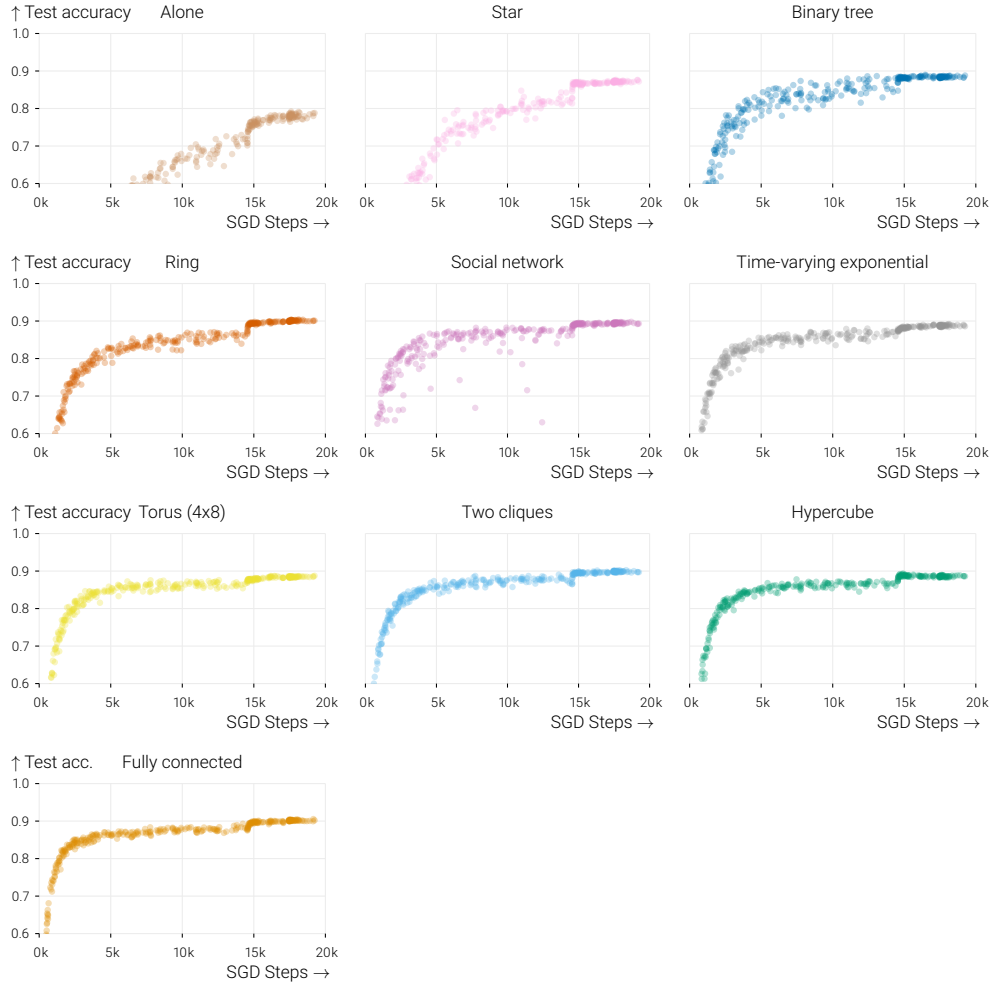


Figure C.4: Test accuracy over the course of training a VGG-11 network on Cifar-10. See appendix C.5 for all details on the experimental setup. The plots are ordered by improving training loss after 2.5k SGD steps. This ordering correlates well with the speed of improvements in test accuracy.



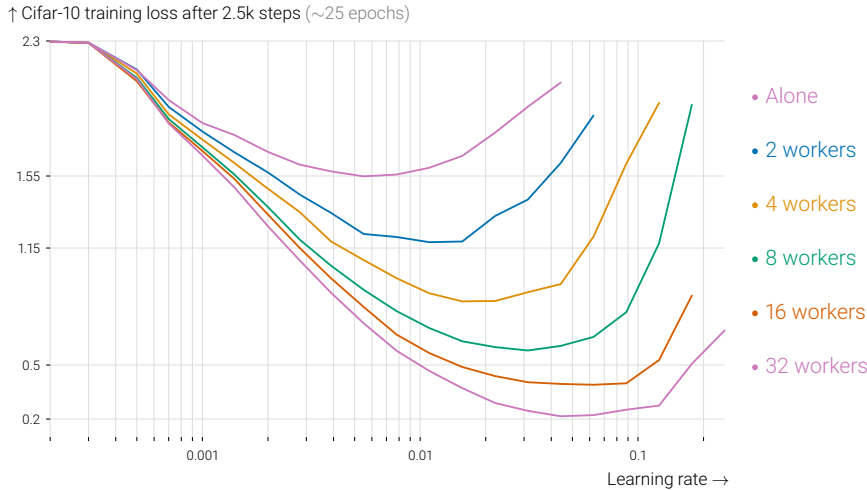


Figure C.5: Training loss reached after 2.5k SGD steps with fully-connected topologies of varying size. Averaging with more workers speeds up convergence for fixed learning rates, but also allows larger learning rates to be used. This plot serves as a reference for fig. 4.4, which shows similar plots for a variety of graph topologies.

### C.6.1 Results on Fashion MNIST

We replicated our main experiments (Cifar-10/VGG) on another dataset and another network architecture. We chose for the Fashion MNIST dataset [Xiao et al., 2017] and a simple multi-layer perceptron architecture with one hidden layer of 5000 neurons and ReLU activations. We list the details of our experimental setup in table C.3. We varied two key parameters compared to our Cifar-10 results: we used 64 workers instead of 32, and used SGD *without* momentum and *without* weight decay. Because this task is easier than Cifar-10, the initial phase where both training and test loss converge at similar rates is shorter. We therefore consider the first 500 steps as the ‘initial phase’, as opposed to 2500.

Figures C.6 and C.7 correspond to figures 4.4 and 4.6 from the main paper. We find that the conclusions from the paper also hold in this different experimental setting.

### C.6.2 Heterogeneous data

While our experimental and theoretical data only describe the setting in which workers optimize objectives with a shared optimum, we believe that our insights are meaningful for heterogeneous settings as well. With heterogeneous data, we observe two regimes: in the beginning of training, when the worker’s distant optima are in a similar direction, everything behaves identical to the homogeneous setting. In this regime, our insights seem to apply directly. Heterogeneity only plays a role later during the training, when it leads to conflicting gradient directions. This behavior is illustrated on a toy problem in fig. C.8. We run D-SGD on our isotropic quadratic toy problem ( $d = 100$ ,  $n = 32$ ), but where the optima are removed from zero as a normal distribution with standard deviations 0,  $10^{-7}$ , and  $10^{-3}$  respectively. The (constant) learning rates are tuned for each topology in the homogeneous setting.

Table C.3: Experimental settings for Fashion MNIST. Differences with Cifar-10 in **red**.

Dataset	<b>Fashion MNIST</b> [Xiao et al., 2017]
Data augmentation	<b>None</b>
Data normalization	Subtract mean 0.2860 and divide standard deviation 0.3530
Architecture	<b>MLP</b> ( $28 \times 28 \rightarrow \text{ReLU} \rightarrow 5000 \rightarrow \text{ReLU} \rightarrow 10$ )
Training objective	Cross entropy
Evaluation objective	Top-1 accuracy
Number of workers	<b>64</b> (unless otherwise specified)
Topology	Ring (unless otherwise specified)
Gossip weights	Metropolis-Hastings (1/3 for ring, $w_{ij} = 1/(\max(n_i, n_j) + 1)$ , worker $i$ has $n_i$ direct neighbors)
Data distribution	Identical: workers can sample from the whole dataset
Sampling	With replacement (i.i.d.), <i>no</i> shuffled passes
Batch size	16 patches per worker
Momentum	<b>0.0</b>
Learning rate	Exponential grid or tuned for lowest training loss after <b>500 steps</b>
LR decay	None, in the initial phase of training
LR warm-up	None
# Epochs	500 steps
Weight decay	<b>0</b>
Normalization scheme	no normalization layers
Exponential moving average	$\mathbf{x}_{\text{ema}}^{(t)} = 0.95\mathbf{x}_{\text{ema}}^{(t-1)} + 0.05\mathbf{x}^{(t)}$ . This influences evaluation, not training
Repetitions per training	Just 1 per learning rate, but experiments are very consistent across similar learning rates
Reported metrics	<i>Loss after 500 steps</i> : to reduce noise, we take two measures: (I) we use exponential moving average of the model parameters, and (II) we fit a parametric model $\log(l) = at + b$ to the 25 loss evaluations $(t, l)$ closest to $t = 500$ . We then evaluate this function at $t = 500$ .

### C.6.3 The role of $\gamma$ in the experiments

In fig. 4.5, we optimize  $\gamma$  independently for each topology, minimizing the Mean Squared Error between the normalized covariance matrix measured from checkpoints of Cifar-10 training and the covariance in a random walk with the decay parameter  $\gamma$ . The bottom two rows of fig. C.10 below show how fig. 4.5 would change, if you used a  $\gamma$  that is either much too low, or too high.

In fig. 4.6, we choose a value of  $\gamma$  (shared between all topologies) that yields a good correspondence between the performance of fully connected topologies (with 2, 4, 8, 16 and 32 workers) and the other topologies. We opt for sharing a single  $\gamma$  here, to test whether this metric could have predictive power for the quality of graphs. Figure C.9 below shows how the figure changes if you use a value of  $\gamma$  that is either much too low, or much too high.

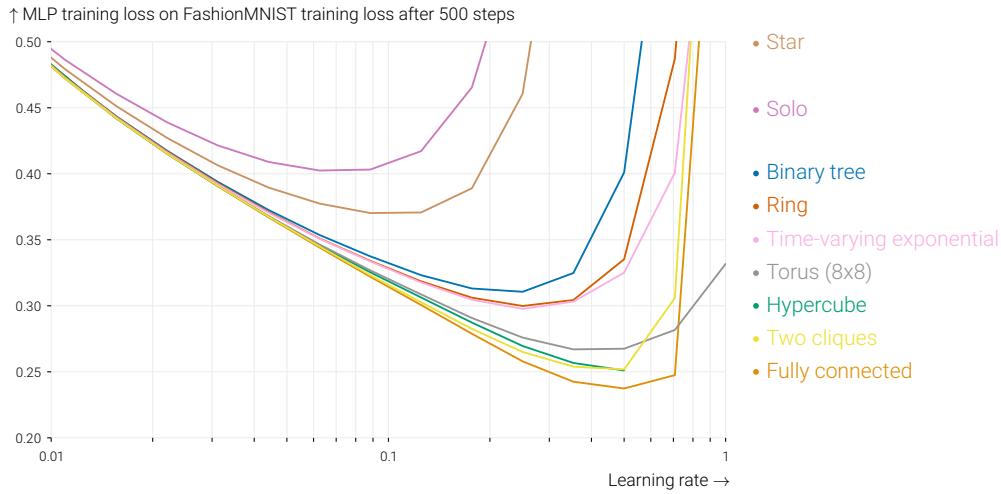


Figure C.6: Training loss reached after 500 SGD steps with a variety of 64-worker graph topologies. In all cases, averaging yields a small increase in speed for small learning rates, but a large gain over training alone comes from being able to increase the learning rate. While the star has a better spectral gap (0.0156) than the ring (0.0032), it performs worse, and does not allow large learning rates.

