

Algorithms for Efficient and Robust Distributed Deep Learning

Présentée le 18 juillet 2022

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

Tao LIN

Acceptée sur proposition du jury

Prof. P. Frossard, président du jury
Prof. M. Jaggi, Prof. B. Falsafi, directeurs de thèse
Prof. K. Keutzer, rapporteur
Dr J. Feng, rapporteur
Dr M. Salzmann, rapporteur

Simplicity is the ultimate sophistication.
— Leonardo da Vinci

To my family and friends...

Acknowledgments

I am unimaginably fortunate to have Martin Jaggi as my doctoral advisor and mentor, for all his constant support, encouragement, and constructive feedback, not just in the academic area, but also in the well-being of my personal life. I cannot emphasize enough how much effort Martin has taken to set up a healthy research environment for me to thrive in. This thesis and my delightful Ph.D. journey would not have happened if I had not been able to have Martin on the path. I would also like to thank my co-supervisor, Babak Falsafi. The supervision from Babak is unique and must have far-reaching implications for my career: I learned how to conduct research, communicate with others, and interact with the community.

I am also thankful to my thesis committee members, Kurt Keutzer, Jiashi Feng, Mathieu Salzmann, and Pascal Frossard, for their valuable feedback, suggestions and enthusiasm. Their guidance made this thesis better.

I am especially grateful to all of my incredible collaborators for their helps during my Ph.D. journey: Sebastian, Anastasia, Lingjing, Praneeth, Mengjie, Thijs, Fei, Lie, Chen, Luis, and many others. Thank you for taking time to talk with me, share your thoughts, jointly contribute to some interesting research projects, and teach me a lot about research. Aside from research, I also enjoy the moments with other TMLo members, such as climbing with JB and Scott, and wine tasting with Maksym. I am also grateful to the lab secretary Jennifer, and the rest of the EDIC staff members for their assistance with all of the administrative tasks.

My entire journey over the last few years would not have been as pleasurable if I hadn't had so many friends who shared so many amazing and lovely experiences with me. I thank all of them for staying with me and for being so kind to me. The gratitude first goes to my indoor and outdoor sport partners in swimming, badminton, bouldering, skiing, hiking, via ferrata, and sport climbing. Xingyu, Zhengchao, Ting, Jiande, Sailan, Yuan, Shengzhao, Lu, Xiaohua, Junwen, Shiyu, Yihui, Qin, Linjia, Ziqi, Yingkun, Zhidian, Chan, and many more, thank you! It is inconceivable for me to conceive how I would spend my difficult moments if I did not spend them with you. Besides some friends mentioned above, my special appreciation goes to Fei, Ka Wai, Kaicheng, Shuxuan, Zhaowen, Xiaodong, Yue, Xiaoyu, Yanfei, Yao, Yingzhao and Jiayi, for spending numerous evenings together and sharing happy hours over food and beverages.

Last but not the least, I want to thank my family, especially my parents, for their blessings and love.

Acknowledgments

As I close this part, I must remind the reader that this acknowledgment is far from complete of how many kind people have helped me. Each and every one of them has inspired me, and I hope I can pay forward at least a tiny fraction of that kindness.

Lausanne, June 21, 2022

T. L.



Abstract

The success of deep learning may be attributed in large part to remarkable growth in the size and complexity of deep neural networks. However, present learning systems raise significant efficiency concerns and privacy: (1) currently, training systems are lagging behind the fast growth of deep neural architectures, and the training efficiency of deep learning algorithms cannot be guaranteed; (2) most learning is operated in a centralized way, yet massive amounts of data are created on decentralized edge devices and may contain sensitive information about users. All of these considerations lead to the necessity to migrate to distributed deep learning.

In this thesis, we study *efficiency* and *robustness*, the two fundamental problems that have emerged in distributed deep learning. We first propose strategies to improve communication efficiency—a bottleneck to scaling distributed learning systems out and up—from various aspects: the study starts by understanding the trade-off between communication frequency and generalization performance, and then extends to decentralized and sparse communication topologies with compressed communication. Next, we investigate the computational efficiency issue of deep learning, which is yet another crucial factor that determines the learning and deployment efficiency. The proposed solutions can be generalized to various scenarios. Finally, learning with edge devices introduces various kinds of heterogeneity (e.g. data heterogeneity and system heterogeneity) in practice. As the last key contribution of this thesis, we develop robust decentralized/federated algorithms that are resistant to real-world challenges such as client data distribution shifts and heterogeneous computing systems.

Keywords: deep learning, optimization, generalization, efficiency, robustness, collaborative learning, distributed learning, decentralized learning, federated learning.

Résumé

Le succès de l'apprentissage en profondeur peut être attribué en grande partie à la croissance remarquable de la taille et de la complexité des réseaux de neurones profonds. Cependant, les systèmes d'apprentissage actuels soulèvent d'importants problèmes d'efficacité et de confidentialité : (1) actuellement, les systèmes de formation sont en retard par rapport à la croissance rapide des architectures neuronales profondes, et l'efficacité de la formation des algorithmes d'apprentissage en profondeur ne peut être garantie ; (2) la plupart des apprentissages sont opérés de manière centralisée, mais des quantités massives de données sont créées sur des appareils périphériques décentralisés et peuvent contenir des informations sensibles sur les utilisateurs. Toutes ces considérations conduisent à la nécessité de migrer vers un apprentissage en profondeur distribué.

Dans cette thèse, nous étudions *l'efficacité* et *la robustesse*, les deux problèmes fondamentaux qui ont émergé dans l'apprentissage profond distribué. Nous proposons d'abord des stratégies pour améliorer l'efficacité de la communication—un goulot d'étranglement à la mise à l'échelle des systèmes d'apprentissage distribué—sous divers aspects : l'étude commence par comprendre le compromis entre la fréquence de communication et les performances de généralisation, puis s'étend aux topologies de communication décentralisées et parcimonieuses avec une communication compressée. Ensuite, nous étudions la question de l'efficacité de calcul de l'apprentissage en profondeur, qui est encore un autre facteur crucial qui détermine l'efficacité de l'apprentissage et du déploiement. Les solutions que nous avons proposées peuvent être généralisées à divers scénarios. Enfin, l'apprentissage avec des appareils de périphérie introduit divers types d'hétérogénéité (par exemple, l'hétérogénéité des données et l'hétérogénéité du système) dans la pratique. Comme dernière contribution clé de cette thèse, nous développons des algorithmes décentralisés/fédérés robustes qui résistent aux défis du monde réel tels que les changements de distribution des données client et les systèmes informatiques hétérogènes.

Mots clés : apprentissage profond, optimisation, généralisation, efficacité, robustesse, apprentissage collaboratif, apprentissage distribué, apprentissage décentralisé, apprentissage fédéré.

Contents

Acknowledgements	i
Abstract (English/Français)	iii
1 Introduction	1
1.1 Thesis Outline and Contributions	3
1.1.1 Organization	5
1.2 Publications/Pre-prints Related to the Thesis	6
1.3 Additional Publications/Pre-prints of the Author not Present in This Thesis	7
I Learning Efficiency: Communication Efficiency	9
2 Don't Use Large Mini-Batches, Use Local SGD	11
2.1 Preface	11
2.2 Introduction	12
2.3 Related Work	14
2.4 Post-Local SGD and Hierarchical Local SGD	15
2.5 Experimental Results	16
2.5.1 Superior Scalability of Local SGD over Mini-batch SGD	18
2.5.2 (Post)-Local SGD Closes the Generalization Gap of Large-batch Training	19
2.6 Discussion and Interpretation	22
2.6.1 Post-local SGD Converges to Flatter Minima	23
2.7 Conclusion	24
3 Extrapolation for Large-batch Training in Deep Learning	25
3.1 Preface	25
3.2 Introduction	26
3.3 Related Work	27
3.4 Optimization with Extrapolation	29
3.4.1 Accelerated (Stochastic) Local Extragradient for Distributed Training	29
3.4.2 Unified Extrapolation Framework	31
3.5 Theoretical Analysis of Nesterov Momentum and EXTRAP-SGD	32
3.5.1 Analysis for Mini-batch SGD (with Nesterov Momentum)	32

Contents

3.5.2	Convergence of EXTRAP-SGD	33
3.6	Experiments	34
3.6.1	Experimental Setup	35
3.6.2	Evaluation on Large-batch Training	36
3.6.3	Ablation Study	39
3.7	Conclusion	40
4	Decentralized Deep Learning with Arbitrary Communication Compression	41
4.1	Preface	41
4.2	Introduction	42
4.3	Related Work	43
4.4	CHOCO-SGD	44
4.5	Convergence of CHOCO-SGD on Smooth Non-Convex Problems	46
4.6	Comparison to Baselines for Various Compression Schemes	47
4.7	Use Case I: On-Device Peer-to-Peer Learning	48
4.8	Use Case II: Efficient Large-Scale Training in a Datacenter	52
4.9	Conclusion	53
5	Consensus Control for Decentralized Deep Learning	55
5.1	Preface	55
5.2	Introduction	56
5.3	Related Work	57
5.3.1	Decentralized Learning	57
5.3.2	Critical Learning Phase in Deep Learning	58
5.4	Theoretical Understanding	58
5.4.1	Notation and Setting	59
5.4.2	Decentralized Consensus Optimization	59
5.4.3	Controlling the Consensus Distance	61
5.5	Inspecting Consensus Distance for Decentralized Training	61
5.5.1	Experiment Design: Controlled Training Phases	62
5.5.2	Experimental Setup	63
5.5.3	Findings on Computer Vision Tasks	64
5.5.4	Preliminary Study on Training Transformer Models	67
5.6	Impact on Practice	68
5.7	Conclusion	69
II	Learning Efficiency: Computational Efficiency	71
6	Dynamic Model Pruning with Feedback	73
6.1	Preface	73
6.2	Introduction	74
6.3	Related Work	74

6.4	Method	76
6.5	Convergence Analysis	78
6.6	Experiments	80
6.6.1	Experimental Setup	80
6.6.2	Experiment Results	82
6.7	Discussion	83
7	Masking as an Efficient Alternative to Fine-tuning	87
7.1	Preface	87
7.2	Introduction	88
7.3	Related Work	89
7.4	Method	90
7.4.1	Background on Transformer and Fine-tuning	90
7.4.2	Learning the Mask	90
7.4.3	Configuration of Masking	91
7.5	Datasets and Experimental Setup	92
7.6	Experiments	93
7.6.1	Initial Sparsity of Binary Masks	93
7.6.2	Layer-wise Behaviors	95
7.6.3	Comparing Fine-tuning and Masking	95
7.7	Discussion	97
7.7.1	Intrinsic Evaluations	97
7.7.2	Properties of the Binary Masked Models	99
7.7.3	Loss Landscape	100
7.8	Conclusion	101
III	Robustness to Heterogeneous Environments	103
8	Ensemble Distillation for Robust Model Fusion in Federated Learning	105
8.1	Preface	105
8.2	Introduction	106
8.3	Related Work	107
8.4	Ensemble Distillation for Robust Model Fusion	108
8.5	Experiments	110
8.5.1	Setup	110
8.5.2	Evaluation on the Common Federated Learning Settings	111
8.5.3	Case Studies	115
8.6	Understanding FedDF	116
9	Accelerating Decentralized Deep Learning on Heterogeneous Data	119
9.1	Preface	119
9.2	Introduction	120

Contents

9.3	Related Work	121
9.4	Method	123
9.4.1	Notation and Setting	123
9.4.2	QG-DSGDm Algorithm	123
9.4.3	Convergence Analysis	126
9.4.4	Connection with Other Methods	126
9.5	Understanding QG-DSGDm	127
9.5.1	Faster Convergence in Average Consensus	127
9.5.2	QG-DSGDm (Single Worker Case) Recovers QHM	128
9.6	Experiments	129
9.6.1	Setup	129
9.6.2	Results	131
10	Conclusion and Future Work	137
IV	Appendices	141
11	Appendix for Local SGD Variants	143
11.1	Details on Deep Learning Experimental Setup	143
11.1.1	Dataset	143
11.1.2	Models and Model Initialization	143
11.1.3	Large Batch Learning Schemes	144
11.1.4	Hyperparameter Choices and Training Procedure	145
11.1.5	System Performance Evaluation	147
11.2	Local SGD Training	147
11.2.1	Formal Definition of the Local SGD Algorithm	147
11.2.2	Numerical Illustration of Local SGD on a Convex Problem	147
11.2.3	More Results on Local SGD Training	149
11.2.4	Practical Improvement Potentials for Standard Local SGD Training	150
11.3	Post-local SGD Training	156
11.3.1	The Algorithm of Post-local SGD	156
11.3.2	The Effectiveness of Turning on Post-local SGD after the First Learning Rate Decay	156
11.3.3	The Speedup of Post-local SGD Training on CIFAR	157
11.3.4	Understanding the Generalization of Post-local SGD	157
11.3.5	Post-local SGD Training on Diverse Tasks	161
11.4	Hierarchical Local SGD	166
11.4.1	The Illustration of Hierarchical Local SGD	167
11.4.2	Hierarchical Local SGD Training	168
11.5	Discussion and Future Work	171
12	Appendix for EXTRAP-SGD	173