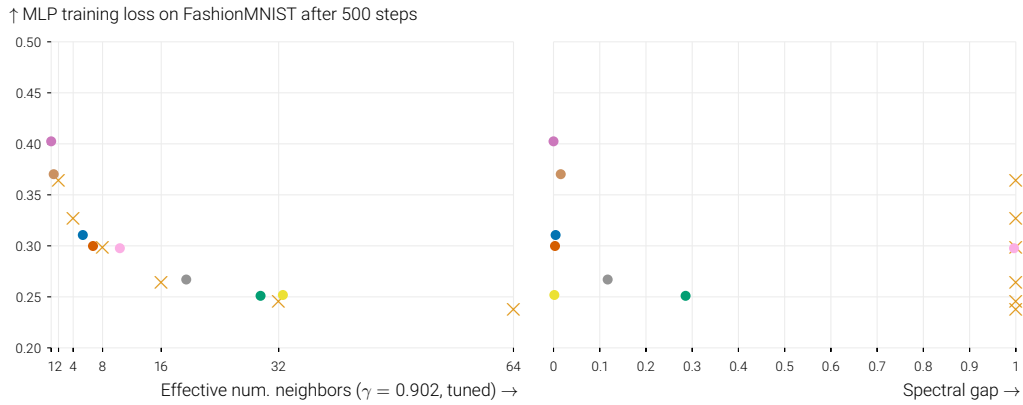


Figure C.7: Fashion MNIST training loss after 500 steps for all studied topologies with their optimal learning rates. Colors match fig. C.6, and  $\times$  indicates fully-connected graphs with varying number of workers. After fitting a decay parameter  $\gamma = 0.902$  that captures problem specifics, the effective number of neighbors (left) as measured by variance reduction in a random walk (like in section 4.4) explains the relative performance of these graphs much better than the spectral gap of these topologies (right).

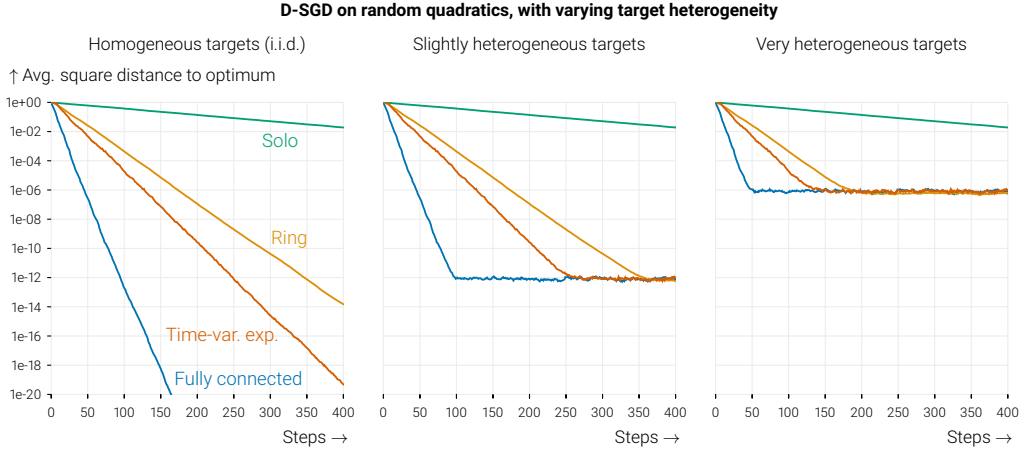


Figure C.8: Convergence curves on our isotropic random quadratics problem (section 4.4, with  $d = 100$ ,  $n = 32$ ), but where the optima are removed from zero as a zero-mean normal distribution with standard deviations 0,  $10^{-7}$ , and  $10^{-3}$  respectively. Constant learning rates are tuned independently for each topology in the homogeneous setting. Heterogeneity does not affect the initial phase of training, and our insights about maximum learning rates and the quality of communication topologies hold in this regime.

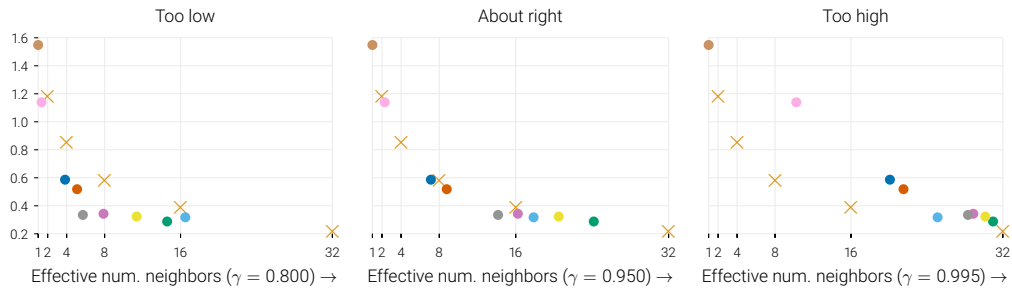


Figure C.9: Extension of fig. 4.6, demonstrating how the fit changes if you use a value of  $\gamma$  that is either too low (left) or too high (right).

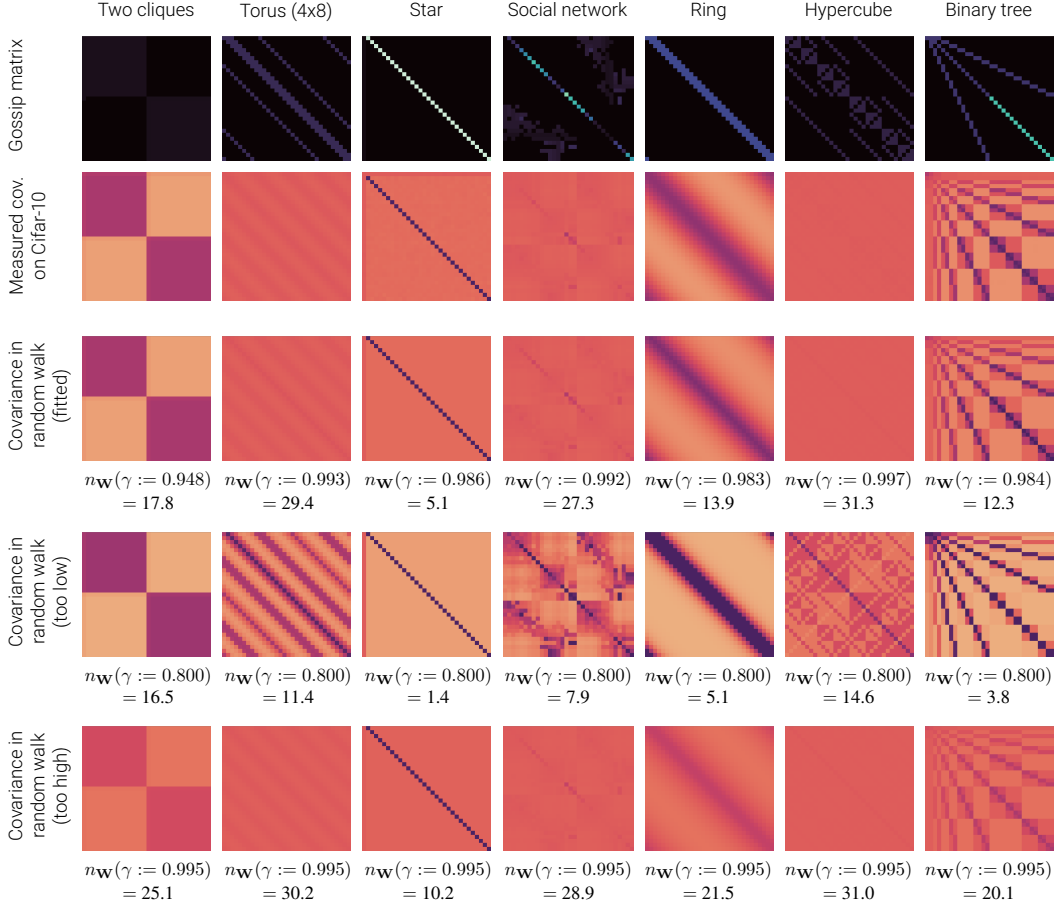


Figure C.10: Extension of fig. 4.5. Measured covariance in Cifar-10 (second row) between workers using various graphs (top row). After 10 epochs, we store a checkpoint of the model and train repeatedly for 100 SGD steps, yielding 100 models for 32 workers. We show normalized covariance matrices between the workers. These are very well approximated by the covariance in the random walk process of section 4.4 (third row). We print the fitted decay parameters and corresponding ‘effective number of neighbors’. The bottom two rows show how fig. 4.5 would change, if you used a  $\gamma$  that is either much too low, or too high.



## Appendix D

# Appendix for RelaySGD

### D.1 Convergence Analysis of RelaySGD

Please refer to [Vogels et al., 2021] for the convergence analysis of RelaySGD.

### D.2 Detailed experimental setup

#### D.2.1 Cifar-10

See table D.1.

Table D.1: Default experimental settings for Cifar-10/VGG-11

Dataset	Cifar-10 [Krizhevsky et al.]
Data augmentation	random horizontal flip and random $32 \times 32$ cropping
Architecture	VGG-11 [Krizhevsky and Hinton, 2009]
Training objective	cross entropy
Evaluation objective	top-1 accuracy
Number of workers	16
Topology	SGP: time-varying exponential, RelaySGD: double binary trees, baselines: best of ring or double binary trees
Gossip weights	Metropolis-Hastings (1/3 for ring)
Data distribution	Heterogeneous, not shuffled, according to Dirichlet sampling procedure from Lin et al. [2021]
Batch size	32 patches per worker
Momentum	0.9 (Nesterov)
Learning rate	Tuned c.f. appendix D.3.1
LR decay	/10 at epoch 150 and 180
LR warm-up	Step-wise linearly within 5 epochs, starting from 0
# Epochs	200
Weight decay	$10^{-4}$
Normalization scheme	no normalization layer
Repetitions	3, with varying seeds
Reported metric	Worst result of any worker of the worker’s mean test accuracy over the last 5 epochs

#### D.2.2 ImageNet

See table D.2.

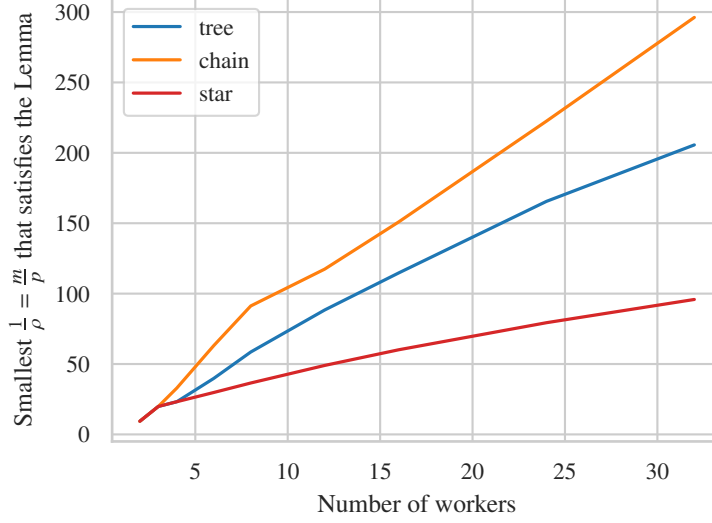


Figure D.1: Optimal ratios for  $\rho = p/m$  for Lemma 1 computed empirically for three common types of graph topologies.

### D.2.3 BERT finetuning

See table D.3.

### D.2.4 Random quadratics

We generate quadratics  $\frac{1}{n} \sum_{i=1}^n f_i(\mathbf{x})$  of  $\mathbf{x} \in \mathbb{R}^d$  where

$$f_i(\mathbf{x}) = \|\mathbf{A}_i \mathbf{x} + \mathbf{b}_i\|_2^2.$$

Here the local Hessian  $\mathbf{A}_i \in \mathbb{R}^{d \times d}$  control the shape of worker  $i$ 's local objective functions and the offset  $\mathbf{b}_i \in \mathbb{R}^d$  allows for shifting the worker's optimum. The generation procedure is as follows:

1. Sample  $\mathbf{A}_i \in \mathbb{R}^{d \times d}$  from an i.i.d. element-wise standard normal distribution, independently for each worker.
2. Control the smoothness  $L$  and strong-convexity constant  $\mu$ . Decompose  $\mathbf{A}_i = \mathbf{U}_i \mathbf{S}_i \mathbf{V}_i^\top$  using Singular Value Decomposition, and replace  $\mathbf{A}_i$  with  $\mathbf{A}_i \leftarrow \mathbf{U}_i \tilde{\mathbf{S}}_i \mathbf{V}_i^\top$ , where  $\tilde{\mathbf{S}}_i \in \mathbb{R}^{d \times d}$  is a diagonal matrix with diagonal entries  $[\mu, \frac{d-2}{d-1}\mu + \frac{1}{d-1}L, \dots, L]$ .
3. Control the heterogeneity  $\zeta_2$  by shifting worker's optima into random directions.
  - (a) Sample random directions  $\mathbf{d}_i \in \mathbb{R}^d$  from an i.i.d. element-wise standard normal distributions, independently for each worker.
  - (b) Instantiate a scalar  $s \leftarrow 1$  and optimize it using binary search:
  - (c) Move local optima by  $s\mathbf{d}_i$  by setting  $\mathbf{b}_i \leftarrow \mathbf{A}_i s\mathbf{d}_i$ .
  - (d) Move all optima  $\mathbf{b}_i \leftarrow \mathbf{b}_i - \mathbf{A}_i \mathbf{x}^*$  such that the global optimum  $\mathbf{x}^*$  remains at zero.



Table D.2: Default experimental settings for ImageNet

Dataset	ImageNet [Deng et al., 2009]
Data augmentation	random resized crop ( $224 \times 224$ ), random horizontal flip
Architecture	ResNet-20-EvoNorm [Liu et al., 2020, Lin et al., 2021]
Training objective	cross entropy
Evaluation objective	top-1 accuracy
Number of workers	16
Topology	SGP: time-varying exponential, RelaySGD: double binary trees, baselines: best of ring or double binary trees
Gossip weights	Metropolis-Hastings (1/3 for ring)
Data distribution	Heterogeneous, not shuffled, according to Dirichlet sampling procedure from Lin et al. [2021]
Batch size	32 patches per worker
Momentum	0.9 (Nesterov)
Learning rate	based on centralized training (scaled to $0.1 \times \frac{32 \times 16}{256}$ )
LR decay	/10 at epoch 30, 60, 80
LR warm-up	Step-wise linearly within 5 epochs, starting from 0.1
# Epochs	90
Weight decay	$10^{-4}$
Normalization layer	EvoNorm [Liu et al., 2020]
Repetitions	Just one
Reported metric	Mean of all worker’s test accuracies over the last 5 epochs

(e) Evaluate  $\zeta^2 = \frac{1}{n} \sum_{i=1}^n \|\nabla f_i(\mathbf{x}^*)\|_2^2$  and adjust the scale factor  $s$  until  $\zeta^2$  is as desired. Repeat from step (c).

- Control the initial distance to the optimum  $r_0$ . Sample a random vector for the optimum  $\mathbf{x}^*$  from an i.i.d. element-wise normal distribution and scale it to have norm  $r_0$ . Shift all worker’s optima in this direction by updating  $\mathbf{b}_i \leftarrow \mathbf{b}_i + \mathbf{A}_i \mathbf{x}^*$ .

## D.3 Hyperparameters and tuning details

### D.3.1 Cifar-10

For our image classification experiments on Cifar-10, we have independently tuned learning rates for each algorithm, at each data heterogeneity level  $\alpha$ , and separately for SGD with and without momentum. We followed the following procedure:

- We found an appropriate learning rate for centralized (all-reduce) training (by using the procedure below)
- Start the search from this learning rate. For RelaySGD, we apply a correction computed as in appendix D.4.1.
- Grid-search the learning rate by multiplying and dividing by powers of two. Try larger and smaller learning rates, until the best result found so far is sandwiched between two learning rates that gave worse results.
- Repeat the experiment with 3 random seeds.
- If any of those replicas diverged, reduce the learning rate by a factor two until it does.

For the experiments in table 5.1, we used the learning rates listed in table D.4.

Table D.3: Default experimental settings for BERT fine-tuning

Dataset	AG News [Zhang et al., 2015]
Data augmentation	none
Architecture	DistilBERT [Sanh et al., 2019]
Training objective	cross entropy
Evaluation objective	top-1 accuracy
Number of workers	16
Topology	restricted to a ring (chain for RelaySGD)
Gossip weights	Metropolis-Hastings (1/3 for ring)
Data distribution	Heterogeneous, not shuffled, according to Dirichlet sampling procedure from Lin et al. [2021]
Batch size	32 patches per worker
Adam $\beta_1$	0.9
Adam $\beta_2$	0.999
Adam $\epsilon$	$10^{-8}$
Learning rate	Tuned c.f. appendix D.3.3
LR decay	constant learning rate
LR warm-up	no warm-up
# Epochs	5
Weight decay	0
Normalization layer	LayerNorm [Ba et al., 2016]
Repetitions	3, with varying seeds
Reported metric	Mean of all worker’s test accuracies over the last 5 epochs

Table D.4: Learning rates used for Cifar-10/ VGG-11. Numbers between parentheses indicate the number of converged replications with this learning rate.

Algorithm	Topology	$\alpha = 1.00$ (most homogeneous)	$\alpha = 0.1$	$\alpha = .01$ (most heterogeneous)
All-reduce	fully connected	0.100 (3)	0.100 (3)	0.100 (3)
+momentum		0.100 (3)	0.100 (3)	0.100 (3)
<b>RelaySGD</b>	binary trees	1.200 (3)	0.600 (3)	0.300 (3)
+local momentum		0.600 (3)	0.300 (3)	0.150 (3)
D-SGD	ring	0.400 (3)	0.100 (3)	0.200 (3)
+quasi-global mom.		0.100 (3)	0.025 (3)	0.050 (3)
D <sup>2</sup> [Tang et al., 2018b]	ring	0.200 (3)	0.200 (3)	0.100 (3)
+local momentum		0.050 (3)	0.050 (3)	0.013 (3)
Stochastic gradient push	time-varying exponential	0.400 (3)	0.200 (3)	0.200 (3)
+local momentum		0.100 (3)	0.100 (3)	0.025 (3)

### D.3.2 ImageNet

Due to the high resource requirements, we did not tune the learning rate for our ImageNet experiments. We identified a suitable learning rate based on prior work, and used this for all experiments. For RelaySGD, we used the analytically computed learning rate correction from appendix D.4.1.

### D.3.3 BERT finetuning

For DistilBERT fine-tuning experiments on AG News, we have independently tuned learning rate for each algorithm. We search the learning rate in the grid of  $\{1e-5, 3e-5, 5e-5, 7e-5, 9e-5\}$ , and we extend the grid to ensure that the best hyperparameter lies in the middle of our search grids, otherwise we extend our search grid.

For the experiments in table 5.4, we used the learning rates listed in table D.5.

Table D.5: Tuned learning rates used for AG News / DistilBERT (table 5.4)

Algorithm	Topology	Learning rate
Centralized Adam	fully-connected	3e-5
<b>Relay-Adam</b>	chain	9e-4
D-SGD Adam	ring	1e-6
Quasi-global Adam [Lin et al., 2021]	ring	1e-6

### D.3.4 Random quadratics

For Figures 5.2 and 5.3, we tuned the learning rate for each compared method to reach a desired quality level as quickly as possible, using binary search. We made a distinction between methods that are expected to converge linearly, and methods that are expected to reach a plateau. For experiments with stochastic noise, we tuned a learning rate without noise first, and then lowered the learning rate if needed to reach a desirable plateau. Please see the supplied code for implementation details.

## D.4 Algorithmic details

### D.4.1 Learning-rate correction for RelaySGD

In D-SGD as well as all other algorithms we compared to, a gradient-based update  $\mathbf{u}_i^{(t)}$  from worker  $i$  at time  $t$  will eventually, as  $t \rightarrow \infty$  distribute uniformly with weights  $\frac{1}{n}$  over all workers. In RelaySGD, the update also distributes uniformly (typically much quicker), but it will converge to a weight  $\alpha \leq \frac{1}{n}$ . The constant  $\alpha$  is fixed throughout training and depends only on the network topology used. To correct for this loss in energy, you can scale the learning rate by a factor  $\frac{1}{\alpha n}$ .

Experimentally, we pre-compute  $\alpha$  for each architecture by initialing a *scalar* model for each worker to zero, updating the models to 1, and running RelaySGD until convergence with no further model updates. The worker will converge to the value  $\alpha$ . The correction factors that result from this procedure are illustrated in fig. D.2.

In our deep learning experiments, we find that for each learning rate were centralized SGD converges, RelaySGD with the corrected learning rate converges too. Note that this learning rate correction is only useful if you already have a tuned learning rate from centralized experiments, or experiments with algorithms such as D-SGD. If you start from scratch, tuning the learning rate for RelaySGD is no different from tuning the learning rate for any of the other algorithms.

### D.4.2 RelaySGD with momentum

RelaySGD follows Algorithm 5, but replaces the local update in line 3 with a local momentum. For Nesterov momentum with momentum-parameter  $\alpha$ , this is:

$$\begin{aligned}\mathbf{m}_i^{(t)} &= \alpha \mathbf{m}_i^{(t-1)} + \nabla f_i(\mathbf{x}_i^{(t)}) \quad (\text{initialize } \mathbf{m}_i^0 = 0) \\ \mathbf{x}_i^{(t+1/2)} &= \mathbf{x}_i^{(t)} - \gamma \left( \nabla f_i(\mathbf{x}_i^{(t)}) + \alpha \mathbf{m}_i^{(t)} \right).\end{aligned}$$

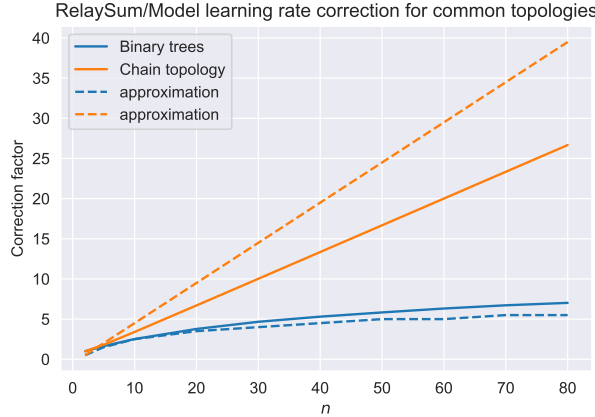


Figure D.2: This network-topology-dependent correction factor is computed as follows: Each worker initializes a scalar model to 0 and sends a single fixed value 1 as gradient update through the RelaySGD algorithm. For D-SGD and all-reduce, workers would converge to 1, but for RelaySGD, we lose some of this energy. If the workers converge to a value  $\alpha$ , we will scale the learning rate with  $1/\alpha$  for RelaySGD compared to all-reduce.