12.1	Nonconvex Proof for Nesterov Momentum	173
12.1.1	Main Proof of Theorem 3.5.2	175
12.2	Nonconvex Proof for EXTRAP-SGD	177
12.2.1	Main Proof of Theorem 3.5.4	183
12.2.2	Proof of Corollary 3.5.6	188
12.3	Detailed Experimental Setup	189
12.3.1	Hyperparameter Tuning Procedure and the Corresponding Values . . .	190
12.4	Algorithmic Details	191
12.4.1	EXTRAP-SGD with Post-local SGD	191
12.4.2	Generalized EXTRAP-SGD with Adam	191
12.5	Additional Results	191
12.5.1	ResNet-20 on CIFAR-10	191
12.5.2	LSTM on WikiText2	193
12.5.3	Impact of Momentum Factors	193
13 Appendix for CHOCO-SGD		197
13.1	Convergence of CHOCO-SGD	197
13.1.1	A General Framework for Decentralized SGD with Arbitrary Averaging	197
13.1.2	Proofs	198
13.1.3	Convergence for Arbitrary T	201
13.2	Useful Inequalities	203
13.3	Compression Schemes	203
13.4	CHOCO-SGD with Momentum	204
13.5	Error Feedback Interpretation of CHOCO-SGD	205
13.6	Detailed Experimental Setup and Tuned Hyperparameters	205
13.7	Additional Plots	207
14 Appendix for Consensus Control in Decentralized Learning		211
14.1	Efficient Implementation of Consensus Control for Data-Center Training	211
14.1.1	Theoretical Justification	211
14.1.2	Experiments with the Efficient Consensus Control Scheme	214
14.2	Related Work	215
14.2.1	Connection with Prior Work	215
14.2.2	Discussion on “Convergence Analysis v.s. Generalization Performance”	216
14.3	Theory	216
14.3.1	Proof of Proposition 5.4.2, Critical Consensus Distance	216
14.3.2	Proof of Proposition 5.4.3, Typical Consensus Distance	217
14.3.3	Sufficient Bounds to Meet Critical Consensus Distance	218
14.3.4	Proof of Lemma 5.4.4, Repeated Gossip	219
14.4	Detailed Experimental Setup	220
14.5	Additional Results	221
14.5.1	Understanding on Consensus Averaging Problem	221
14.5.2	Understanding the Decentralized Deep Learning Training for CV Tasks	223

Contents

14.5.3	Consensus Control with Other Topologies	226
14.5.4	Results for Training Transformer on Multi30k	226
15	Appendix for DPF	227
15.1	Algorithm	227
15.2	Implementation Details	227
15.3	Additional Results for Unstructured Pruning	229
15.3.1	Complete Results of Unstructured Pruning on CIFAR-10	229
15.3.2	Understanding the Training Dynamics and Lottery Ticket Effect	230
15.3.3	Computational Overhead and the Impact of Hyperparameters	231
15.3.4	Implicit Neural Architecture Search	232
15.4	Additional Results for Structured Pruning	233
15.4.1	Generalization Performance for CIFAR-10	233
15.4.2	Understanding the Lottery Ticket Effect	233
15.4.3	Model Sparsity Visualization	235
15.5	Missing Proofs	236
16	Appendix for Masking as an Efficient Alternative to Fine-tuning	239
16.1	Reproducibility Checklist	239
16.1.1	Computing Infrastructure	239
16.1.2	Number of Parameters	239
16.1.3	Validation Performance	239
16.1.4	Hyperparameter Search	239
16.1.5	Datasets	240
16.2	More on Mode Connectivity	241
16.3	More Empirical Results	242
16.4	Numerical Values of Plots	243
16.4.1	Layer-wise Behaviors	243
16.4.2	Memory Consumption	243
17	Appendix for FedDF	245
17.1	Detailed Related Work Discussion	245
17.2	Algorithmic Description	246
17.3	Additional Experimental Setup and Evaluations	248
17.3.1	Detailed Description for Toy Example (Figure 8.1)	248
17.3.2	Detailed Experiment Setup	248
17.3.3	Some Empirical Understanding of FEDAVG	251
17.3.4	The Advantages of FedDF	252
17.4	Details on Generalization Bounds	258
17.4.1	Proof for Generalization Bounds	259
18	Appendix for Quasi-Global Momentum	261
18.1	Detailed Experimental Setup	261

18.1.1	Visualization for Communication Topologies	261
18.1.2	Visualization of Non-IID Local Data	261
18.2	Detailed Algorithm Description and Connections	262
18.2.1	Detailed Algorithm Description	262
18.2.2	Difference between DMSGD and QG-DSGDm	264
18.2.3	Connections Between SGDm and QG-SGDm	267
18.2.4	Comparison with D^2 and Gradient Tracking (GT) methods	270
18.3	Global Convergence Rate Proofs	271
18.4	Additional Results	282
18.4.1	Results on Distributed Average Consensus Problem	282
18.4.2	Results on 2D Illustration	282
18.4.3	The Learning Curves on CV tasks	283
18.4.4	The Ineffectiveness of Tuning Momentum Factor for DSGDm-N	283
18.4.5	The Superior Performance of QG-DSGDm-N Generalize to Various Topology Scales	284
18.4.6	Multiple-step QG-DSGDm-N variant	284
18.4.7	Understanding QG-DSGDm and QG-DSGDm-N on the Single Worker Case	284
18.4.8	Understanding QG-DSGDm on the Single Worker Case via Toy Function	286
Bibliography		318
Curriculum Vitae		319

1 Introduction

The importance of distributed deep learning. Recent progress in deep learning has shown impressive results in many domains, which also led to a dramatic increase in the size, complexity, and computational power of the training system. For example, the model size of GPT-3 (Brown et al., 2020) by OpenAI was raised to 175B parameters and consumed 3.114×10^{23} FLOPS (floating-point operations) of compute during pre-training—which would cost \$4.6 million in electricity cost alone (at \$1.5 per kilowatt-hour) (gpt, 2020a)—compared to only 1×10^{16} FLOPS/s-days for the smaller GPT-2 model (with 1.5B parameters) that was released one year ago (Radford et al., 2019).

In the meantime, significant efficiency concerns and privacy risks arise as currently most learning is operated in a centralized way, while tremendous amounts of data tend to be generated on decentralized edge devices and may contain users' personal information: for example, one report from International Data Corporation (idc, 2021) predicts the potential to have 27.1 billion worldwide IoT connections in 2025. Given that edge devices create over 70% of data (idc, 2021), bringing the learning closer to edge devices with preserved data locality and privacy naturally becomes a key research direction.

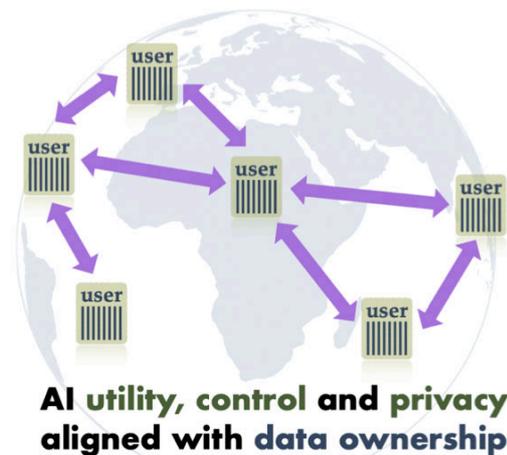
All these motivate the necessity of moving to distributed deep learning, for the aspects of (1) learning efficiency (in data centers and on edge devices), and (2) privacy-preserving learning on edge devices.

Distributed deep learning in data centers. In response to the question of learning efficiency for ML models with millions or even billions of parameters, distributed (but centrally controlled) training algorithms have been developed for use in data centers. Existing approaches, on the other hand, are still inefficient and fall short of social needs such as sustainability, accessibility, and AI resource availability: for example, GPT-3 requires training in a data center (gpt, 2020b) with more than 285,000 CPU cores, 10,000 V100 GPUs and 400 gigabits per second of network connectivity for each GPU server. Aside from the challenges of deploying a cost-effective training platform from a computer system perspective, the lack of theoretical & empirical understanding

Introduction

of deep learning makes it difficult to design efficient learning algorithms that can trade off achievable accuracy and training scalability. Diving into the theoretical foundation of optimization and generalization for deep learning is a timely research path to enable constructing practical large-scale distributed learning algorithms in data centers.

Decentralized and federated deep learning. Decentralized machine learning methods—allowing training in a peer-to-peer communication network (without a central coordinator and without data sharing)—have emerged as a key paradigm in large-scale machine learning (Lian et al., 2017, 2018; Koloskova et al., 2019, 2020b). A weaker variant of decentralized learning—known as federated learning (Kairouz et al., 2021)—performs collaborative learning with the support of a central coordinator, while maintaining data locality on each user device. Decentralized versions of current workhorse Stochastic Gradient Descent (SGD) methods for deep learning training offer (1) scalability to large datasets and systems in large data centers (Lian et al., 2017; Assran et al., 2019; Koloskova et al., 2020a), as well as (2) emerging distributed privacy-preserving learning on edge devices (a.k.a. on-device learning and federated learning) (Kairouz et al., 2021; Koloskova et al., 2020a), where the training data remains distributed over a large number of clients (e.g. mobile phones, robots, or hospitals) and is kept locally (never transmitted during training).



However, most learning algorithms (1) do currently not have their utility, control, and privacy linked with the data ownership of the users, and (2) are lacking robustness to dynamic learning environments especially for heterogeneous edge devices. All of these challenges cause a degradation in performance and applicability, and thus make such learning systems falling behind the societal demands for carbon neutrality, as well as new regulatory measures such as GDPR (General Data Protection Regulation) in Europe, or China’s recently enacted Personal Information Protection Law (PIPL). We aim to seize the opportunity to build decentralized analogs of current machine learning algorithms, which are at the same time (i) efficient, and (ii) robust to the heterogeneous learning environment.

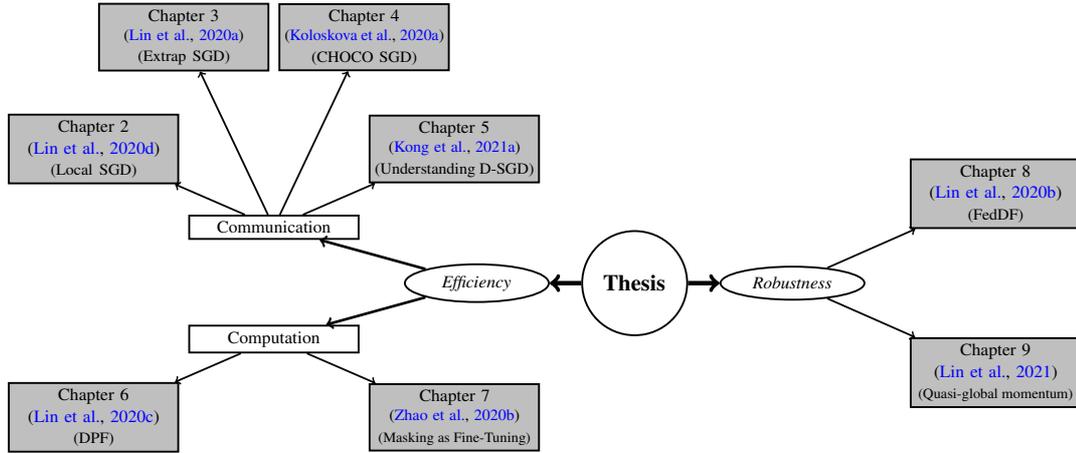


Figure 1.1 – Thesis outline.

1.1 Thesis Outline and Contributions

The journey of this thesis starts from the exploration to improve “(1) learning efficiency: communication efficiency” (Lin et al., 2020d,a; Koloskova et al., 2020a; Kong et al., 2021a), and then extends to the critical “(2) learning efficiency: computational efficiency” (Lin et al., 2020c; Zhao et al., 2020b). In addition to efficiency, the thesis investigates “(3) robustness to heterogeneous environments” (Lin et al., 2020b, 2021) to guard the realization of collaborative learning in the wild. Consequently, this thesis contributes some key algorithmic building blocks to the research community for **efficient** and **robust** distributed deep learning (outlined in Figure 1.1), allowing:

1. diverse clients to perform machine learning without transmitting their data (Lin et al., 2020d,b, 2021; Koloskova et al., 2020a) in a decentralized/federated fashion,
2. with **efficiency** in resource-constrained environments, either data centers (Lin et al., 2020d,c,a; Koloskova et al., 2020a; Kong et al., 2021a), or edge devices (Lin et al., 2020c,b; Koloskova et al., 2020a; Zhao et al., 2020b),
3. with **robustness** to distribution shifts (Lin et al., 2021) and allowing different neural network architectures (Lin et al., 2020b).

Learning Efficiency: Communication Efficiency. The tremendous amount of model parameters and the number of synchronization steps are the current bottlenecks for distributed training: for example, the energy cost of training/deploying contemporary ML models such as those used for natural language applications has climbed tremendously in the last years, reaching millions of dollars in electricity cost for a single training run. These issues further deteriorate for large and heterogeneous computing systems like edge computing: when scaling up the number of worker devices, the parallelism per device remains unchanged, while the communication efficiency can decrease dramatically. Improving communication efficiency for models while maintaining the generalization performance of learning is a non-trivial but crucial task in distributed deep

learning.

- **Local SGD reduces the total communication rounds and improves generalization performance.** Our local SGD algorithms in Chapter 2 (Lin et al., 2020d), pioneering work in the field, suggested performing more local update steps before synchronization and therefore gave the flexibility to adapt to the communication-computation trade-offs that would maximize parallel execution.

We also conducted an extensive study (Lin et al., 2020d) on the communication efficiency v.s. performance trade-offs associated with local SGD for training in data centers and provided a new variant that significantly improves the generalization performance compared to large-batch training and converges to flatter minima. We further pushed the limit of existing large-batch training—for maximum scalability—by proposing a novel distributed smoothing method for sharpness-aware neural network training that is both communication and compute efficient (see Chapter 3 (Lin et al., 2020a)).

- **Decentralized learning lowers the communication cost.** Along with the above-mentioned technique of reducing the communication frequency, decentralized learning—instead of communicating on a fully-connected graph topology—is another way to alleviate the communication overhead by allowing a peer-to-peer fashion and is independent of the network topology sizes. We went one step further and proposed CHOCO-SGD in Chapter 4, the first practical decentralized learning algorithm (Koloskova et al., 2020a) under arbitrary communication compression to achieve convergence, both in a data center and social network setting.

Despite the intriguing property of constant-level communication cost in decentralized learning compared to the current most common All-Reduce communication protocol (logarithmic complexity w.r.t. network topology sizes), decentralized learning has failed to gain widespread adoption in the deep learning community, even in data centers. This is mainly due to the degraded learning performance: the training and test performance of models trained in a decentralized fashion are in general worse than that of models trained in a centralized fashion, even with homogeneous data. As a crucial and fundamental task, we comprehended the trade-off between efficiency and optimization/generalization quality in Kong et al. (2021a) (Chapter 5), where the identified key parameter (i.e. consensus distance between devices) explains the quality gap and the associated empirical insights allow a principled design of better decentralized learning algorithms.

Learning Efficiency: Computational Efficiency. In addition to the communication efficient techniques described above, computationally efficient learning and deployment are timely and demanding for both data-centers and edge devices. Model compression is a critical component to save energy and satisfy device capacity limits on mobile user devices. This is motivated by recent findings of compressibility of a model before or during training, allowing for the exclusion of at least 90% of model parameters from training. Our work in Chapter 6 (Lin et al., 2020c) on dynamic model pruning during training enables sparse models to be trained effectively and directly in a single training pass with no additional overhead compared to dense models.

Taking the ingredient of model compression (Lin et al., 2020c), we further proposed to employ efficient binary masks in Chapter 7, as a technique of performing fine-tuning for memory-efficient multi-task inference on edge devices (Zhao et al., 2020b), applicable to recent transformer models.

Robustness to Heterogeneous Environments. The prevalence of heterogeneity across diverse edge devices poses open challenges for both decentralized and federated learning. To mature the practicability of decentralized/federated, a few key building blocks are provided below to equip algorithms with robustness to the dynamic and heterogeneous learning agents in the wild.

- **Robustness to Heterogeneous Data.** In realistic scenarios, clients participating in distributed training often have very different data distributions. Such heterogeneous data distributions generate very distinct optimization objectives on each client, which results in sluggish and unstable global convergence, as well as poor generalization performance. Our work in Chapter 8 (Lin et al., 2020b) took one inspiring step for federated learning: we propose an effective on-server model aggregation to allow knowledge aggregation and training across heterogeneous data.

We also provide deeper insights into the key question of heterogeneous data for the more general case of decentralized learning. Our quasi-global momentum (Lin et al., 2021), as a first practical deep learning algorithm in the decentralized setting (introduced in Chapter 9), mitigates the optimization difficulties, and the resultant generalization performance outperforms all other available methods while remaining both communication and computation efficient.

- **Robustness: Enabling Collaborative Training with Heterogeneous Devices.** Edge devices, such as mobile devices and robotics, are exposed to a variety of hardware constraints, e.g. processing power and RAM capacity. Learning efficiency is harmed by such variability, and the asynchrony induced in collaborative training degrades training performance. To facilitate the ease of deployment of these decentralized/federated learning and inference systems, we need to empower our algorithms with multiple robustness blocks to these issues. Our work on ensemble distillation (Lin et al., 2020b), as in Chapter 8, is the first of its kind: our algorithm allows edge devices to learn with a variety of neural network architectures that are tailored to their hardware, and enables efficient and effective knowledge transfer between architectures. This solution addressed the issue that all prior federated learning algorithms assumed the same architecture across all edge devices, and provided a new direction to save energy while allowing efficient and reliable distributed collaborative learning.

1.1.1 Organization

The aforementioned contributions are described in details in the next eight chapters. Each of these chapters maps to a published paper written by the author (as the main contributor). Chapters start with a preface, consisting of a summary of the work and list of author contributions using the CRediT framework (Brand et al., 2015). The appendices for all the chapters are collected at the end in the fourth part.

1.2 Publications/Pre-prints Related to the Thesis

- Thesis Chapter 2 was published in ([Lin et al., 2020d](#))
Don't Use Large Mini-batches, Use Local SGD.
Tao Lin, Sebastian U. Stich, Kumar Kshitij Patel, Martin Jaggi.
In the Proceedings of International Conference on Learning Representations (ICLR) 2020.
- Thesis Chapter 3 was published in ([Lin et al., 2020a](#))
Extrapolation for Large-batch Training in Deep Learning.
Tao Lin^{*}, Lingjing Kong^{*}, Sebastian U. Stich, Martin Jaggi.
In the Proceedings of International Conference on Machine Learning (ICML) 2020.
- Thesis Chapter 4 was published in ([Koloskova et al., 2020a](#))
Decentralized Deep Learning with Arbitrary Communication Compression.
Anastasia Koloskova^{*}, Tao Lin^{*}, Sebastian U. Stich, Martin Jaggi.
In the Proceedings of International Conference on Learning Representations (ICLR) 2020.
- Thesis Chapter 5 was published in ([Kong et al., 2021a](#))
Consensus Control for Decentralized Deep Learning.
Lingjing Kong^{*}, Tao Lin^{*}, Anastasia Koloskova, Martin Jaggi, Sebastian U. Stich.
In the Proceedings of International Conference on Machine Learning (ICML) 2021.
- Thesis Chapter 6 was published in ([Lin et al., 2020c](#))
Dynamic Model Pruning with Feedback.
Tao Lin, Sebastian U. Stich, Luis Barba, Daniil Dmitriev, Martin Jaggi.
In the Proceedings of International Conference on Learning Representations (ICLR) 2020.
- Thesis Chapter 7 was published in ([Zhao et al., 2020b](#))
Masking as an Efficient Alternative to Finetuning for Pretrained Language Models.
Mengjie Zhao^{*}, Tao Lin^{*}, Fei Mi, Martin Jaggi, Hinrich Schütze.
In the Proceedings of Empirical Methods in Natural Language Processing (EMNLP) 2020.
- Thesis Chapter 8 was published in ([Lin et al., 2020b](#))
Ensemble Distillation for Robust Model Fusion in Federated Learning.
Tao Lin^{*}, Lingjing Kong^{*}, Sebastian U. Stich, Martin Jaggi.
In the Proceedings of Advances in Neural Information Processing Systems (NeurIPS) 2020.
- Thesis Chapter 9 was published in ([Lin et al., 2021](#))
Quasi-Global Momentum: Accelerating Decentralized Deep Learning on Heterogeneous Data.
Tao Lin, Sai Praneeth Karimireddy, Sebastian U. Stich, Martin Jaggi.
In the Proceedings of International Conference on Machine Learning (ICML) 2021.

1.3 Additional Publications/Pre-prints of the Author not Present in This Thesis

Apart from the research presented in rest of the thesis, the author also took part in other research projects, on the topics of, (1) efficient and/or robust (distributed) deep learning algorithms, (2) understanding behaviors in deep learning training, and (3) deep learning applications. The related publications/pre-prints of the author (but not present in this thesis) are given below.

Efficient and/or robust (distributed) deep learning algorithms.

- *RelaySum for Decentralized Deep Learning on Heterogeneous Data.*
Thijs Vogels*, Lie He*, Anastasia Koloskova, *Tao Lin*, Sai Praneeth Karimireddy, Sebastian U. Stich, Martin Jaggi. In Proceedings of Advances in Neural Information Processing Systems (NeurIPS) 2021 ([Vogels et al., 2021](#)).
- *An Improved Analysis of Gradient Tracking for Decentralized Machine Learning.*
Anastasia Koloskova, *Tao Lin*, Sebastian U. Stich. In Proceedings of Advances in Neural Information Processing Systems (NeurIPS) 2021 ([Koloskova et al., 2021](#)).
- *Generalized Class Incremental Learning.*
Fei Mi*, Lingjing Kong*, *Tao Lin*, Kaicheng Yu, Boi Faltings. In Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR) 2020 (workshop track) ([Mi et al., 2020](#)).
- *Training DNNs with Hybrid Block Floating Point.*
Mario Drumond, *Tao Lin*, Martin Jaggi, Babak Falsafi. In Proceedings of Advances in Neural Information Processing Systems (NeurIPS) 2018 ([Drumond et al., 2018](#)).
- *Test-Time Robust Personalization for Federated Learning.*
Liangze Jiang*, *Tao Lin**. Arxiv pre-print ([Jiang and Lin, 2022](#)).
- *Towards Federated Learning on Time-Evolving Heterogeneous Data.*
Yongxin Guo*, *Tao Lin**, Xiaoying Tang. Arxiv pre-print ([Guo et al., 2021](#)).
- *Communication-Efficient Gradient Tracking for Decentralized Deep Learning.*
Yue Liu, *Tao Lin*, Anastasia Koloskova, Sebastian Stich. Technical report ([Liu et al., 2021b](#)).
- *Slimmable Training for Heterogeneous Federated Learning Systems.*
Lingjing Kong*, *Tao Lin**, Sebastian U. Stich, Martin Jaggi. Technical report ([Kong et al., 2021b](#)).
- *Representation Memorization for Fast Learning New Knowledge without Forgetting.*
Fei Mi, *Tao Lin*, Boi Faltings. Arxiv pre-print ([Mi et al., 2021](#)).

Understanding behaviors in deep learning training.

- *Understanding Memorization from the Perspective of Optimization via Efficient Influence Estimation.*
Futong Liu, *Tao Lin*, Martin Jaggi. NeurIPS 2021 (OPT workshop track) ([Liu et al., 2021a](#)).

Introduction

- *On the Loss Landscape of Adversarial Training: Identifying Challenges and How to Overcome Them.*
Chen Liu, Mathieu Salzmann, *Tao Lin*, Ryota Tomioka, Sabine Süsstrunk. In Proceedings of Advances in Neural Information Processing Systems (NeurIPS) 2020 ([Liu et al., 2020a](#)).

Deep learning applications.

- *Learning Disentangled Behaviour Patterns for Wearable-based Human Activity Recognition.*
Jie Su, Zhenyu Wen, *Tao Lin*, Yu Guan. In Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT) 2022 ([Su et al., 2022](#)).
- *Exploring Interpretable LSTM Neural Networks over Multi-Variable Data.*
Tian Guo, *Tao Lin*, Nino Antulov-Fantulin. In Proceedings of International Conference on Machine Learning (ICML) 2019 ([Guo et al., 2019](#)).

Learning Efficiency: Communication Efficiency **Part I**

2 Don't Use Large Mini-Batches, Use Local SGD

2.1 Preface

Contribution and sources. This chapter reproduces [Lin et al. \(2020d\)](#). Martin Jaggi initially pointed out the direction of understanding the empirical performance of local SGD, and the author greatly extended the concept to the case of large mini-batches. The implementation, deep learning experiment design, evaluation of experiments, empirical deep learning insights, and theoretical interpretation, were all conducted by the author. Sebastian U. Stich contributed to the related work section on the convergence theory of local SGD and provided some experimental results for the toy convex case. Kumar Kshitij Patel was involved in the discussion of the theoretical interpolation of post-local SGD.

Summary. Mini-batch stochastic gradient methods (SGD) are state of the art for distributed training of deep neural networks. Drastic increases in the mini-batch sizes have led to key efficiency and scalability gains in recent years. However, progress faces a major roadblock, as models trained with large batches often do not generalize well, i.e. they do not show good accuracy on new data.

As a remedy, we propose a *post-local* SGD and show that it significantly improves the generalization performance compared to large-batch training on standard benchmarks while enjoying the same efficiency (time-to-accuracy) and scalability. We further provide an extensive study of the communication efficiency v.s. performance trade-offs associated with a host of *local SGD* variants.