#### D.4.3 RelaySGD with Adam

Modifying RelaySGD (Algorithm 5) to use Adam is analogous to RelaySGD with momentum (appendix D.4.2). All Adam state is updated locally. We use the standard Adam implementation of PyTorch 1.18.

#### D.4.4 D<sup>2</sup> with momentum

We made slight modifications to the D<sup>2</sup> algorithm from Tang et al. [2018b] to allow time-varying learning rates and local momentum. The version we use is listed as Algorithm 12. Note that D<sup>2</sup> requires the smallest eigenvalue of the gossip matrix  $\mathbf{W}$  to be  $\geq -1/3$ . This property is satisfied for Metropolis-Hasting matrices used on rings and double binary trees, but it was not in our Social Network Graph experiment (fig. 5.3). For this reason, we used the gossip matrix  $(\mathbf{W} + \mathbf{I})/2$ , from the otherwise-equivalent Exact Diffusion algorithm [Yuan et al., 2019] on the social network graph.

#### D.4.5 Gradient Tracking

Algorithm 13 lists our implementation of Gradient Tracking from Lorenzo and Scutari [2016].

#### D.4.6 Stochastic Gradient Push with the time-varying exponential topology

Stochastic Gradient Push with the time-varying exponential topology from [Assran et al., 2019] demonstrates that decentralized learning algorithms can reduce communication in a data center setting where each node could talk to each other node. Algorithm 14 lists our implementation of this algorithm.

**Algorithm 12**  $D^2$  [Tang et al., 2018b] with momentum

---

**Input:**  $\forall i, \mathbf{x}_i^{(0)} = \mathbf{x}^{(0)}$ , learning rate  $\gamma$ , momentum  $\alpha$ , gossip matrix  $\mathbf{W} \in \mathbb{R}^{n \times n}$ ,  $\mathbf{c}_i^{(0)} = \mathbf{0} \in \mathbb{R}^d$ .

- 1: **for**  $t = 0, 1, \dots$  **do**
- 2:   **for** node  $i$  **in parallel**
- 3:     Update the local momentum buffer  $\mathbf{m}_i^{(t)} = \alpha \mathbf{m}_i^{(t-1)} + \nabla f_i(\mathbf{x}_i^{(t)})$ .
- 4:     Compute a local update  $\mathbf{u}_i^{(t)} = -\gamma(\nabla f_i(\mathbf{x}_i^{(t)}) + \alpha \mathbf{m}_i^{(t)})$ .
- 5:     Update the local model  $\mathbf{x}_i^{(t+1/2)} = \mathbf{x}_i^{(t)} + \mathbf{u}_i^{(t)} + \mathbf{c}_i^{(t)}$ .
- 6:     Average with neighbors:  $\mathbf{x}_i^{(t+1)} = \sum_{j \in \mathcal{N}_i} \mathbf{W}_{ij} \mathbf{x}_j^{(t+1/2)}$ .
- 7:     Update the local correction  $\mathbf{c}_i^{(t+1)} = \mathbf{x}_i^{(t+1)} - \mathbf{x}_i^{(t)} - \mathbf{u}_i^{(t)}$ .
- 8:   **end for**
- 9: **end for**

---

**Algorithm 13** Gradient Tracking [Lorenzo and Scutari, 2016]

---

**Input:**  $\forall i, \mathbf{x}_i^{(0)} = \mathbf{x}^{(0)}$ , learning rate  $\gamma$ , gossip matrix  $\mathbf{W} \in \mathbb{R}^{n \times n}$ ,  $\mathbf{c}_i^{(0)} = \mathbf{0} \in \mathbb{R}^d$ .

- 1: **for**  $t = 0, 1, \dots$  **do**
- 2:   **for** node  $i$  **in parallel**
- 3:     Compute a local update  $\mathbf{u}_i^{(t)} = -\gamma \nabla f_i(\mathbf{x}_i^{(t)})$ .
- 4:     Update the local model  $\mathbf{x}_i^{(t+1/2)} = \mathbf{x}_i^{(t)} + \mathbf{u}_i^{(t)} + \mathbf{c}_i^{(t)}$ .
- 5:     Average with neighbors:  $\mathbf{x}_i^{(t+1)} = \sum_{j \in \mathcal{N}_i} \mathbf{W}_{ij} \mathbf{x}_j^{(t+1/2)}$ .
- 6:     Update the correction and average:  $\mathbf{c}_i^{(t+1)} = \sum_{j \in \mathcal{N}_i} \mathbf{W}_{ij} (\mathbf{c}_i^{(t)} - \mathbf{u}_i^{(t)})$ .
- 7:   **end for**
- 8: **end for**

---

## D.5 Additional experiments on RelaySGD

### D.5.1 Rings vs double binary trees on Cifar-10

In our experiments that target data-center inspired scenarios where the network topology is arbitrarily selected by the user to save bandwidth, RelaySGD uses double binary trees to communicate. They use the same memory and bandwidth as rings (2 models sent/received per iteration) but their delays only scale with  $\log n$ , enabling RelaySGD, in theory, to run with very large numbers of workers  $n$ . Table D.6 shows that in our Cifar-10 experiments with 16 there are minor improvements from using double binary trees over rings. Our baselines D-SGD and  $D^2$ , however, perform significantly better on rings than on trees, so we use those results in the main paper.

### D.5.2 Scaling the number of workers on Cifar-10

In this experiment (table D.7), use momentum-SGD on 16, 32 and 64 workers compare the scaling of RelaySGD to SGP [Assran et al., 2019]. We fix the parameter  $\alpha$  that determines the level of data heterogeneity to  $\alpha = 0.01$ . Note that this level of  $\alpha$  could lead to more challenging heterogeneity when there are many workers (and hence many smaller local subsets of the data), compared to when there are few workers.

**Algorithm 14** Stochastic Gradient Push with time-varying exponential topology [Assran et al., 2019]

---

**Input:**  $\forall i, \mathbf{x}_i^{(0)} = \mathbf{x}^{(0)}$ , learning rate  $\gamma$ ,  $n = 2^k$  workers,  $t' = 0$ .











```

1: for  $t = 0, 1, \dots$  do
2:   for node  $i$  in parallel
3:      $\mathbf{x}_i^{(t+1/2)} = \mathbf{x}_i^{(t)} + \mathbf{u}_i^{(t)} - \gamma \nabla f_i(\mathbf{x}_i^{(t)})$ . (or momentum/Adam, like RelaySGD)
4:     for 2 communication steps to equalize bandwidth with RelaySGD do
5:       Compute an offset  $o = 2^{t' \bmod k}$ .
6:       Send  $\mathbf{x}_i^{(t+1/2)}$  to worker  $i - o$ .
7:       Receive and overwrite  $\mathbf{x}_i^{(t+1/2)} \leftarrow \frac{1}{2} (\mathbf{x}_i^{(t+1/2)} + \mathbf{x}_{i+o}^{(t+1/2)})$ .
8:        $t' \leftarrow t' + 1$ .
9:     end for
10:    Set  $\mathbf{x}_i^{(t+1)} = \mathbf{x}_i^{(t+1/2)}$ .
11:  end for
12: end for

```

---

Table D.6: Comparing the performance of the algorithms in table 5.1 on rings and double binary trees in the high-heterogeneity setting  $\alpha = 0.01$ . In both topologies, workers send and receive two full models per update step. With 16 workers, RelaySGD with momentum seems to benefit from double binary trees, RelaySGD has more consistently good results on a chain. We still opt for double binary trees based on their promise to scale to many workers. Other methods do not benefit from double binary trees over rings.

Algorithm	Ring (Chain for RelaySGD)	Double binary trees
<b>RelaySGD</b>	86.5% 	84.6% 
+local momentum	88.4% 	89.1% 
D-SGD	53.9% 	36.0% 
+quasi-global mom.	63.3% 	57.5% 
D <sup>2</sup>	38.2% 	did not converge
+local momentum	61.0% 	did not converge

### D.5.3 Independence of heterogeneity

The benefits of RelaySGD over some other methods shows most when workers have heterogeneous training objectives. Figure D.3 compares several algorithms with varying levels of data heterogeneity on synthetic quadratics on a ring topology with 32 workers. Like D<sup>2</sup>, RelaySGD converges linearly, and does not require more steps when the data becomes more heterogeneous. Note that, even though RelaySGD operates on a chain network instead of a ring, it is as fast as D<sup>2</sup>. On other topologies, such as a star topology, or on trees, RelaySGD can even be faster than D<sup>2</sup> (see Appendix D.5.4), while maintaining the same independence of heterogeneity.

### D.5.4 Star topology

On star-topologies, the set of neighbors of worker 0 is  $\{1, 2, \dots, n\}$  and the set of neighbors for every other worker is just  $\{0\}$ . While D<sup>2</sup> and RelaySGD are equally fast in the synthetic experiments on *ring* topologies in appendix D.5.3, RelaySGD is significantly faster on *star*

Table D.7: Scaling the number of workers in heterogeneous Cifar-10. The heterogeneity level  $\alpha = 0.01$  is kept constant, although it does change its meaning when the number of workers changes. RelaySGD scales at least well as Stochastic Gradient Push [Assran et al., 2019] (with equal communication budget). It is surprising that RelaySGD with 64 workers performs significantly better on a chain topology than on the double binary trees. This behavior does not match what our observations on quadratic toy-problems.













Algorithm	Topology	16 workers	32 workers	64 workers
All-reduce (baseline)	fully connected	89.5% 	88.9% 	87.2% 
RelaySGD	binary trees	89.3% 	86.1% 	63.7% 
	chain	88.4% 	86.6% 	83.1% 
Stochastic gradient push	time-varying exponential	87.0% 	68.9% 	62.4% 

Table D.8: Tuned learning rates for table D.7. We tuned the learning rate for each setting on a multiplicative grid with spacing  $\sqrt{2}$ , and then repeated each experiment 3 times. If both repetitions diverged, we would change to a smaller learning rate in the grid. Numbers in parentheses are the ‘effective’ learning rates corrected according to appendix D.4.1.

Algorithm	Topology	16 workers	32 workers	64 workers
All-reduce (baseline)	fully connected	0.1 (0.100)	0.05 (0.050)	0.05 (0.050)
RelaySGD	binary trees	0.282 (0.066)	0.2 (0.035)	0.2 (0.027)
	chain	0.2 (0.047)	0.4 (0.070)	0.8 (0.108)
Stochastic gradient push	time-varying exp.	0.025 (0.025)	0.025 (0.025)	0.0125 (0.013)

topologies as illustrates by fig. D.4.

## D.6 RelaySum for distributed mean estimation

We conceptually separate the optimization algorithm RelaySGD from the communication mechanism RelaySum that uniformly distributes updates across a peer-to-peer network. We made this choice because we envision other applications of the RelaySum mechanism outside of optimization for machine learning. To illustrate this point, this section introduces RelaySum for Distributed Mean Estimation (Algorithm 15).

In distributed mean estimation, workers are connected in a network just as in our optimization setup, but instead of models gradients, they receive samples  $\hat{\mathbf{d}}^{(t)} \sim \mathcal{D}$  of the distribution  $\mathcal{D}$  at time step  $t$ . The workers estimate the mean  $\bar{\mathbf{d}}$  the mean of  $\mathcal{D}$ , and we measure their average squared error to the true mean.