2.2 Introduction

Fast and efficient training of large scale deep-learning models relies on distributed hardware and on distributed optimization algorithms. For efficient use of system resources, these algorithms crucially must (i) enable parallelization while being communication efficient, and (ii) exhibit good generalization behavior, i.e. good performance on unseen data (test-set). Most machine learning applications currently depend on stochastic gradient descent (SGD) (Robbins and Monro, 1951) and in particular its mini-batch variant (Bottou, 2010; Dekel et al., 2012). However, this algorithm faces generalization difficulties in the regime of very large batch sizes as we will review now.

Mini-batch SGD. For a sum-structured optimization problem of the form $\min_{\mathbf{x} \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^N f_i(\mathbf{x})$ where $\mathbf{x} \in \mathbb{R}^d$ denotes the parameters of the model and $f_i: \mathbb{R}^d \rightarrow \mathbb{R}$ the loss function of the i -th training example, the mini-batch SGD update for $K \geq 1$ workers is given as

$$\mathbf{x}_{(t+1)} := \mathbf{x}_{(t)} - \eta_{(t)} \left[\frac{1}{K} \sum_{k=1}^K \frac{1}{B} \sum_{i \in \mathcal{J}_{(t)}^k} \nabla f_i(\mathbf{x}_{(t)}) \right], \quad (2.1)$$

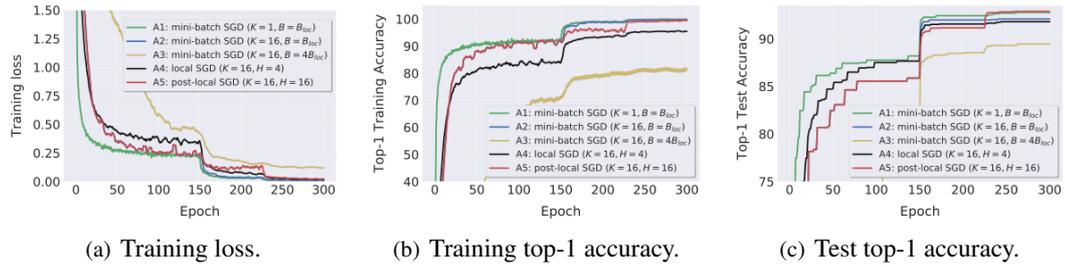
where $\eta_{(t)} > 0$ denotes the learning rate and $\mathcal{J}_{(t)}^k \subseteq [N]$ the subset (mini-batch) of training datapoints selected by worker k (typically selected uniformly at random from the locally available datapoints on worker k). For convenience, we will assume the same batch size B per worker.

Local SGD. Motivated to better balance the available system resources (computation v.s. communication), local SGD (a.k.a. local-update SGD, parallel SGD, or federated averaging) has recently attracted increased research interest (McDonald et al., 2009; Zinkevich et al., 2010; McDonald et al., 2010; Zhang et al., 2014, 2016; McMahan et al., 2017). In local SGD, each worker $k \in [K]$ evolves a local model by performing H sequential SGD updates with mini-batch size B_{loc} , before communication (synchronization by averaging) among the workers. Formally,

$$\mathbf{x}_{(t+h+1)}^k := \mathbf{x}_{(t+h)}^k - \eta_{(t)} \left[\frac{1}{B_{\text{loc}}} \sum_{i \in \mathcal{J}_{(t+h)}^k} \nabla f_i(\mathbf{x}_{(t+h)}^k) \right], \quad \mathbf{x}_{(t+1)}^k := \frac{1}{K} \sum_{k=1}^K \mathbf{x}_{(t+H)}^k, \quad (2.2)$$

where $\mathbf{x}_{(t+h)}^k$ denotes the local model on machine k after t global synchronization rounds and subsequent $h \in [H]$ local steps ($\mathcal{J}_{(t+h)}^k$ is defined analogously). Mini-batch SGD is a special case if $H = 1$ and $B_{\text{loc}} = B$. Furthermore, the communication patterns of both algorithms are identical if $B = HB_{\text{loc}}$. However, as the updates (eq. (2.2)) are different from the mini-batch updates (eq. (2.1)) for any $H > 1$, the generalization behavior (test error) of both algorithms is expected to be different.

Large batch SGD. Recent schemes for scaling training to a large number of workers rely on standard mini-batch SGD (2.1) with very large overall batch sizes (Shallue et al., 2019; You et al., 2018; Goyal et al., 2017), i.e. increasing the global batch size linearly with the number of workers K . However, the batch size interacts differently with the overall system efficiency on



Algorithm	accuracy on training set		accuracy on test set		system performance		
	loss value	top-1 acc.	top-1 acc.		parallelism	communication	
A1: small mini-batch SGD ($K=1, B=B_{\text{loc}}$)	0.01	100% <i>excellent</i>	93% <i>excellent</i>		$\times 1$	-	<i>poor</i>
A2: large mini-batch SGD ($K=16, B=B_{\text{loc}}$)	0.01	100% <i>excellent</i>	92% <i>good</i>		$\times 16$	$\div 1$	<i>ok</i>
A3: huge mini-batch SGD ($K=16, B=4B_{\text{loc}}$)	0.10	81% <i>poor</i>	89% <i>poor</i>		$\times 16$	$\div 4$	<i>good</i>
A4: local SGD ($K=16, H=4$)	0.01	95% <i>ok</i>	92% <i>good</i>		$\times 16$	$\div 4$	<i>good</i>
A5: post-local SGD ($K=16, H=16$)	0.01	99% <i>excellent</i>	93% <i>excellent</i>		$\times 16$	$\div 1$ (phase 1), $\div 16$ (phase 2)	<i>good</i>

Figure 2.1 – Illustration of the **generalization gap**. Large-batch SGD (A2, blue) matches the training curves of small-batch SGD (A1, green), i.e. has no optimization difficulty (left & middle). However, it does not reach the same test accuracy (right) while the proposed post-local SGD (A5, red) does. Post-local SGD (A5) is defined by starting local SGD from the model obtained by large-batch SGD (A2) at epoch 150. Mini-batch SGD with larger mini-batch size (A3, yellow) even suffers from optimization issues. Experiments are for ResNet-20 on CIFAR-10 ($B_{\text{loc}} = 128$), with fine-tuned learning rate for mini-batch SGD with the warmup scheme in Goyal et al. (2017). The inline table highlights the comparison of system/generalization performance for various algorithms.

one hand, and with generalization performance on the other hand. While a larger batch size in general increases throughput, it may negatively affect the final accuracy on both the train- and test-set. Two main scenarios of particular interest can be decoupled as follows:

Scenario 1. The communication restricted setting, where the synchronization time is much higher than the gradient computation time. In this case the batch size of best efficiency is typically large, and is achieved by using partial computation (gradient accumulation) while waiting for communication. We in particular study the interesting case when the mini-batch sizes of both algorithms satisfy the relation $B = HB_{\text{loc}}$, as in this case both algorithms (local SGD and mini-batch SGD) evaluate the same number of stochastic gradients between synchronization steps.

Scenario 2. The regime of poor generalization of large-batch SGD, that is the use of very large overall batches (often a significant fraction of the training set size), which is known to cause drastically decreased generalization performance (Chen and Huo, 2016; Keskar et al., 2017; Hoffer et al., 2017; Shallue et al., 2019; Golmant et al., 2018). If sticking to standard mini-batch SGD and maintaining the level of parallelization, the batch size B would have to be reduced below the device (locally optimal) capacity in order to alleviate this generalization issue, which however impacts training time.¹

¹Note that in terms of efficiency on current GPUs, the computation time on device for small batch sizes is not constant but scales non-linearly with B , as shown in Table 11.2 in Appendix 11.1.

Main Results. Key aspects of the empirical performance of local SGD compared to mini-batch baselines are illustrated in Figure 2.1. In scenario 1), comparing local SGD with $H=4$ (A4) with mini-batch SGD of same effective batch size $B=4B_{\text{loc}}$ (A3) reveals a stark difference, both in terms of train and test error (local SGD achieves lower training loss and higher test accuracy). This motivates the use of *local SGD as an alternative to large-batch training*—a hypothesis that we confirm in our experiments. Further, in scenario 2), mini-batch SGD with smaller batch size $B=B_{\text{loc}}$ (A2) is observed to suffer from poor generalization, although the training curve matches the single-machine baseline (A1). *The generalization gap can thus not be explained as an optimization issue alone.* Our proposed *post-local SGD* (A5) (defined by starting local SGD from the model obtained by large-batch SGD (A2) at epoch 150) closes this generalization gap with the single-machine baseline (A1) and is also more communication efficient than the mini-batch competitors. In direct comparison, post-local SGD is more communication-efficient than mini-batch SGD (while less than local SGD). It achieves better generalization performance than both these algorithms.

Contributions. Our main contributions can thus be summarized as follows:

- **Trade-offs in Local SGD:** We provide the first comprehensive empirically study of the trade-offs in local SGD for deep learning—when varying the number of workers K , number of local steps H and mini-batch sizes—for both scenarios 1) on communication efficiency and 2) on generalization.
- **Post-local SGD:** We propose *post-local SGD*, a simple but very efficient training scheme to address the current generalization issue of large-batch training. It allows us to scale the training to much higher number of parallel devices. Large batches trained by post-local SGD enjoy improved communication efficiency, while at the same time strongly outperforming most competing small and large batch baselines in terms of accuracy. Our empirical experiments on standard benchmarks show that post-local SGD can reach flatter minima than large-batch SGD on those problems.

2.3 Related Work

The generalization gap in large-batch training. State-of-the-art distributed deep learning frameworks (Abadi et al., 2016; Paszke et al., 2017; Seide and Agarwal, 2016) resort to synchronized large-batch SGD training, allowing scaling by adding more computational units and performing data-parallel synchronous SGD with mini-batches divided between devices. Training with large batch size (e.g. batch size $> 10^3$ on ImageNet) typically degrades the performance both in terms of training and test error (often denoted as *generalization gap*) (Chen and Huo, 2016; Li, 2017; Li et al., 2014; Keskar et al., 2017; Shallue et al., 2019; McCandlish et al., 2018; Golmant et al., 2018; Masters and Lusch, 2018). Goyal et al. (2017) argue that the test error degrades because of optimization issues and propose to use a “learning rate warm-up” phase with linear scaling of the step-size. You et al. (2017b) propose Layer-wise Adaptive Rate Scaling (LARS)

2.4. Post-Local SGD and Hierarchical Local SGD

to scale to larger mini-batch size, but the generalization gap does not vanish. [Hoffer et al. \(2017\)](#) argue that the generalization gap can be closed when increasing the number of iterations along with the batch size. However, this diminishes the efficiency gains of parallel training.

[Keskar et al. \(2017\)](#) empirically show that larger batch sizes correlate with sharper minima ([Hochreiter and Schmidhuber, 1997a](#)) found by SGD and that flat minima are preferred for better generalization. This interpretation—despite being debated in [Dinh et al. \(2017\)](#)—was further developed in [Hoffer et al. \(2017\)](#); [Yao et al. \(2018\)](#); [Izmailov et al. \(2018\)](#). [Neelakantan et al. \(2015\)](#) propose to add isotropic white noise to the gradients to avoid over-fitting and better optimization. [Zhu et al. \(2019\)](#); [Xing et al. \(2018\)](#) further analyze “structured” anisotropic noise and highlight the importance of anisotropic noise (over isotropic noise) for improving generalization. The importance of the scale of the noise for non-convex optimization has also been studied in [Smith and Le \(2018\)](#); [Chaudhari and Soatto \(2018\)](#). [Wen et al. \(2019\)](#) propose to inject noise (sampled from the expensive empirical Fisher matrix) to large-batch SGD. However, to our best knowledge, none of the prior work (except our post-local SGD) can provide a computation efficient way to inject noise to achieve as good generalization performance as small-batch SGD, for both of CIFAR and ImageNet experiments.

Local SGD and convergence theory. While mini-batch SGD is very well studied ([Zinkevich et al., 2010](#); [Dekel et al., 2012](#); [Takác et al., 2013](#)), the theoretical foundations of local SGD variants are still developing. [Jain et al. \(2018\)](#) study one-shot averaging on quadratic functions and [Bijral et al. \(2016\)](#) study local SGD in the setting of a general graph of workers. A main research question is whether local SGD provides a linear speedup with respect to the number of workers K , similar to mini-batch SGD. Recent work partially confirms this, under the assumption that H is not too large compared to the total iterations T . [Stich \(2019a\)](#) and [Dieuleveut and Patel \(2019\)](#) show convergence at rate $\mathcal{O}((KTB_{\text{loc}})^{-1})$ on strongly convex and smooth objective functions when $H = \mathcal{O}(T^{1/2})$. For smooth non-convex objective functions, [Zhou and Cong \(2018\)](#) show a rate of $\mathcal{O}((KTB_{\text{loc}})^{-1/2})$ (for the decrement of the stochastic gradient), [Yu et al. \(2019b\)](#) give an improved result $\mathcal{O}((HKT B_{\text{loc}})^{-1/2})$ when $H = \mathcal{O}(T^{1/4})$. For a discussion of more recent theoretical results we refer to ([Kairouz et al., 2021](#)). [Alistarh et al. \(2018\)](#) study convergence under adversarial delays. [Zhang et al. \(2016\)](#) empirically study the effect of the averaging frequency on the quality of the solution for some problem cases and observe that more frequent averaging at the beginning of the optimization can help. Similarly, [Bijral et al. \(2016\)](#) argue to average more frequently at the beginning.