In algorithm 15, the output estimates  $\mathbf{x}_i^{(t)}$  of a worker  $i$  is a uniform average of all samples that can reach a worker  $i$  at that time step. This algorithm enjoys variance reduction of  $\mathcal{O}(\frac{1}{nT})$ , a desirable property that is in general not shared by gossip-averaging-based algorithms on arbitrary graphs.

In fig. D.5, we compare this algorithm to a simple gossip-based baseline.

## D.7 Alternative optimizer based on RelaySum

Apart from RelaySGD presented in the main paper, there are other ways to build optimization algorithms based on the RelaySum communication mechanism. In this section, we describe

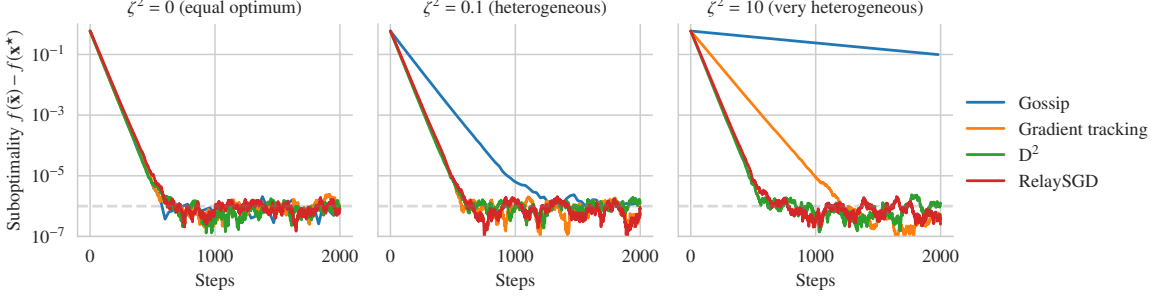


Figure D.3: Random quadratics on *ring* networks of size 32 with varying data heterogeneity  $\zeta^2$  and all other theoretical quantities fixed. To simulate stochastic noise, we add random normal noise to each gradient update. For each method, the learning rate is tuned to reach suboptimality  $\leq 10^{-6}$  the fastest. RelaySGD operates on a chain network instead of a ring. Like  $D^2$ , it does not require more steps when the worker’s objectives are more heterogeneous.

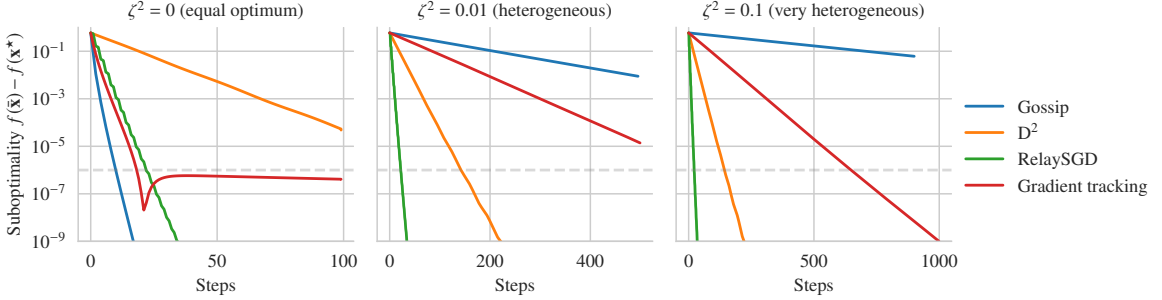


Figure D.4: Random quadratics on *star* networks of size 32 with varying data heterogeneity  $\zeta^2$  and all other theoretical quantities fixed. For each method, the learning rate is tuned to reach suboptimality  $\leq 10^{-6}$  the fastest. Like  $D^2$ , RelaySGD does not require more steps when the worker’s objectives are more heterogeneous. Note that for  $\zeta^2 = 0$  (left figure), our tuning procedure found a learning rate where Gradient Tracking does converge to  $< \leq 10^{-6}$ , but does not converge linearly. It would with a lower learning rate.

RelaySGD/Grad (Algorithm 16), an alternative to RelaySGD that does use the RelaySum mechanism on *gradient updates* rather than on *models*.

RelaySGD/Grad distributes each update uniformly over all workers in a finite number of steps. This means that worker’s models differ by only a finite number of  $\mathcal{O}(\tau_{\max} \max n)$  that are scaled as  $\frac{1}{n}$ . With this property, it achieves tighter consensus than typical gossip averaging, and it also works well in deep learning. Contrary to RelaySGD, however, this algorithm is not fully independent of data heterogeneity, due to the delay in the updates. When the data heterogeneity  $\zeta^2 > 0$ , RelaySGD/Grad does not converge linearly, but its suboptimality saturates at a level that depends on  $\zeta^2$ .

The sections below study this alternative algorithm in detail, both theoretically and experimentally. The key differences between RelaySGD and RelaySGD/Grad are:



**Algorithm 15** RelaySum for Distributed Mean Estimation

---

**Input:**  $\forall i, \mathbf{x}_i^{(0)} = \mathbf{0}, \mathbf{y}_i^{(0)} = \mathbf{0}, s_i^{(0)} = 0; \forall i, j, \mathbf{m}_{i \rightarrow j}^{(-1)} = \mathbf{0}$ , tree network

- 1: **for**  $t = 0, 1, \dots$  **do**
- 2:   **for** node  $i$  **in parallel**
- 3:     **for** each neighbor  $j \in \mathcal{N}_i$  **do**
- 4:       Get a sample  $\hat{\mathbf{d}}_i^{(t)} \sim \mathcal{D}$ .
- 5:       Send  $\mathbf{m}_{i \rightarrow j}^{(t)} = \hat{\mathbf{d}}_i^{(t)} + \sum_{k \in \mathcal{N}_i \setminus j} \mathbf{m}_{k \rightarrow i}^{(t-1)}$ .
- 6:       Send  $c_{i \rightarrow j}^{(t)} = 1 + \sum_{k \in \mathcal{N}_i \setminus j} c_{k \rightarrow i}^{(t-1)}$ .
- 7:       Receive  $\mathbf{m}_{j \rightarrow i}^{(t)}$  and  $c_{j \rightarrow i}^{(t)}$  from node  $j$ .
- 8:     **end for**
- 9:     Update the sum of samples  $\mathbf{y}_i^{(t+1)} = \mathbf{y}_i^{(t)} + \hat{\mathbf{d}}_i^{(t)} + \sum_{j \in \mathcal{N}_i} \mathbf{m}_{j \rightarrow i}^{(t)}$ .
- 10:    Update the sum of counts  $s_i^{(t+1)} = s_i^{(t)} + 1 + \sum_{j \in \mathcal{N}_i} c_{j \rightarrow i}^{(t)}$ .
- 11:    Output average estimate  $\mathbf{x}_i^{(t)} = \mathbf{y}_i^{(t)} / s_i^{(t)}$ .
- 12:   **end for**
- 13: **end for**

---

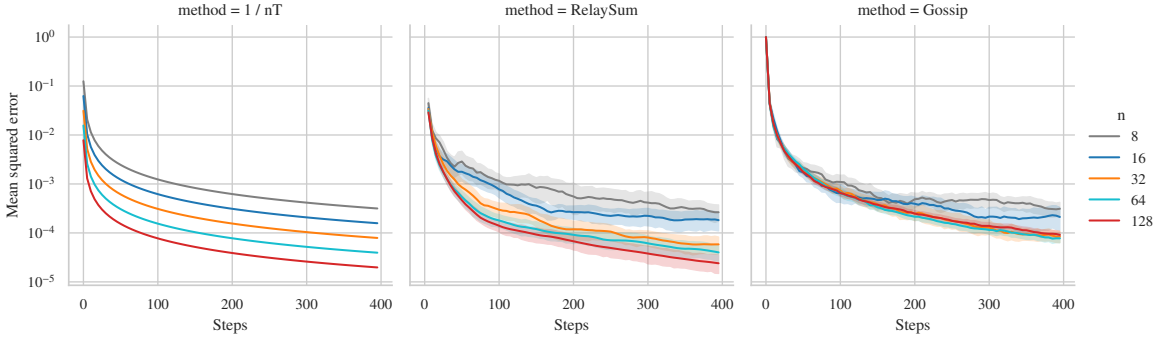


Figure D.5: RelaySum for Distributed Mean Estimation compared to a gossip-based baseline, on a ring topology (chain for RelaySGD). Workers receive samples from a normal distribution  $\mathcal{N}(1, 1)$  with mean 1. RelaySum, using Algorithm 15 achieves a variance reduction of  $\mathcal{O}(\frac{1}{nT})$ .

	RelaySGD	RelaySGD/Grad
Provably independent of data heterogeneity $\zeta^2$	yes	no
Distributes updates exactly uniform in finite steps	no	yes
Loses energy of gradient updates (appendix D.4.1)	yes	no
Works experimentally with momentum / Adam	yes	no
Robust to lost messages + can support workers joining/leaving	yes	no

---

**Algorithm 16** RelaySGD/Grad

---

**Input:**  $\forall i, \mathbf{x}_i^{(0)} = \mathbf{x}^{(0)}; \forall i, j, \mathbf{m}_{i \rightarrow j}^{(-1)} = \mathbf{0}$ , learning rate  $\gamma$ , tree network

- 1: **for**  $t = 0, 1, \dots$  **do**
- 2:   **for** node  $i$  **in parallel**
- 3:      $\mathbf{u}_i^{(t)} = -\gamma \nabla f_i(\mathbf{x}_i^{(t)}, \xi_i^{(t)})$
- 4:     **for** each neighbor  $j \in \mathcal{N}_i$  **do**
- 5:       Send  $\mathbf{m}_{i \rightarrow j}^{(t)} = \mathbf{u}_i^{(t)} + \sum_{k \in \mathcal{N}_i \setminus j} \mathbf{m}_{k \rightarrow i}^{(t-1)}$ .
- 6:       Receive  $\mathbf{m}_{j \rightarrow i}^{(t)}$  from node  $j$ .
- 7:     **end for**
- 8:      $\mathbf{x}_i^{(t+1)} = \mathbf{x}_i^{(t)} + \frac{1}{n} \left( \mathbf{u}_i^{(t)} + \sum_{j \in \mathcal{N}_i} \mathbf{m}_{j \rightarrow i}^{(t)} \right)$
- 9:   **end for**
- 10: **end for**

---

### D.7.1 Empirical analysis of RelaySGD/Grad

In table D.9, we compare RelaySGD/Grad to RelaySGD on deep-learning based image classification on Cifar-10 with VGG-11. Without momentum, and with low levels of heterogeneity, RelaySGD/Grad sometimes outperforms RelaySGD.

Figure D.6 illustrates a key difference between RelaySGD/Grad and RelaySGD. While RelaySGD behaves independently of heterogeneity, and converges linearly with a fixed step size, RelaySGD/Grad reaches a plateau based on the learning rate and level of heterogeneity.

Table D.9: Comparing RelaySGD/Grad with RelaySGD on Cifar-10 [Krizhevsky and Hinton, 2009] with the VGG-11 architecture. We vary the data heterogeneity  $\alpha$  [Lin et al., 2021] between 16 workers. For low-heterogeneity cases and without momentum, RelaySGD/Grad sometimes performs better than RelaySGD.