2.4 Post-Local SGD and Hierarchical Local SGD

In this section we present two novel variants of local SGD. First, we propose post-local SGD to reach high generalization accuracy (cf. Section 2.5.2 below), and second, hierarchical SGD designed from a systems perspective aiming at optimal resource adaptivity (computation v.s. communication trade-off).

Post-local SGD: Large-batch Training Alternative for Better Generalization. We propose post-local SGD, a variant where local SGD is only started in the second phase of training, after t' initial steps² with standard mini-batch SGD. Formally, the update in (2.2) is performed with an iteration dependent $H_{(t)}$ given as

$$H_{(t)} = \begin{cases} 1, & \text{if } t \leq t', & (\text{mini-batch SGD}) \\ H, & \text{if } t > t'. & (\text{local SGD}) \end{cases} \quad (\text{post-local SGD})$$

As the proposed scheme is identical to mini-batch SGD in the first phase (with local batch size $B = B_{\text{loc}}$), we can leverage previously tuned learning rate warm-up strategies and schedules for large-batch training (Goyal et al., 2017) without additional tuning. Note that we only use ‘small’ local mini-batches of size $B = B_{\text{loc}}$ in the warm-up phase, and switch to the communication efficient larger effective batches HB_{loc} in the second phase, while also achieving better generalization in our experiments (cf. also the discussion in Section 2.6). We would like to point out that it is crucial to use local SGD in the second phase, as e.g. just resorting to large batch training does achieve worse performance (see e.g. 3rd row in Table 2.1 below).

Hierarchical Local SGD: Optimal Use of Systems Resources in Heterogeneous Systems.

Real world systems come with diverse communication bandwidths on several levels, e.g. with GPUs or other accelerators grouped hierarchically within a chip, machine, rack or even at the level of entire data-centers. In this scenario, we propose to employ local SGD as an inner loop on each level of the hierarchy, adapted to the corresponding computation vs communication trade-off of that particular level. The resulting scheme, *hierarchical local SGD*, can offer significant benefits in terms of system adaptivity and performance, as we show with experiments and a discussion in Appendix 11.4.

2.5 Experimental Results

In this section we systematically evaluate the aforementioned variants of local SGD on deep learning tasks. In summary, the main findings presented in Sections 2.5.1 and 2.5.2 below are:

Local SGD, on the one hand, *can serve as a communication-efficient alternative to mini-batch SGD for various practical purposes (communication restricted Scenario 1)*. For example in Figure 2.3(a) (with fixed B_{loc} and K), in terms of 92.48% best test accuracy achieved by our mini-batch SGD implementation for $K=16$, practitioners can either choose to achieve reasonable good training quality (91.2%, matching He et al. (2016a)) with a $2.59\times$ speedup (time-to-accuracy) in training time ($H=8$), or achieve slight better test performance (92.57%) with slightly reduced communication efficiency ($1.76\times$ speedup for $H=2$).

² The switching time t' between the two phases in our experiments is determined by the first learning rate decay. However it could be tuned more generally aiming at capturing the time when trajectory starts to get into the influence basin of a local minimum (Robbins and Monro, 1951; Smith et al., 2018; Loshchilov and Hutter, 2017; Huang et al., 2017a). Results in Appendix 11.2.4 and 11.3.2 empirically evaluate the impact of various t' s on optimization and generalization.

Post-local SGD, on the other hand, in addition to overcoming communication restrictions *does provide a state-of-the-art remedy for the generalization issue of large-batch training (Scenario 2)*. Unlike local SGD with large H and K (e.g. $H=16, K=16$) which can encounter optimization issues during initial training (which can impact later generalization performance), we found that post-local SGD elegantly enables an ideal trade-off between optimization and generalization within a fixed training budget (e.g. Table 2.2, Table 2.4, Figure 2.4 and many others in Appendix 11.3.5), and can achieve the same or even better performance than small mini-batch baselines.

Setup. We briefly outline the general experimental setup, and refer to Appendix 11.1 for full details.

Datasets. We evaluate all methods on the following two main (standard) tasks: (1) Image classification for CIFAR-10/100 (Krizhevsky and Hinton, 2009), and (2) Image classification for ImageNet (Russakovsky et al., 2015). The detailed data augmentation scheme refers to Appendix 11.1.

Models. We use ResNet-20 (He et al., 2016a) with CIFAR-10 as a base configuration to understand the properties of (post-)local SGD. We then empirically evaluate the large-batch training performance of post-local SGD, for ResNet-20, Densenet-40-12 (Huang et al., 2017b) and WideResNet-28-10 (Zagoruyko and Komodakis, 2016) on CIFAR-10/100. Finally, we train ResNet-50 (He et al., 2016a) on ImageNet to investigate the accuracy and scalability of (post-)local SGD training.

Implementation and platform. Our algorithms are implemented³ in PyTorch (Paszke et al., 2017), with a flexible configuration of the machine topology supported by Kubernetes. The cluster consists of Intel Xeon E5-2680 v3 servers and each server has 2 NVIDIA TITAN Xp GPUs. We use the notion $a \times b$ -GPU to denote the topology of the cluster, i.e. a nodes and each with b GPUs.

Specific learning schemes for large-batch SGD. We rely on the recently proposed schemes for efficient large batch training (Goyal et al., 2017), which are formalized by (i) linearly scaling the learning rate w.r.t. the global mini-batch size; (ii) gradual warm-up of the learning rate from a small value. See Appendix 11.1.3 for more details.

Distributed training procedure on CIFAR-10/100. The experiments follow the common mini-batch SGD training scheme for CIFAR (He et al., 2016a,b; Huang et al., 2017b) and all competing methods access the same total number of data samples (i.e. gradients) regardless of the number of local steps. Training ends when the distributed algorithms have accessed the same number of samples as the single-worker baseline. The data is disjointly partitioned and reshuffled globally every epoch. The learning rate scheme follows (He et al., 2016a; Huang et al., 2017b), where

³ Our code is available at <https://github.com/epfml/LocalSGD-Code>.

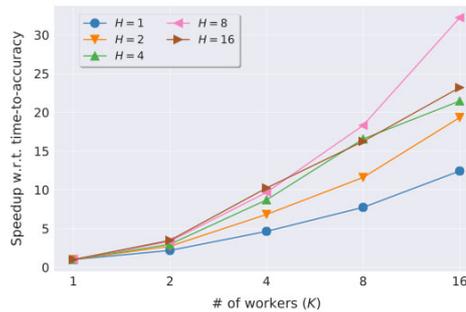
we drop the initial learning rate by a factor of 10 when the model has accessed 50% and 75% of the total number of training samples. Unless mentioned specifically, the used learning rate is scaled by the global mini-batch size (BK for mini-batch SGD and $B_{\text{loc}}K$ for local SGD) where the initial learning rate is fine-tuned for each model and each task for the single worker. See Appendix 11.1.4 for more details.

2.5.1 Superior Scalability of Local SGD over Mini-batch SGD

First, we empirically study local SGD training for the communication restricted case (i.e. Scenario 1). As local SGD (with local batch size B_{loc}) needs H times fewer communication rounds than mini-batch SGD (with the same batch size $B = B_{\text{loc}}$) we expect local SGD to significantly outperform mini-batch SGD in terms of time-to-accuracy and scalability and verify this experimentally. We further observe that local SGD shows better generalization performance than mini-batch SGD.

Significantly better scalability when increasing the number of workers on CIFAR, in terms of time-to-accuracy. Figure 2.2 demonstrates the speedup in time-to-accuracy for training ResNet-20 for CIFAR-10, with varying number of GPUs K and the number of local steps H , both from 1 to 16.

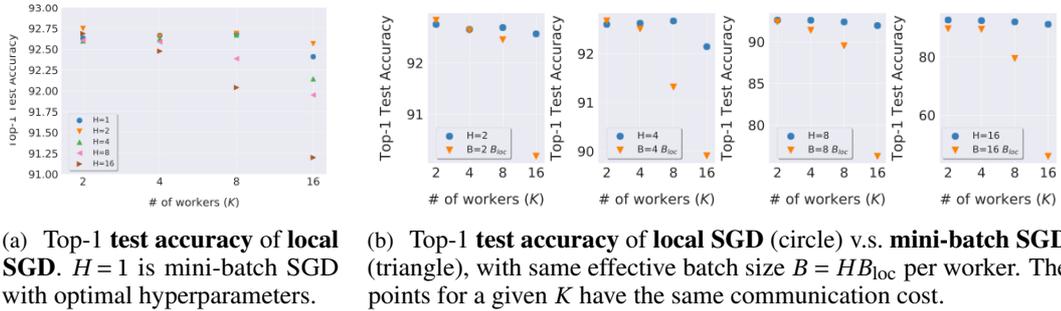
Figure 2.2 – Scaling behavior of **local SGD** in clock-time for increasing number of workers K , for a varied number of local steps H , for training **ResNet-20** on **CIFAR-10** with $B_{\text{loc}} = 128$. The reported **speedup** (averaged over three runs) is over single GPU training time for reaching the baseline top-1 test accuracy (91.2% as in He et al. (2016a)). We use a 8×2 -GPU cluster with 10 Gbps network. $H=1$ recovers mini-batch SGD.



We demonstrate in Figure 2.2 that *local SGD scales 2x better than its mini-batch SGD counterpart, in terms of time-to-accuracy* as we increase the number of workers K on a commodity cluster. The local update steps (H) result in a strong advantage over the standard large-batch training. Mini-batch SGD fixes the batch size to $B = B_{\text{loc}}$, and while increasing the number of workers K gets impacted by the communication bottleneck (section 2.2), even as parallelism per device remains unchanged. In this experiment, local SGD on 8 GPUs even achieves a 2x lower time-to-accuracy than mini-batch SGD with 16 GPUs. Moreover, the (near) linear scaling performance for $H=8$ in Figure 2.2, shows that the main hyperparameter H of local SGD is robust and consistently different from its mini-batch counterpart, when scaling the number of workers.

Effectiveness and scalability of local SGD to even larger datasets (e.g. ImageNet) and larger clusters. Local SGD presents a competitive alternative to the current large-batch ImageNet

2.5. Experimental Results



(a) Top-1 test accuracy of local SGD. $H=1$ is mini-batch SGD with optimal hyperparameters. (b) Top-1 test accuracy of local SGD (circle) v.s. mini-batch SGD (triangle), with same effective batch size $B = HB_{\text{loc}}$ per worker. The points for a given K have the same communication cost.

Figure 2.3 – Training ResNet-20 on CIFAR-10 under various choices of K and H , with fixed $B_{\text{loc}} = 128$. All results are averaged over three runs and all settings access to the same total number of training samples. We fine-tune the learning rate of mini-batch SGD for each setting.

Table 2.1 – Test performance (generalization) for local SGD variants and mini-batch SGD (highlighting selected data from Figure 2.3, Table 2.2 and Figure 2.4). Mini-batch SGD suffers from the generalization gap (and sometimes even from optimization issues due to insufficient training) when scaling to larger H and K . Same experimental setup as in Figure 2.3 (fine-tuned mini-batch SGD baselines). The ‘→’ denotes the transition at the first learning rate decay. The reported results are averaged over three runs.

Algorithm	Top-1 acc.	Effect. batch size	Algorithm	Top-1 acc.	Effect. batch size
Mini-batch SGD ($K=4$)	92.6%	$KB=512$	Mini-batch SGD ($K=16$)	92.5%	$KB=2048$
Mini-batch SGD ($K=4$)	89.5%	$KB=8192$	Mini-batch SGD ($K=16$)	76.3%	$KB=16384$
Mini-batch SGD ($K=4$)	92.5%	$KB=512 \rightarrow 8192$	Mini-batch SGD ($K=16$)	92.0%	$KHB_{\text{loc}}=2048 \rightarrow 16384$
Local SGD ($H=16, K=4$)	92.5%	$KHB_{\text{loc}}=8192$	Local SGD ($H=8, K=16$)	92.0%	$KHB_{\text{loc}}=16384$
Post-local SGD ($H=16, K=4$)	92.7%	$KHB_{\text{loc}}=512 \rightarrow 8192$	Post-local SGD ($H=8, K=16$)	92.9%	$KHB_{\text{loc}}=2048 \rightarrow 16384$

training methods. Figure 11.4 in Appendix 11.2.3 shows that we can efficiently train state-of-the-art ResNet-50 (at least $1.5\times$ speedup to reach 75% top-1 accuracy) for ImageNet (Goyal et al., 2017; You et al., 2017b) via local SGD on a 16×2 -GPU cluster.

Local SGD significantly outperforms mini-batch SGD at the same effective batch size and communication ratio. Figure 2.3 compares local SGD with mini-batch SGD of the same effective batch size, that is the same number of gradient computations per communication round as described in Scenario 1 above (Section 2.2).

2.5.2 (Post)-Local SGD Closes the Generalization Gap of Large-batch Training

Even though local SGD has demonstrated effective communication efficiency with guaranteed test performance (Scenario 1), we observed in the previous section that it still encounters difficulties when scaling to very large mini-batches (Figure 2.3(a)), though it is much less affected than mini-batch SGD (Figure 2.3(b)). Post-local SGD can address the generalization issue of large batch training (Scenario 2) as we show now. First, we would like to highlight some key findings in Table 2.1.

Chapter 2. Don't Use Large Mini-Batches, Use Local SGD

Table 2.2 – Top-1 **test accuracy** of training various CNN models via **post-local SGD** on $K = 16$ GPUs with a large global batch size ($KB_{\text{loc}} = 2048$). The reported results are the average of three runs and all settings access to the same total number of training samples. We compare to small and large mini-batch baselines. The \star indicates a fine-tuned learning rate, where the tuning procedure refers to Appendix 11.1.4.

	CIFAR-10				CIFAR-100			
	Small batch baseline \star	Large batch baseline \star	Post-local SGD (H=16)	Post-local SGD (H=32)	Small batch baseline \star	Large batch baseline \star	Post-local SGD (H=16)	Post-local SGD (H=32)
	$K=2, B=128$	$K=16, B=128$	$K=16, B_{\text{loc}}=128$	$K=16, B_{\text{loc}}=128$	$K=2, B=128$	$K=16, B=128$	$K=16, B_{\text{loc}}=128$	$K=16, B_{\text{loc}}=128$
ResNet-20	92.63 \pm 0.26	92.48 \pm 0.17	92.80 \pm 0.16	93.02 \pm 0.24	68.84 \pm 0.06	68.17 \pm 0.18	69.24 \pm 0.26	69.38 \pm 0.20
DenseNet-40-12	94.41 \pm 0.14	94.36 \pm 0.20	94.43 \pm 0.12	94.58 \pm 0.11	74.85 \pm 0.14	74.08 \pm 0.46	74.45 \pm 0.30	75.03 \pm 0.05
WideResNet-28-10	95.89 \pm 0.10	95.43 \pm 0.37	95.94 \pm 0.06	95.76 \pm 0.25	79.78 \pm 0.16	79.31 \pm 0.23	80.28 \pm 0.13	80.65 \pm 0.16

We now focus on the generalization issues (Section 2.2) of large-batch SGD, isolating the potential negative impact from the optimization difficulty (e.g. the insufficient training epochs (Shallue et al., 2019)) from the performance on the test set. Our evaluation starts from a (standard) constant effective mini-batch size of 2048 or 4096 for mini-batch SGD⁴ on CIFAR dataset, as in (Keskar et al., 2017; Hoffer et al., 2017; Yao et al., 2018).

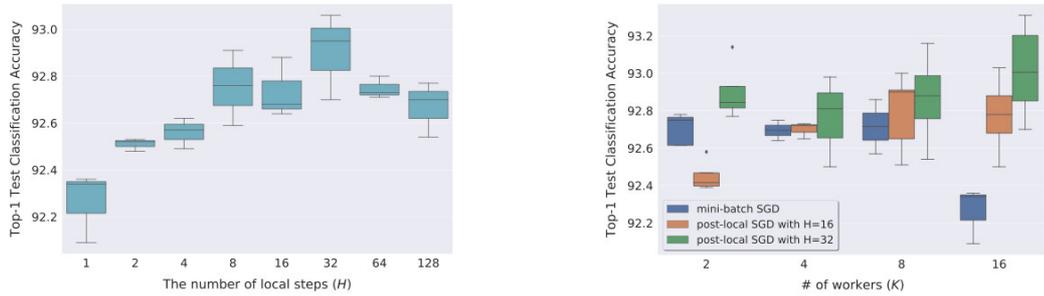
Post-local SGD generalizes better and faster than mini-batch SGD. Table 2.2 summarizes the generalization performance of post-local SGD on large batch size ($K=16, B_{\text{loc}}=128$) across various architectures on CIFAR tasks for $H=16$ and $H=32$. Under the same setup, Table 11.5 in the Appendix 11.3.3 evaluates the speedup of training, while Figure 2.1 demonstrates the learning curves of mini-batch SGD and post-local SGD, highlighting the generalization difficulty of large-batch SGD. We can witness that post-local SGD achieves at least $1.3\times$ speedup over the whole training procedure compared to the mini-batch SGD counterpart ($K=16, B=128$), while enjoying the significantly improved generalization performance.

We further demonstrate the generalization performance and the scalability of post-local SGD, for diverse tasks (e.g. Language Modeling), and for even larger global batch sizes ($KB_{\text{loc}}=4096$ for CIFAR-100 and 4096 and 8192 respectively for ImageNet), in Appendix 11.3.5. For example, Table 11.6 presents the severe generalization issue (2% drop) of the fine-tuned large-batch SGD training ($KB=4096$) for above three CNNs on CIFAR-100, and cannot be addressed by increasing the training steps (Table 11.7). Post-local SGD ($KB_{\text{loc}}=4096$) with default hyperparameters can perfectly close this generalization gap or even better than the fine-tuned small mini-batch baselines.⁵ For ImageNet training in Figure 11.12, the post-local SGD outperforms mini-batch SGD baseline for both of $KB=4096$ (76.18 and 75.87 respectively) and $KB=8192$ (75.65 and 75.64 respectively) with $1.35\times$ speedup for the post-local SGD training phase.

⁴ It is less challenging (e.g. comparing the first and third rows in the right column of Table 2.1) to train with mini-batch SGD for medium-size, constant effective mini-batch size. In our evaluation, the training of the post-local SGD and mini-batch SGD will start from the same effective mini-batch size with the same number of workers K . Even under this relaxed comparison, post-local SGD still significantly outperform mini-batch SGD.

⁵ We omit the comparison with other noise injection methods as none of them (Neelakantan et al., 2015; Wen et al., 2019) can completely address the generalization issue even for CIFAR dataset.

2.5. Experimental Results



(a) Performance of post-local SGD for various numbers of local steps H on 16 GPUs. $H=1$ is mini-batch SGD and $KB_{\text{loc}}=2048$.

(b) Performance of post-local SGD for various numbers of workers K for $H=16$ and $H=32$. The local batch size is fixed to $B_{\text{loc}}=B=128$ both for local and mini-batch SGD).

Figure 2.4 – Top-1 **test accuracy** of training **ResNet-20** on **CIFAR-10**. Box-plot figures are from 3 runs.

Table 2.3 – Top-1 **test accuracy** of training **ResNet-20** on **CIFAR** via sign-based compression scheme and **post-local SGD** ($KB_{\text{loc}}=2048$ and $K=16$). $H=1$ in the table for the simplicity corresponds to the original sign-based algorithm and we fine-tune their hyperparameters. The reported results are the average of three runs. The learning rate schedule is fixed for various local update steps (for $H>1$). For the detailed tuning procedure, training scheme, as well as the pseudo code of the used variants, we refer to Appendix 11.3.5.

	CIFAR-10				CIFAR-100			
	$H=1$	$H=16$	$H=32$	$H=64$	$H=1$	$H=16$	$H=32$	$H=64$
signSGD (Bernstein et al., 2018)	91.61 \pm 0.28	91.77 \pm 0.04	92.22 \pm 0.11	92.00 \pm 0.15	67.21 \pm 0.11	67.43 \pm 0.36	68.32 \pm 0.18	68.12 \pm 0.11
EF-signSGD (Karimireddy et al., 2019)	92.45 \pm 0.28	92.63 \pm 0.16	92.72 \pm 0.05	92.76 \pm 0.06	68.19 \pm 0.10	69.14 \pm 0.36	69.00 \pm 0.14	69.49 \pm 0.21

The effectiveness of post-local SGD training for various numbers of H and K . As seen in Figure 2.4(a), applying any number of local steps over the case of large-batch training (when $KB_{\text{loc}}=2048$) improves the generalization performance compared to mini-batch SGD. Figure 2.4(b) illustrates that post-local SGD is better than mini-batch SGD in general for various numbers of workers K (as well as KB_{loc}). Thus, post-local SGD presents consistently excellent generalization performance.

Post-local SGD can improve the training efficiency upon other compression techniques.

The results in Table 2.3 illustrate that post-local SGD can be combined with other communication-efficient techniques (e.g. the sign-based compression schemes) for further improved communication efficiency and better test generalization performance.

Post-local SGD can improve upon other SOTA optimizers. The benefits of post-local SGD training on ImageNet are even more pronounced for larger batches (e.g. $KB_{\text{loc}}>8192$) (Goyal et al., 2017; Shallue et al., 2019; Golmant et al., 2018). Considering other optimizers, Table 2.4 below shows that post-local SGD can improve upon LARS⁶ (You et al., 2017b).

⁶ The LARS implementation follows the code github.com/NVIDIA/apex for mixed precision and distributed training in Pytorch. LARS uses layer-wise learning rate based on the ratio between $\|\mathbf{x}\|_2$ and $\|\nabla f_i(\mathbf{x})\|_2$, thus our post-local SGD can be simply integrated without extra modification and parameter synchronization.

Chapter 2. Don't Use Large Mini-Batches, Use Local SGD

Table 2.4 – The Top-1 test accuracy of training **ResNet50** on **ImageNet**. We report the performance (w/ or w/o post-local SGD) achieved by using SGD with Nesterov Momentum, learning schemes in (Goyal et al., 2017), and LARS. We follow the general experimental setup described in Section 11.1.4, and use $K=32$ and $H=4$.

	SGD + Momentum + LARS	SGD + Momentum + LARS + Post-local SGD
$KB_{\text{loc}}=8,192$	75.99	76.13
$KB_{\text{loc}}=16,384$	74.52	75.23

2.6 Discussion and Interpretation

Generalization issues of large-batch SGD training are not yet very well understood from a theoretical perspective. Hence, a profound theoretical study of the generalization local SGD is beyond the scope of this work but we hope the favorable experimental results will trigger future research in this direction. In this section we discuss some related work and we argue that local SGD can be seen as a way to inject and control stochastic noise to the whole training procedure.

Connecting Local Updates with Stochastic Noise Injection. The update eq. (2.1) can alternatively be written as

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \hat{\mathbf{g}}_t = \mathbf{x}_t - \eta \mathbf{g}_t + \eta (\mathbf{g}_t - \hat{\mathbf{g}}_t) = \mathbf{x}_t - \eta \mathbf{g}_t + \eta \boldsymbol{\epsilon}, \quad (2.3)$$

where $\hat{\mathbf{g}}_t := \frac{1}{B} \sum_i \nabla f_i(\mathbf{x}_t)$, $\mathbf{g}_t := \mathbb{E} \hat{\mathbf{g}}_t = \nabla f(\mathbf{x}_t)$ and $\boldsymbol{\epsilon} := \mathbf{g}_t - \hat{\mathbf{g}}_t$. For most data-sets the noise $\boldsymbol{\epsilon}$ can be well approximated by a zero-mean Gaussian with variance $\boldsymbol{\Sigma}(\mathbf{x})$. The variance matrix $\boldsymbol{\Sigma}(\mathbf{x})$, following Hoffer et al. (2017) and Hu et al. (2019), for uniform sampling of mini-batch indices can be approximated as

$$\boldsymbol{\Sigma}(\mathbf{x}) \approx \left(\frac{1}{B} - \frac{1}{N}\right) \mathbf{K}(\mathbf{x}) = \left(\frac{1}{B} - \frac{1}{N}\right) \left(\frac{1}{N} \sum_{i=1}^N \nabla f_i(\mathbf{x}) \nabla f_i(\mathbf{x})^\top\right). \quad (2.4)$$

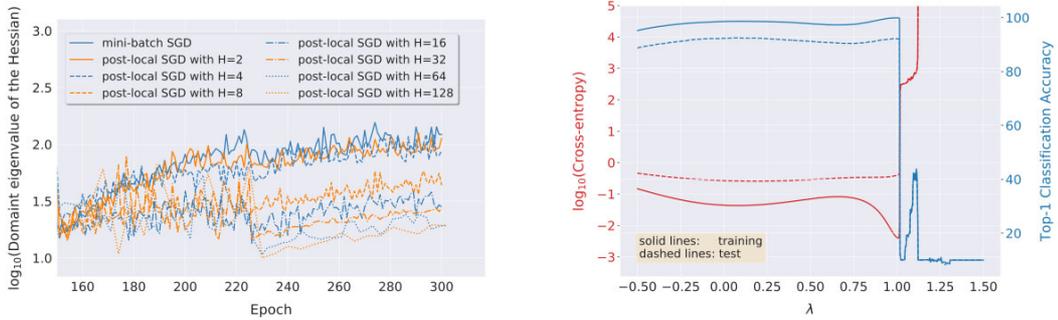
Recent works (Jastrzębski et al., 2018; Li et al., 2017b; Mandt et al., 2017; Zhu et al., 2019) interpret eq. (2.3) as an Euler-Maruyama approximation to the continuous-time Stochastic Differential Equation (SDE): $d\mathbf{x}_t = -\mathbf{g}_t dt + \sqrt{\eta \frac{N-B}{BN}} \mathbf{R}(\mathbf{x}) d\boldsymbol{\theta}(t)$ where $\mathbf{R}(\mathbf{x}) \mathbf{R}(\mathbf{x})^\top = \mathbf{K}(\mathbf{x})$ and $\boldsymbol{\theta}(t) \sim \mathcal{N}(0, \mathbf{I})$. When $B \ll N$ (in eq. (2.4)), the learning rate and the batch size only appear in the SDE through the ratio $\rho := \eta B^{-1}$. Jastrzębski et al. (2018) argue that thus mainly the ratio ρ controls the stochastic noise and training dynamics, and that ρ positively correlates with wider minima and better generalization—matching with recent experimental successes of ImageNet training (Goyal et al., 2017).