Algorithm	Topology	$\alpha = 1.00$ (most homogeneous)	$\alpha = 0.1$	$\alpha = .01$ (most heterogeneous)
All-reduce (baseline) +momentum	fully connected	87.0% $\longrightarrow$ $\#$ 90.2% $\longrightarrow$ $\#$	87.0% $\longrightarrow$ $\#$ 90.2% $\longrightarrow$ $\#$	87.0% $\longrightarrow$ $\#$ 90.2% $\longrightarrow$ $\#$
RelaySGD +local momentum	chain	87.3% $\longrightarrow$ $\#$ 89.5% $\longrightarrow$ $\#$	87.2% $\longrightarrow$ $\#$ 89.2% $\longrightarrow$ $\#$	86.5% $\longrightarrow$ $\#$ 88.4% $\longrightarrow$ $\#$
RelaySGD/Grad +local momentum	chain	88.8% $\longrightarrow$ $\#$ 86.9% $\longrightarrow$ $\#$	88.5% $\longrightarrow$ $\#$ 87.8% $\longrightarrow$ $\#$	83.5% $\longrightarrow$ $\#$ 68.6% $\longrightarrow$ $\#$

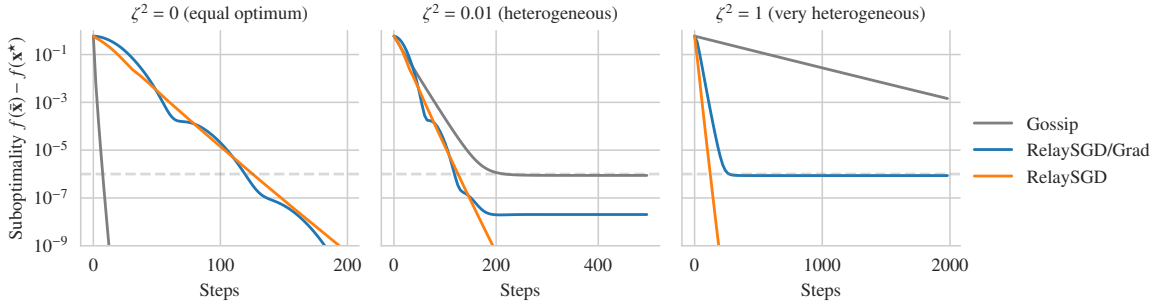


Figure D.6: Comparing RelaySGD/Grad against RelaySGD on random quadratics with varying levels of heterogeneity  $\zeta^2$ , without stochastic noise, on a ring/chain of 32 nodes. Learning rates are tuned to reach suboptimality  $\leq 10^{-6}$  as quickly as possible. In contrast to RelaySGD, RelaySGD/Grad with a fixed learning rate does not converge linearly. Compared to D-SGD (Gossip), RelaySGD/Grad is still less sensitive to data heterogeneity.



# Bibliography

- Alekh Agarwal and John C. Duchi. Distributed delayed stochastic optimization. In *Proc. CDC*, pages 5451–5452, 2012.
- Saurabh Agarwal, Hongyi Wang, Kangwook Lee, Shivaram Venkataraman, and Dimitris S. Papailiopoulos. Adaptive gradient communication via critical learning regime identification. In *Proc. MLSys*, 2021.
- Saurabh Agarwal, Hongyi Wang, Shivaram Venkataraman, and Dimitris S. Papailiopoulos. On the utility of gradient compression in distributed training systems. In *Proc. MLSys*, 2022.
- Mohammadreza Alimohammadi, Ilia Markov, Elias Frantar, and Dan Alistarh. L-greco: An efficient and general framework for layerwise-adaptive gradient compression. *CoRR*, abs/2210.17357, 2022.
- Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. QSGD: communication-efficient SGD via gradient quantization and encoding. In *Proc. NeurIPS*, pages 1709–1720, 2017.
- Peter Arbenz. Lecture notes on solving large scale eigenvalue problems. *D-MATH, ETH Zürich*, 2, 2016.
- Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. In *Proc. ICML*, volume 80, pages 254–263, 2018.
- Mahmoud Assran, Nicolas Loizou, Nicolas Ballas, and Michael G. Rabbat. Stochastic gradient push for distributed deep learning. In *Proc. ICML*, volume 97, pages 344–353, 2019.
- AT&T Laboratories Cambridge. AT&T database of faces. URL [https://scikit-learn.org/0.19/datasets/olivetti\\_faces.html](https://scikit-learn.org/0.19/datasets/olivetti_faces.html).
- Ammar Ahmad Awan, Ching-Hsiang Chu, Hari Subramoni, and Dhabaleswar K. Panda. Optimized broadcast for deep learning workloads on dense-gpu infiniband clusters: MPI or nccl? In *Proc. EuroMPI*, pages 2:1–2:9, 2018.
- Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016.
- Alexei Baevski and Michael Auli. Adaptive input representations for neural language modeling. In *Proc. ICLR*, 2019.

- B. Le Bars, Aurélien Bellet, Marc Tommasi, and Anne-Marie Kermarrec. Yes, topology matters in decentralized optimization: Refined convergence and topology learning under heterogeneous data. *CoRR*, abs/2204.04452, 2022.
- Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. SIGNSGD: compressed optimisation for non-convex problems. In *Proc. ICML*, volume 80, pages 559–568, 2018.
- Jeremy Bernstein, Jiawei Zhao, Kamyar Azizzadenesheli, and Anima Anandkumar. signsgd with majority vote is communication efficient and fault tolerant. In *Proc. ICLR*, 2019.
- Raphaël Berthier, Francis R. Bach, and Pierre Gaillard. Accelerated gossip in networks of given dimension using jacobi polynomial iterations. *SIAM J. Math. Data Sci.*, 2(1):24–47, 2020.
- Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Proc. NeurIPS*, 2020.
- Sebastian Caldas, Jakub Konečný, H. Brendan McMahan, and Ameet Talwalkar. Expanding the reach of federated learning by reducing client resource requirements. *CoRR*, abs/1812.07210, 2018.
- David E. Carlson, Volkan Cevher, and Lawrence Carin. Stochastic spectral descent for restricted boltzmann machines. In *Proc. AISTATS*, volume 38, 2015.
- Themistoklis Charalambous, Ye Yuan, Tao Yang, Wei Pan, Christoforos N. Hadjicostis, and Mikael Johansson. Distributed finite-time average consensus in digraphs in the presence of time delays. *IEEE Trans. Control. Netw. Syst.*, 2(4):370–381, 2015.
- Chia-Yu Chen, Jiamin Ni, Songtao Lu, Xiaodong Cui, Pin-Yu Chen, Xiao Sun, Naigang Wang, Swagath Venkataramani, Vijayalakshmi Srinivasan, Wei Zhang, and Kailash Gopalakrishnan. Scalecom: Scalable sparsified gradient compression for communication-efficient distributed training. In *Proc. NeurIPS*, 2020.
- Minsik Cho, Vinod Muthusamy, Brad Nemanich, and Ruchir Puri. GradZip: Gradient compression using alternating matrix factorization for large-scale deep learning, 2019.
- Edo Collins, Siavash Arjomand Bigdeli, and Sabine Süsstrunk. Detecting memorization in relu networks. *CoRR*, abs/1810.03372, 2018.
- PyTorch Contributors. DDP communication hooks, 2020. URL [https://pytorch.org/docs/stable/ddp\\_comm\\_hooks.html](https://pytorch.org/docs/stable/ddp_comm_hooks.html).
- Jean-Baptiste Cordonnier. Convex optimization using sparsified stochastic gradient descent with memory. Technical report, 2018.

- Yatin Dandi, Anastasia Koloskova, Martin Jaggi, and Sebastian U. Stich. Data-heterogeneity-aware mixing for decentralized learning. *CoRR*, abs/2204.06477, 2022.
- Allison Davis, Burleigh Bradford Gardner, and Mary R Gardner. *Deep South: A social anthropological study of caste and class*. Univ of South Carolina Press, 1930.
- Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc’Aurelio Ranzato, Andrew W. Senior, Paul A. Tucker, Ke Yang, and Andrew Y. Ng. Large scale distributed deep networks. In *Proc. NeurIPS*, pages 1232–1240, 2012.
- Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In *Proc. CVPR*, pages 248–255, 2009.
- Jonathan Frankle, David J. Schwab, and Ari S. Morcos. The early phase of neural network training. In *Proc. ICLR*, 2020.
- Keshi Ge, Yongquan Fu, Yiming Zhang, Zhiquan Lai, Xiaoge Deng, and Dongsheng Li. S2 reducer: High-performance sparse communication to accelerate distributed deep learning. In *Proc. ICASSP*, pages 5233–5237, 2022.
- Leonidas Georgopoulos. *Definitive Consensus for Distributed Data Inference*. PhD thesis, EPFL, Switzerland, 2011. URL <https://doi.org/10.5075/epfl-thesis-5026>.
- Saeed Ghadimi and Guanghui Lan. Accelerated gradient methods for nonconvex nonlinear and stochastic programming. *Mathematical Programming*, 156(1-2):59–99, 2016.
- Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017.
- Suriya Gunasekar, Jason D. Lee, Daniel Soudry, and Nathan Srebro. Characterizing implicit bias in terms of optimization geometry. In *Proc. ICML*, volume 80, pages 1827–1836, 2018.
- Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using NetworkX. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.
- Julien M. Hendrickx, Raphaël M. Jungers, Alexander Olshevsky, and Guillaume Vankeerberghen. Graph diameter, eigenvalues, and minimum-time consensus. *Automatica*, 50(2): 635–640, 2014.
- Samuel Horváth and Peter Richtárik. A better alternative to error feedback for communication-efficient distributed learning. In *Proc. ICLR*, 2021.
- Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, and Kurt Keutzer. Firecaffe: Near-linear acceleration of deep neural network training on compute clusters. In *Proc. CVPR*, pages 2592–2600, 2016.
- Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. ICML*, volume 37, pages 448–456, 2015.

- Nikita Iykin, Daniel Rothchild, Enayat Ullah, Vladimir Braverman, Ion Stoica, and Raman Arora. Communication-efficient distributed SGD with sketching. In *Proc. NeurIPS*, pages 13144–13154, 2019.
- Sylvain Jeaugey. Massively scale your deep learning training with NCCL 2.4. <https://devblogs.nvidia.com/massively-scale-deep-learning-training-nccl-2-4/>, 2019. [Online; accessed 21-May-2019].
- Björn Johansson, Maben Rabi, and Mikael Johansson. A randomized incremental subgradient method for distributed optimization in networked systems. *SIAM J. Optim.*, 20(3):1157–1170, 2009.
- John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Židek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.
- Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista A. Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D’Oliveira, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badi Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaïd Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning. *CoRR*, abs/1912.04977, 2019.
- Sham Kakade, Shai Shalev-Shwartz, Ambuj Tewari, et al. On the duality of strong convexity and strong smoothness: Learning applications and matrix regularization. *Unpublished Manuscript*, 2(1):35, 2009.
- Dhiraj D. Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, Jiyan Yang, Jongsoo Park, Alexander Heinecke, Evangelos Georganas, Sudarshan Srinivasan, Abhisek Kundu, Misha Smelyanskiy, Bharat Kaul, and Pradeep Dubey. A study of BFLOAT16 for deep learning training. *CoRR*, abs/1905.12322, 2019.
- Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U. Stich, and Ananda Theertha Suresh. SCAFFOLD: stochastic controlled averaging for on-device federated learning. *CoRR*, abs/1910.06378, 2019a.
- Sai Praneeth Karimireddy, Quentin Rebjock, Sebastian U. Stich, and Martin Jaggi. Error feedback fixes signsgd and other gradient compression schemes. In *Proc. ICML*, volume 97, pages 3252–3261, 2019b.