This explanation fails in the large batch regime. When the batch size grows too large, the relative magnitude of the stochastic noise decreases as $\eta B^{-1} \neq \eta(N-B)/BN$, i.e. ρ is not anymore uniquely describing the training dynamics for large batch size and small dataset size when $B \not\ll N$ (Jastrzębski et al., 2018). This could be a reason why the generalization difficulty of large-batch training remains, as clearly illustrated e.g. in Figure 2.1, Table 2.2, Figure 2.4(b), and as well as in Shallue et al. (2019); Golmant et al. (2018).

The local update step of local SGD is a natural and computation-free way to inject well-structured stochastic noise to the SGD training dynamics. By using the same ratio $\frac{\eta}{B}$ as mini-batch SGD, the main difference of the local SGD training dynamics comes from the stochastic noise ϵ during the local update phase (K times smaller local mini-batch with H local update steps). The ϵ will be approximately sampled with the variance matrix $K\Sigma(\mathbf{x})$ instead of $\Sigma(\mathbf{x})$ in mini-batch SGD, causing the stochastic noise determined by K and H to increase. This could be one of the reasons for post-local SGD to generalize as good as mini-batch SGD (small mini-batch size) and leading to flatter minima than large-batch SGD in our experiments. Similar positive effects of adding well-structured stochastic noise to SGD dynamics on non-convex problems have recently also been observed in [Smith and Le \(2018\)](#); [Chaudhari and Soatto \(2018\)](#); [Zhu et al. \(2019\)](#); [Xing et al. \(2018\)](#); [Wen et al. \(2019\)](#).

2.6.1 Post-local SGD Converges to Flatter Minima

Figure 2.5(a) evaluates the spectrum of the Hessian for various local minima. We observe that large-batch SGD tends to get stuck at locations with high Hessian spectrum while post-local SGD tends to prefer low curvature solutions with better generalization error. Figure 2.5(b) linearly interpolates two minima obtained by mini-batch SGD and post-local SGD, and Figure 11.9 (in Appendix) visualizes the sharpness of the model trained by various methods. These results give evidence that post-local SGD converges to flatter minima than mini-batch SGD for training ResNet-20 on CIFAR-10.



(a) The dominant eigenvalue of the Hessian, for model trained from various schemes. It is evaluated on the test dataset per epoch, and only the phase related to the post-local SGD strategy is visualized. Mini-batch SGD or post-local SGD with small H (e.g. $H=2, 4$) have noticeably larger dominant eigenvalue.

(b) 1-d linear interpolation between models $\mathbf{x}_{\text{post-local SGD}} (\lambda = 0)$ and $\mathbf{x}_{\text{mini-batch SGD}} (\lambda = 1)$, i.e. $\hat{\mathbf{x}} = \lambda \mathbf{x}_{\text{mini-batch SGD}} + (1 - \lambda) \mathbf{x}_{\text{post-local SGD}}$. The solid lines correspond to evaluate $\hat{\mathbf{x}}$ on the whole training dataset while the dashed lines are on the test dataset. The model parameters only vary from the post-local SGD phase.

Figure 2.5 – Understanding the generalization ability of post-local SGD for large-batch training (**ResNet-20** on **CIFAR-10** with $BK = B_{\text{loc}}K = 2048$). We use fixed $B = B_{\text{loc}} = 128$ with $K = 16$ GPUs. The detailed experimental setup as well as more visualization of results are available in Appendix 11.3.4.

2.7 Conclusion

We leverage the idea of local SGD for training in distributed and heterogeneous environments. Ours is the first work to extensively study the trade-off between communication efficiency and generalization performance of local SGD. Our local SGD variant, called post-local SGD, not only outperforms large-batch SGD's generalization performance but also matches that of small-batch SGD. In our experiments post-local SGD converged to flatter minima compared to traditional large-batch SGD, which partially explains its improved generalization performance. We also provide extensive experiments with another variant hierarchical local SGD, showing its adaptivity to available system resources. Overall, local SGD comes off as a simpler and more efficient algorithm, replacing complex ad-hoc tricks used for current large-batch SGD training.

Acknowledgements

The authors thank the anonymous reviewers and Thijs Vogels for their precious comments and feedback.

3 Extrapolation for Large-batch Training in Deep Learning

3.1 Preface

Contribution and sources. This chapter reproduces [Lin et al. \(2020a\)](#). The author proposed the initial idea and led the whole project. Together with Lingjing Kong, the author developed/implemented/evaluated several algorithm variants, theoretically analyzed the convergence of the final algorithm, and conducted the experiments. Detailed individual contributions:

Tao Lin (author): Conceptualization, Software (50 %), Experiments (70 %), Formal analysis (50 %), Writing—original draft preparation (60 %).

Lingjing Kong: Software (50 %), Experiments (30 %), Formal analysis (50 %), Writing—original draft preparation (40 %).

Sebastian U. Stich: Writing—review and editing.

Martin Jaggi: Supervision, Administration, Writing—review and editing.

Summary. Deep learning networks are typically trained by Stochastic Gradient Descent (SGD) methods that iteratively improve the model parameters by estimating a gradient on a very small fraction of the training data. A major roadblock faced when increasing the batch size to a substantial fraction of the training data for reducing training time is the persistent degradation in performance (generalization gap). To address this issue, recent work propose to add small perturbations to the model parameters when computing the stochastic gradients and report improved generalization performance due to smoothing effects. However, this approach is poorly understood; it requires often model-specific noise and fine-tuning.

To alleviate these drawbacks, in this chapter we propose to use instead computationally efficient extrapolation (extragradient) to stabilize the optimization trajectory while still benefiting from smoothing to avoid sharp minima. This principled approach is well grounded from an optimization perspective and we show that a host of variations can be covered in a unified framework that we propose. We prove the convergence of this novel scheme and rigorously evaluate its empirical

performance on ResNet, LSTM, and Transformer. We demonstrate that in a variety of experiments the scheme allows scaling to much larger batch sizes than before whilst reaching or surpassing SOTA accuracy.

3.2 Introduction

The workhorse training algorithm for most machine learning applications—including deep learning—is Stochastic Gradient Descent (SGD). Recently, data parallelism has emerged in deep learning, where large-batch (Goyal et al., 2017) is used to reduce the gradient computation rounds so as to accelerate the training. However, in practice, these large-batch variants suffer from severe issues of test quality loss (Shallue et al., 2019; McCandlish et al., 2018; Golmant et al., 2018), which voids gained computational advantage. While still not completely understood, recent research has linked part of this loss of efficiency to the existence of sharp minima. In contrast, landscapes with flat minima have in empirical studies shown generalization benefits (Keskar et al., 2017; Yao et al., 2018; Lin et al., 2020d), though this topic is still actively debated (Dinh et al., 2017).

Another line of research tries to understand general deep learning training from an optimization perspective, in terms of the optimization trajectory in the loss surface (Neysshabur, 2017; Jastrzebski et al., 2020). Golatkar et al. (2019) empirically show that regularization techniques only affect early learning dynamics (initial optimization phase) but matter little in the final phase of training (converging to a local minimum), similar to the critical initial learning phase described in Achille et al. (2019) and the break-even point analysis on the entire optimization trajectory of Jastrzebski et al. (2020).

These new insights are consistent with empirically developed techniques for the SOTA large-batch training. For example, gradual learning rate warmup for the first few epochs (Goyal et al., 2017; You et al., 2019a) is often used; and in local SGD (Lin et al., 2020d) (see Chapter 2) for better generalization, stochastic noise injection is only applied *after* the first phase of training (post-local SGD).

These discussions on optimization and generalization motivate us to answer the following questions when using or developing large-batch techniques for better training: *Does the proposed technique improve (initial) optimization, or help to converge to a better local minimum, or both? How and when should we apply the technique?*

In this paper, we first revisit the classical smoothing idea (which was recently attributed to avoiding sharp minima in deep learning (Wen et al., 2018; Haruki et al., 2019)) from optimization perspective. We then propose a computational efficient local extragradient method as a way of smoothing for distributed large-batch training, referred to as EXTRAP-SGD; we further extend it to a general framework (extrapolated SGD) for distributed training. We thoroughly evaluate our method on diverse tasks to understand how and when it improves the training in practice. Our empirical results justify the benefits of EXTRAP-SGD, and explain the effects of smoothing

ill-conditioned loss landscapes as to exhibit more well-behaved regions, which contain multiple solutions of good generalization properties (Garipov et al., 2018). We show the importance of using EXTRAP-SGD in the critical initial optimization phase, for the later convergence to better local minima; the conjecture is verified by the combination with post-local SGD, which achieves SOTA large-batch training.

Our main contributions can be summarized as follows:

- We propose EXTRAP-SGD and extend it to a unified framework (extrapolated SGD) for distributed large-batch training. Extensive empirical results on three benchmarking tasks justify the effects of accelerated optimization and better generalization.
- We provide convergence analysis for methods in the proposed framework, as well as the SOTA large batch training method (i.e. mini-batch SGD with Nesterov momentum). Our analysis explains the large batch optimization inefficiency (diminishing linear speedup) observed in previous empirical work.

3.3 Related Work

Large-batch training. The test performance degradation (often denoted as *generalization gap*) caused by large batch training has recently drawn significant attention (Keskar et al., 2017; Hoffer et al., 2017; Shallue et al., 2019; Masters and Luschi, 2018). Hoffer et al. (2017) argue that the generalization gap in some cases can be closed by increasing training iterations and adjusting the learning rate proportional to the square root of the batch size. Goyal et al. (2017) argue the poor test performance is due to the optimization issue; they try to bridge the generalization gap with the heuristics of linear scale-up of the learning rate during training or during a warmup phase. You et al. (2017b) propose Layer-wise Adaptive Rate Scaling (LARS) for better optimization and scaling to larger mini-batch sizes; but the generalization gap does not vanish. Lin et al. (2020d) in Chapter 2 further proposes post-local SGD on top of these optimization techniques to inject stochastic noise (to mimic the training dynamics of small batch SGD) during the later training phase.

In addition to the techniques developed for improving optimization and generalization, the optimization ineffectiveness (in terms of required training steps-to-target performance) of large-batch training has been observed. Shallue et al. (2019); McCandlish et al. (2018); Golmant et al. (2018) empirically demonstrate the existence of diminishing linear speedup region across various domains and architectures. Such a limit is also theoretically characterized in Ma et al. (2018b); Yin et al. (2018) for mini-batch SGD in the convex setting.

Smoothing the “sharp minima”. Some research links generalization performance to flatness of minima. Entropy SGD (Chaudhari et al., 2017) proposes Langevin dynamics in the inner optimizer loop to smoothen out sharp valleys of the loss landscape. From the perspective of

large-batch training, [Wen et al. \(2018\)](#) perform “sequential averaging” over models perturbed by isotropic noise, as a way to combat sharp minima. [Haruki et al. \(2019\)](#) claim that injecting various anisotropic stochastic noises on local workers can smoothen sharper minima. However, the claimed “sharper minima” is debatable ([Dinh et al., 2017](#)); it is also unclear whether the improved results are due to obtained flatter local minima or improved initial optimization brought by smoothing. We defer detailed discussion to Section 3.4.2 and 3.6.2.

A similar idea, named Sharpness-Aware Minimization (SAM), was later rediscovered in [Foret et al. \(2021\)](#) to improve the generalization performance of deep learning applications. Different from the computationally efficient past mini-batch gradients in our approach, the extrapolation step in [Foret et al. \(2021\)](#) utilizes fresh mini-batch gradients.

Smoothing in classical optimization. Randomized smoothing has a long history in the optimization literature, see e.g. [Nesterov and Spokoiny \(2017\)](#); [Duchi et al. \(2012\)](#); [Scaman et al. \(2018\)](#) which show that a faster convergence rates can be achieved by convolving non-smooth convex functions with Gaussian noise. In contrast to non-smooth convex functions, we focus on the smooth non-convex functions, motivated by deep neural networks.