- David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *Proc. FOCS*, pages 482–491, 2003.
- Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proc. ICLR*, 2015.
- Chih-Kai Ko and Xiaojie Gao. On matrix factorization and finite-time average-consensus. In *Proc. CDC*, pages 5798–5803, 2009.
- Anastasia Koloskova, Tao Lin, Sebastian U. Stich, and Martin Jaggi. Decentralized deep learning with arbitrary communication compression. *CoRR*, abs/1907.09356, 2019a.
- Anastasia Koloskova, Sebastian U. Stich, and Martin Jaggi. Decentralized stochastic optimization and gossip algorithms with compressed communication. In *Proc. ICML*, volume 97, pages 3478–3487, 2019b.
- Anastasia Koloskova, Nicolas Loizou, Sadra Boreiri, Martin Jaggi, and Sebastian U. Stich. A unified theory of decentralized SGD with changing topology and local updates. In *Proc. ICML*, volume 119, pages 5381–5393, 2020.
- Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *CoRR*, abs/1610.05492, 2016.
- Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.
- Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (Canadian Institute for Advanced Research).
- Yuanzhi Li, Tengyu Ma, and Hongyang Zhang. Algorithmic regularization in over-parameterized matrix sensing and neural networks with quadratic activations. In *Proc. COLT*, volume 75, pages 2–47, 2018.
- Yuanzhi Li, Colin Wei, and Tengyu Ma. Towards explaining the regularization effect of initial large learning rate in training neural networks. In *Proc. NeurIPS*, pages 11669–11680, 2019.
- Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent. In *Proc. NeurIPS*, pages 5330–5340, 2017.
- Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. Asynchronous decentralized parallel stochastic gradient descent. In *Proc. ICML*, volume 80, pages 3049–3058, 2018.
- Tao Lin, Sai Praneeth Karimireddy, Sebastian U. Stich, and Martin Jaggi. Quasi-global momentum: Accelerating decentralized deep learning on heterogeneous data. *CoRR*, abs/2102.04761, 2021.
- Yujun Lin, Song Han, Huizi Mao, Yu Wang, and Bill Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. In *Proc. ICLR*, 2018.
- Hanxiao Liu, Andy Brock, Karen Simonyan, and Quoc Le. Evolving normalization-activation layers. In *Proc. NeurIPS*, 2020.

- Paolo Di Lorenzo and Gesualdo Scutari. Next: In-network nonconvex optimization. *IEEE Transactions on Signal and Information Processing over Networks*, 2(2):120–136, 2016.
- Yucheng Lu and Christopher De Sa. Optimal complexity in decentralized training. In *Proc. ICML*, volume 139, pages 7111–7123, 18–24 Jul 2021.
- Yucheng Lu and Christopher De Sa. Moniqua: Modulo quantized communication in decentralized SGD. *CoRR*, abs/2002.11787, 2020.
- Yucheng Lu and Christopher De Sa. Optimal complexity in decentralized training. In *Proc. ICML*, volume 139, pages 7111–7123, 2021.
- Ilia Markov, Hamidreza Ramezani-Kebrya, and Dan Alistarh. Project CGX: scalable deep learning on commodity gpus. *CoRR*, abs/2111.08617, 2021.
- Charles H. Martin and Michael W. Mahoney. Implicit self-regularization in deep neural networks: Evidence from random matrix theory and implications for learning. *CoRR*, abs/1810.01075, 2018.
- Rahul Mazumder, Trevor Hastie, and Robert Tibshirani. Spectral regularization algorithms for learning large incomplete matrices. *J. Mach. Learn. Res.*, 11:2287–2322, 2010.
- Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proc. ICOAI*, volume 54, pages 1273–1282, 2017.
- Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory F. Diamos, Erich Elsen, David García, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training. In *Proc. ICLR*, 2018.
- Angelia Nedic. Distributed gradient methods for convex machine learning problems in networks: Distributed optimization. *IEEE Signal Process. Mag.*, 37(3):92–101, 2020.
- Angelia Nedic and Alex Olshevsky. Stochastic gradient-push for strongly convex functions on time-varying directed graphs. *IEEE Trans. Autom. Control.*, 61(12):3936–3947, 2016.
- Angelia Nedic and Asuman E. Ozdaglar. Distributed subgradient methods for multi-agent optimization. *IEEE Trans. Automat. Contr.*, 54(1):48–61, 2009.
- Angelia Nedic and Asuman E. Ozdaglar. Convergence rate for consensus with delays. *J. Glob. Optim.*, 47(3):437–456, 2010.
- Angelia Nedic, Alex Olshevsky, and Wei Shi. Achieving geometric convergence for distributed optimization over time-varying graphs. *SIAM J. Optim.*, 27(4):2597–2633, 2017.
- Giovanni Neglia, Chuan Xu, Don Towsley, and Gianmarco Calbi. Decentralized gradient methods: does topology matter? In *Proc. AISTATS*, volume 108, pages 2348–2358, 2020.
- Yurii E. Nesterov. *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>.

- NVIDIA. NVIDIA collective communications library (NCCL). <https://developer.nvidia.com/ncc1>, 2019. [Online; accessed 21-May-2019].
- Erkki Oja. Simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15(3):267–273, 1982.
- Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proc. NAACL-HLT*, pages 48–53, 2019.
- Dhabaleswar K. Panda, Ammar Ahmad Awan, and Hari Subramoni. High performance distributed deep learning: a beginner’s guide. In *Proc. SIGPLAN*, pages 452–454, 2019.
- Dario Pasquini, Mathilde Raynal, and Carmela Troncoso. On the privacy of decentralized machine learning. *CoRR*, abs/2205.08443, 2022.
- Radia J. Perlman. An algorithm for distributed computation of a spanningtree in an extended LAN. In *Proc. SIGCOMM*, pages 44–53, 1985.
- Shi Pu and Angelia Nedic. Distributed stochastic gradient tracking methods. *CoRR*, abs/1805.11454, 2018.
- Shi Pu, Wei Shi, Jinming Xu, and Angelia Nedic. Push-pull gradient methods for distributed optimization in networks. *IEEE Trans. Autom. Control.*, 66(1):1–16, 2021.
- Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimization towards training A trillion parameter models. *CoRR*, abs/1910.02054, 2019.
- Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *Proc. ICML*, volume 139, pages 8821–8831, 2021.
- Dominic Richards and Patrick Rebeschini. Optimal statistical rates for decentralised non-parametric regression with linear speed-up. In *Proc. NeurIPS*, pages 1214–1225, 2019.
- Dominic Richards and Patrick Rebeschini. Graph-dependent implicit regularisation for distributed stochastic subgradient descent. *J. Mach. Learn. Res.*, 21:34:1–34:44, 2020.
- Peter Richtárik, Igor Sokolov, and Ilyas Fatkhullin. EF21: A new, simpler, theoretically better, and practically faster error feedback. In *Proc. NeurIPS*, pages 4384–4396, 2021.
- Herbert Robbins and Sutton Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.
- Peter Sanders, Jochen Speck, and Jesper Larsson Träff. Two-tree algorithms for full bandwidth broadcast, reduction and scan. *Parallel Comput.*, 35(12):581–594, 2009.
- Aliaksei Sandryhaila, Soumya Kar, and José M. F. Moura. Finite-time distributed consensus through graph filters. In *Proc. ICASSP*, pages 1080–1084, 2014.
- Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019.

- Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns. In *Proc. INTERSPEECH*, pages 1058–1062, 2014.
- Christopher J. Shallue, Jaehoon Lee, Joseph M. Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E. Dahl. Measuring the effects of data parallelism on neural network training. *J. Mach. Learn. Res.*, 20:112:1–112:49, 2019.
- Shaohuai Shi, Xianhao Zhou, Shutao Song, Xingyao Wang, Zilin Zhu, Xue Huang, Xinan Jiang, Feihu Zhou, Zhenyu Guo, Liqiang Xie, Rui Lan, Xianbin Ouyang, Yan Zhang, Jieqian Wei, Jing Gong, Weiliang Lin, Ping Gao, Peng Meng, Xiaomin Xu, Chenyang Guo, Bo Yang, Zhibo Chen, Yongjian Wu, and Xiaowen Chu. Towards scalable distributed training of deep learning on public cloud clusters. In *Proc. MLSys*, 2021.
- Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proc. ICLR*, 2015.
- Prabhu Teja Sivaprasad, Florian Mai, Thijs Vogels, Martin Jaggi, and François Fleuret. Optimizer benchmarking needs to account for hyperparameter tuning. In *Proc. ICML*, volume 119, pages 9036–9045, 2020.
- GW Stewart. Simultaneous iteration for computing invariant subspaces of non-Hermitian matrices. *Numerische Mathematik*, 25(2):123–136, 1976.
- GW Stewart and JH Miller. Methods of simultaneous iteration for calculating eigenvectors of matrices. *Topics in Numerical Analysis II*, pages 169–185, 1975.
- Sebastian U. Stich. Local SGD converges fast and communicates little. In *Proc. ICLR*, 2019.
- Sebastian U. Stich and Sai Praneeth Karimireddy. The error-feedback framework: Better rates for SGD with delayed gradients and compressed communication. *CoRR*, abs/1909.05350, 2019.
- Sebastian U. Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. Sparsified SGD with memory. In *Proc. NeurIPS*, pages 4452–4463, 2018.
- Shreyas Sundaram and Christoforos N. Hadjicostis. Finite-time distributed consensus in graphs with time-invariant topologies. In *Proc. ACC*, pages 711–716, 2007.
- Hanlin Tang, Shaoduo Gan, Ce Zhang, Tong Zhang, and Ji Liu. Communication compression for decentralized training. In *Proc. NeurIPS*, pages 7663–7673, 2018a.
- Hanlin Tang, Xiangru Lian, Ming Yan, Ce Zhang, and Ji Liu. D<sup>2</sup>: Decentralized training over decentralized data. In *Proc. ICML*, volume 80, pages 4855–4863, 2018b.
- Hanlin Tang, Xiangru Lian, Shuang Qiu, Lei Yuan, Ce Zhang, Tong Zhang, and Ji Liu. Deep-squeeze: Parallel stochastic gradient descent with double-pass error-compensated compression. *CoRR*, abs/1907.07346, 2019.
- Cécile Trottet, Thijs Vogels, Martin Jaggi, and Mary-Anne Hartley. Modular clinical decision support networks (modn) - updatable, interpretable, and portable predictions for evolving clinical environments. *CoRR*, abs/2211.06637, 2022.

- Konstantinos I. Tsianos and Michael G. Rabbat. Distributed consensus and optimization under communication delays. In *Proc. Allerton*, pages 974–982, 2011.
- Konstantinos I. Tsianos, Sean F. Lawlor, and Michael G. Rabbat. Push-sum distributed dual averaging for convex optimization. In *Proc. CDC*, pages 5453–5458, 2012.
- John N. Tsitsiklis. *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>.
- Thijs Vogels, Sai Praneeth Karimireddy, and Martin Jaggi. Powersgd: Practical low-rank gradient compression for distributed optimization. In *Proc. NeurIPS*, pages 14236–14245, 2019.
- Thijs Vogels, Sai Praneeth Karimireddy, and Martin Jaggi. Practical low-rank communication compression in decentralized deep learning. In *Proc. NeurIPS*, 2020.
- Thijs Vogels, Lie He, Anastasia Koloskova, Sai Praneeth Karimireddy, Tao Lin, Sebastian U. Stich, and Martin Jaggi. Relaysun for decentralized deep learning on heterogeneous data. In *Proc. NeurIPS*, pages 28004–28015, 2021.
- Thijs Vogels, Hadrien Hendrikx, and Martin Jaggi. Beyond spectral gap: the role of topology in decentralized learning. In *Proc. NeurIPS*, 2022.
- Hongyi Wang, Scott Sievert, Shengchao Liu, Zachary Charles, Dimitris S. Papailiopoulos, and Stephen J. Wright. ATOMO: communication-efficient learning via atomic sparsification. In *Proc. NeurIPS*, pages 9872–9883, 2018.
- Hongyi Wang, Saurabh Agarwal, and Dimitris S. Papailiopoulos. Pufferfish: Communication-efficient models at no extra cost. In *Proc. MLSys*, 2021a.
- Jianyu Wang, Anit Kumar Sahu, Zhouyi Yang, Gauri Joshi, and Soumya Kar. MATCHA: speeding up decentralized SGD via matching decomposition sampling. *CoRR*, abs/1905.09435, 2019.
- Yi Wang, Alex Iankoulski, Pritam Damania, and Sundar Ranganathan. Accelerating pytorch ddp by 10x with powersgd, Nov 2021b. URL <https://medium.com/pytorch/accelerating-pytorch-ddp-by-10x-with-powersgd-585aef12881d>.
- Jianqiao Wangni, Jialei Wang, Ji Liu, and Tong Zhang. Gradient sparsification for communication-efficient distributed optimization. In *Proc. NeurIPS*, pages 1306–1316, 2018.
- Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Terngrad: Ternary gradients to reduce communication in distributed deep learning. In *Proc. NeurIPS*, pages 1509–1519, 2017.
- Chenguang Xi and Usman A. Khan. DEXTRA: A fast algorithm for optimization over directed graphs. *IEEE Trans. Automat. Contr.*, 62(10):4980–4993, 2017.
- Chenguang Xi, Van Sy Mai, Ran Xin, Eyad H. Abed, and Usman A. Khan. Linear convergence in optimization over directed graphs with row-stochastic matrices. *IEEE Trans. Autom. Control.*, 63(10):3558–3565, 2018.

- Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.
- Lin Xiao and Stephen P. Boyd. Fast linear iterations for distributed averaging. *Syst. Control. Lett.*, 53(1):65–78, 2004.
- Cong Xie, Shuai Zheng, Oluwasanmi Koyejo, Indranil Gupta, Mu Li, and Haibin Lin. CSER: communication-efficient SGD with error reset. In *Proc. NeurIPS*, 2020.
- Ran Xin and Usman A. Khan. A linear algorithm for optimization over directed graphs with geometric convergence. *IEEE Control. Syst. Lett.*, 2(3):315–320, 2018.
- Ran Xin and Usman A. Khan. Distributed heavy-ball: A generalization and acceleration of first-order methods with gradient tracking. *IEEE Trans. Autom. Control.*, 65(6):2627–2633, 2020.
- Ran Xin, Chenguang Xi, and Usman A. Khan. FROST - fast row-stochastic optimization with uncoordinated step-sizes. *EURASIP J. Adv. Signal Process.*, 2019:1, 2019.
- An Xu and Heng Huang. Detached error feedback for distributed SGD with random sparsification. In *Proc. ICML*, volume 162, pages 24550–24575, 2022.
- Hang Xu, Chen-Yu Ho, Ahmed M Abdelmoniem, Aritra Dutta, El Houcine Bergou, Konstantinos Karatsenidis, Marco Canini, and Panos Kalnis. Compressed communication for distributed deep learning: Survey and quantitative evaluation. Technical report, 2020.
- Hang Xu, Chen-Yu Ho, Ahmed M. Abdelmoniem, Aritra Dutta, El Houcine Bergou, Konstantinos Karatsenidis, Marco Canini, and Panos Kalnis. GRACE: A compressed communication framework for distributed machine learning. In *Proc. ICDCS*, pages 561–572, 2021.
- Bicheng Ying, Kun Yuan, Yiming Chen, Hanbin Hu, Pan Pan, and Wotao Yin. Exponential graph is provably efficient for decentralized deep training. In *Proc. NeurIPS*, pages 13975–13987, 2021.
- Yuichi Yoshida and Takeru Miyato. Spectral norm regularization for improving the generalizability of deep learning. *CoRR*, abs/1705.10941, 2017.
- Da Yu, Huishuai Zhang, Wei Chen, Jian Yin, and Tie-Yan Liu. Large scale private learning via low-rank reparametrization. In *Proc. ICML*, volume 139, pages 12208–12218, 2021.
- Mingchao Yu, Zhifeng Lin, Krishna Narra, Songze Li, Youjie Li, Nam Sung Kim, Alexander G. Schwing, Murali Annamaram, and Salman Avestimehr. Gradiveq: Vector quantization for bandwidth-efficient gradient aggregation in distributed CNN training. In *Proc. NeurIPS*, pages 5129–5139, 2018.
- Kun Yuan, Bicheng Ying, Xiaochuan Zhao, and Ali H. Sayed. Exact diffusion for distributed optimization and learning - part I: algorithm development. *IEEE Trans. Signal Process.*, 67(3):708–723, 2019.
- Kun Yuan, Yiming Chen, Xinmeng Huang, Yingya Zhang, Pan Pan, Yinghui Xu, and Wotao Yin. Decentlam: Decentralized momentum SGD for large-batch deep training. *CoRR*, abs/2104.11981, 2021.

- Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan H. Greenewald, Trong Nghia Hoang, and Yasaman Khazaeni. Bayesian nonparametric federated learning of neural networks. In *Proc. ICML*, volume 97, pages 7252–7261, 2019.
- Alp Yurtsever, Madeleine Udell, Joel A. Tropp, and Volkan Cevher. Sketchy decisions: Convex low-rank matrix optimization with optimal storage. In *Proc. ICOAI*, volume 54, pages 1188–1196, 2017.
- Jiaqi Zhang and Keyou You. Decentralized stochastic gradient tracking for empirical risk minimization. *CoRR*, abs/1909.02712, 2019.
- Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Proc. NeurIPS*, pages 649–657, 2015.
- Xian Yao Zhang, Marco Manzi, Thijs Vogels, Henrik Dahlberg, Markus H. Gross, and Marios Papas. Deep compositional denoising for high-quality monte carlo rendering. *Comput. Graph. Forum*, 40(4):1–13, 2021.
- Zhaorong Zhang, Kan Xie, Qianqian Cai, and Minyue Fu. A bp-like distributed algorithm for weighted average consensus. In *Proc. ASCC*, pages 728–733, 2019.
- Jiawei Zhao. signSGD with majority vote. [github.com/PermiJW/signSGD-with-Majority-Vote](https://github.com/PermiJW/signSGD-with-Majority-Vote), 2019. [Online; accessed 12-May-2019].

# Cirriculum Vitae

## EDUCATION

- 2018 – Current **EPFL – Optimization & Machine Learning, PhD**  
PhD in distributed learning under Prof. Martin Jaggi.
- 2014 – 2016 **ETH Zürich – Computational Science & Engineering Master**  
Courses in numerical methods, statistics, machine learning, and high performance computing. Thesis: “Kernel-predicting convolutional networks for Monte Carlo Rendering” in collaboration with Disney Research, under Prof. Andreas Krause.
- 2011 – 2014 **University College Roosevelt – Liberal Arts & Sciences Bachelor**  
International Honours College of Utrecht University. Major mathematics, computer science and physics. Thesis: “Dynamic Path Planning for a Basic Robot”, under Prof. Henk Meijer, graded A+. GPA 4.0/4.0.

## WORK EXPERIENCE

- Aug. 2022 **Google Research**  
– Dec. 2022 Research intern in federated learning.
- Summer 2020 **Nvidia Research**  
Research intern in machine learning for graphics.
- 2018 – 2022 **Disney Research**  
Consultant, advising on deep learning applications in Monte Carlo path tracing.
- 2017 – 2018 **Disney Research**  
Lab associate. Bringing my Master’s thesis research into production.
- Oct. 2016 **Bloomberg LP**  
– Mar. 2017 Industrial placement. Gained practical experience in writing resilient C++ and JavaScript code in a large organization.
- 2009 – 2018 **Thijs Vogels Internet Applications**  
Owner. Development of websites and web-based software. Responsible for



programming, graphic design, communication with clients and business administration.

Dec. 2014 **ETH Zürich, University College Roosevelt, TNO, HZ University of Applied Sciences**

- Mar. 2015 Research assistant. During my studies, I worked in several different academic labs on (1) cleaning web pages for information retrieval tasks (2) data collection for burglary risk models, (3) search algorithms on knowledge graphs, (4) a statistical measure and web tool to measure vaccine protection and (5) data analysis in a medical study on health related quality of life of very preterm and very low birth weight children.

## **PUBLICATIONS**

2022 **Beyond spectral gap: the role of the topology in decentralized learning**

Vogels, T.\*, Hendrikx, H.\*, Jaggi, M. NeurIPS 2022.

2021 **RelaySum for Decentralized Deep Learning on Heterogeneous Data**

Vogels, T.\*, He, L.\*, Koloskova, A., Lin, T., Karimireddy, S., Stich, S., Jaggi, M. NeurIPS 2021.

2021 **Deep Compositional Denoising for High-quality Monte Carlo Rendering**

Zhang, X., Manzi, M., Vogels, T., Dahlberg, H., Gross, M., Papas, M. EGSR 2021

2020 **Practical Low-Rank Communication Compression in Decentralized Deep Learning**

Vogels, T., Karimireddy, S.P., Jaggi, M. NeurIPS 2020.

2020 **Optimizer Benchmarking Needs to Account for Hyperparameter Tuning**

Sivaprasad, P.T., Mai, F., Vogels, T., Jaggi, M., Fleuret, F. ICML 2020.

2019 **PowerSGD: Practical Low-Rank Gradient Compression for Distributed Optimization**

Vogels, T., Karimireddy, S.P., Jaggi, M. NeurIPS 2019.

2018 **Denoising with Kernel Prediction and Asymmetric Loss Functions**

Vogels, T., Rousselle, F., McWilliams, B., Göthlin, G., Harvill, A., Adler, D., Meyer, M., Novák, J. SIGGRAPH 2018.

2018 **Web2Text: Deep Structured Boilerplate Removal**

Vogels, T., Ganea, O., Eickhoff, C. ECIR 2018.

2017 **Kernel-predicting Convolutional Networks for Monte Carlo Rendering**

Bako, S.\*, Vogels, T.\*, McWilliams, B., Meyer, M., Novák, J., Harvill, A., Sen, P., DeRose, T., Rousselle, F. SIGGRAPH 2017.

## **HONORS & AWARDS**

**2017 EDIC PhD Fellowship**

One year of research funding from the Computer Science department at EPFL.

**2017 Fritz Kutter Award**

Awarded to the best thesis in computer science (diploma, master, doctoral) at a Swiss university (July 2016 – September 2017.)

**2017 Willi Studer Prize**

Awarded to the best student in each ETH Zurich Master's degree programme.

**2014 ETH Excellence Scholarship & Opportunity Award**

Merit-based scholarship of ETH Zürich. Covered study- and living expenses for my master.

**2010 Dutch Mathematical Olympiad: 4th place**

4th place out of 4150 participants.