Extragradient methods and optimization stability. Another useful building block from optimization is the extragradient method, which is a well-known technique to stabilize the training at each iteration by approximating the implicit update. The method was first introduced in [Korpelevich \(1976\)](#) and extended to many variants, e.g. mirror-prox ([Nemirovski, 2004](#)), Optimistic Mirror Descent (OMD) ([Juditsky et al., 2011](#)) (using past gradient information), extragradient method with Nesterov momentum ([Diakonikolas and Orecchia, 2017](#)). Recently its stochastic variants have found new applications in machine learning, e.g. Generative Adversarial Network (GAN) training ([Daskalakis et al., 2018](#); [Gidel et al., 2019](#); [Chavdarova et al., 2019](#); [Mishchenko et al., 2020](#)), and low bit model training ([Leng et al., 2018](#)).

On the theoretical side, several papers analyze the convergence of stochastic variants of extragradient. [Juditsky et al. \(2011\)](#) study stochastic mirror-prox under restrictive assumptions. [Xu et al. \(2019\)](#) analyze stochastic extragradient in a more general non-convex setting and demonstrate tighter upper bounds than mini-batch SGD, when using mini-batch size $\mathcal{O}(1/\epsilon^2)$. [Mishchenko et al. \(2020\)](#) revisit and slightly extend the stochastic extragradient for better implicit update approximation. However, their work focuses on min-max GAN training and argues the stochastic extragradient method might not be better than SGD for traditional function minimizations tasks. Our work is the first that combines the idea of Nesterov momentum and extragradient (from past information) for stochastic optimization in the setting of distributed training.

3.4 Optimization with Extrapolation

Problem Setting and Notation. We consider sum-structured optimization problems of the form $\min_{\mathbf{x} \in \mathbb{R}^d} f(\mathbf{x}) := \frac{1}{N} \sum_{i=1}^N f_i(\mathbf{x})$, where \mathbf{x} denotes the parameters of the model (neural networks in our case), and f_i denotes the loss function of the i -th (out of N) training data examples. To introduce our notation, we recall a standard update of mini-batch SGD at iteration t , computed on K devices:

$$\mathbf{x}_{t+1} := \mathbf{x}_t - \eta_t \left[\frac{1}{KB} \sum_{k=1}^K \sum_{i \in \mathcal{S}_t^k} \nabla f_i(\mathbf{x}_t) \right]. \quad (3.1)$$

Here \mathcal{S}_t^k denotes a subset of the training points selected on device k (typically selected uniformly at random) and we denote by $B := |\mathcal{S}_t^k|$ the local mini-batch size and by η_t the step-size (learning rate).

3.4.1 Accelerated (Stochastic) Local Extragradient for Distributed Training

Motivated by the idea of randomized smoothing in the classic optimization literature (as for reducing the Lipschitz constant of the gradient), we here introduce the novel idea of using extragradient locally, as a way of smoothing loss surface, for efficient distributed large-batch training.

The original idea of extrapolation (or extragradient (Korpelevich, 1976)) was developed to stabilize optimization dynamics on saddle-point problems for a single worker, such as e.g. in GAN training (Gidel et al., 2019; Chavdarova et al., 2019). The idea is to compute the gradient at an extrapolated point, different from the current point from which the update will be performed:

$$\mathbf{x}_{t+\frac{1}{2}} = \mathbf{x}_t - \eta \nabla f(\mathbf{x}_t), \quad \mathbf{x}_{t+1} = \mathbf{x}_t - \eta \nabla f(\mathbf{x}_{t+\frac{1}{2}}).$$

This step is intrinsically different from the well-known and widely used accelerated method (i.e. Nesterov momentum):

$$\begin{aligned} \mathbf{x}_{t+\frac{1}{2}} &= \mathbf{x}_t + u \mathbf{v}_t, \mathbf{v}_{t+1} = u \mathbf{v}_t - \eta \nabla f(\mathbf{x}_{t+\frac{1}{2}}), \\ \mathbf{x}_{t+1} &= \mathbf{x}_{t+\frac{1}{2}} - \eta \nabla f(\mathbf{x}_{t+\frac{1}{2}}), \end{aligned}$$

where here $1 > u \geq 0$ denotes the momentum parameter. The key difference lies in the lookahead step for the gradient computation of these two methods.

Considering various extrapolated local models (with Nesterov momentum) under the distributed training, EXTRAP-SGD combines the effects of randomized smoothing and the stabilized optimization dynamics through extrapolation. Our EXTRAP-SGD follows the idea of extrapolating from the past (currently only used for single worker training (Gidel et al., 2019; Daskalakis et al., 2018)), as a means to avoid additional cost for gradients used to form an extrapolation point.

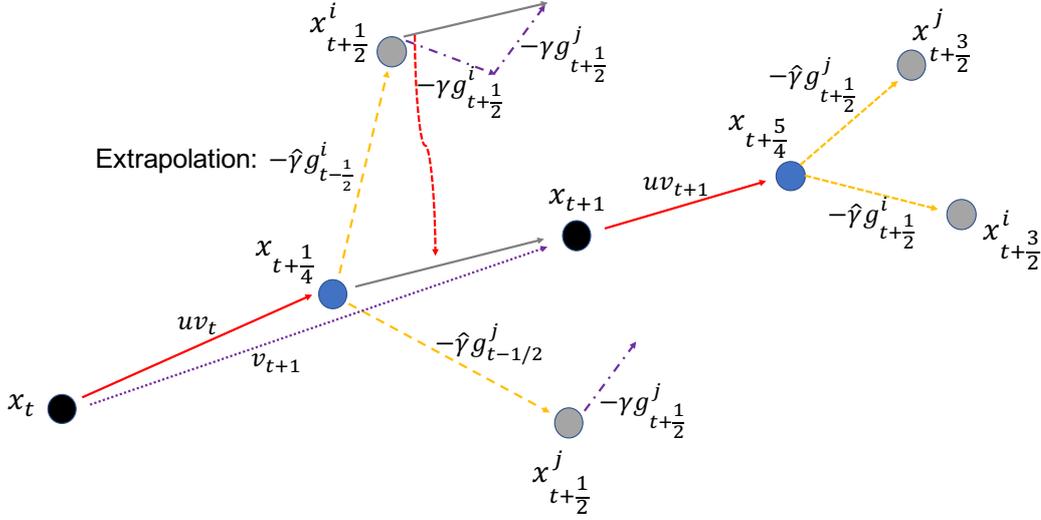


Figure 3.1 – Illustrated EXTRAP-SGD ($K=2$).

Algorithm 1 EXTRAP-SGD.

Requires: learning rate η ; inner learning rate $\hat{\eta}$; momentum factor u ; initial parameter \mathbf{x}_0 ; initial moment vector $\mathbf{v}_0=0$; time step $t=0$, worker index k .

- 1: **while** \mathbf{x}_t not converged **do**
- 2: $\mathbf{x}_{t+1/4}^k = \mathbf{x}_t - \frac{\hat{\eta}}{B} \sum_{i \in \mathcal{S}_t^k} \nabla f_i(\mathbf{x}_{t-1/2}^k)$ ▷ extrapolation step
- 3: $\mathbf{x}_{t+1/2}^k = \mathbf{x}_{t+1/4}^k + u\mathbf{v}_t$ ▷ Nesterov momentum step
- 4: $\mathbf{v}_{t+1} = u\mathbf{v}_t - \frac{\eta}{KB} \sum_{k,i \in \mathcal{S}_t^k} \nabla f_i(\mathbf{x}_{t+1/2}^k)$ ▷ update buffer
- 5: $\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{v}_{t+1}$ ▷ actual update
- 6: **return** \mathbf{x}_{t+1}

The EXTRAP-SGD method is detailed in Algorithm 1 and illustrated in Figure 3.1¹, where the previous local gradients are used for the extrapolation and the superscript k stresses that local models are different. Using the past local mini-batch gradients ($\nabla f_i(\mathbf{x}_{t-1/2}^k)$) for extrapolation (at time t) is a computationally efficient and allows the extrapolation scale $\hat{\eta}$ to directly take on the learning rate used for small mini-batch training with size B , thus avoids the difficulty of hyperparameter tuning. Note that setting $\hat{\eta}=0$ in Algorithm 1 (line 2) recovers the SOTA large batch training method, i.e. mini-batch SGD with Nesterov momentum.

To the best of our knowledge, it is the first time such an extragradient method is used locally with Nesterov Momentum under the framework of smoothing, for accelerated and smoothed distributed optimization.

¹we omit the extrapolation step in line 2 when $t=0$.

3.4.2 Unified Extrapolation Framework

Our Algorithm 1 can be extended to a more general extrapolation framework for distributed training, by using diverse extrapolation choices ζ_t^k . It is achieved by replacing line 2 in Algorithm 1 by $\mathbf{x}_{t+\frac{1}{4}}^k = \mathbf{x}_t - \hat{\eta}\zeta_t^k$. We denote our framework as *extrapolated SGD*, covering various choices of noise ζ_t^k (e.g. Gaussian noise, uniform noise, and stochastic gradient noise) and EXTRAP-SGD (past mini-batch gradients). We detail some choices of ζ_t^k below and use them as our close baselines for EXTRAP-SGD:

- ζ_t^k as a form of isotropic noise. The noise can be sampled from e.g. an isotropic Gaussian or uniform distribution. Following the idea of Li et al. (2018), the strength of noises added to a filter can be linearly scaled by the l_2 norm of the filter, instead of fixing a constant perturbation strength over various layers. Formally, the scaled noise ζ_t^k of j -th filter at layer i on worker k follows $\|\mathbf{x}_{t,i,j}\| \cdot \hat{\zeta}_{t,i,j}^k / \|\zeta_{t,i,j}^k\|$. A similar idea was proposed in SmoothOut (Wen et al., 2018), corresponding to letting $\zeta_t^k := \zeta_t$ in our framework.
- ζ_t^k as a form of anisotropic noise. Kleinberg et al. (2018) interpret sequential SGD updates as GD with stochastic gradient noise convolution over update steps. This motivates to use stochastic gradient noise for smoothing (similarly proposed in Haruki et al. (2019)), thus ζ_t^k can be chosen as:

$$\frac{1}{B} \sum_{i \in \mathcal{S}_t^k} \nabla f_i(\mathbf{x}_{t-\frac{1}{2}}^k) - \frac{1}{KB} \sum_k \sum_{i \in \mathcal{S}_t^k} \nabla f_i(\mathbf{x}_{t-\frac{1}{2}}^k).$$

The side effects of these noise extrapolation variants are the training setup sensitivity, causing the hyperparameter tuning difficulty and limited practical applications. For example, the isotropic noise requires to manually design the noise distribution for each model and dataset; the anisotropic noise distribution will be dynamically varied by the choices of the number of workers, the local mini-batch size, and the objective of the learning task (Zhang et al., 2020).

Despite the existence of variants (Wen et al., 2018; Haruki et al., 2019) and their reported empirical results², none of them has analyzed their convergence behaviors. In the next section, we provide rigorous convergence analysis for our algorithm for distributed training (illustrated in Algorithm 1, which also includes the SOTA practical training algorithm). In Section 3.6, we empirically evaluate all related methods to better understand the benefits of using extrapolation with smoothing for distributed large-batch training.

² The empirical results of Haruki et al. (2019) are not solid. Taking the results of CIFAR-10 for mini-batch size 8,192 into account and use three trials' experimental results for the same choice of ζ_t^k (e.g. layerwise uniform noise) as an example, our experimental results can reach reasonable test top-1 accuracy (at around 91), much better than their presented results (at around 63).

3.5 Theoretical Analysis of Nesterov Momentum and EXTRAP-SGD

We now turn to the theoretical convergence analysis, i.e. we derive an upper bound on the number of iterations to find an approximate solution with small gradient norm.

Following the convention in distributed stochastic optimization, we denote by f^* a lower bound on the values of $f(\mathbf{x})$ and use the following assumptions:

Assumption 1 (Unbiased Stochastic Gradients). $\forall i \in [N], t \in [T]$, it holds $\mathbb{E}[\nabla f_i(\mathbf{x}_t)] = \nabla f(\mathbf{x}_t)$.

Assumption 2 (Bounded Gradient Variance). $\exists \sigma^2 > 0, \forall i \in [N], t \in [T]$, s.t. $\mathbb{E}[\|\nabla f_i(\mathbf{x}_t) - \nabla f(\mathbf{x}_t)\|^2] \leq \sigma^2$.

Here σ^2 quantifies the variance of stochastic gradients at each local worker and we assume workers access IID training dataset (e.g. data center setting).

Assumption 3 (Lipschitz Gradient). $\exists L > 0$, s.t. $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^d, i \in [N]$, the objective function $f_i : \mathbb{R}^d \rightarrow \mathbb{R}$ satisfies the following condition $\|\nabla f_i(\mathbf{x}) - \nabla f_i(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\|$.

3.5.1 Analysis for Mini-batch SGD (with Nesterov Momentum)

In this section we first recall the known convergence guarantees for mini-batch SGD (without momentum) on non-convex functions for later reference, and then derive new guarantees for mini-batch SGD with Nesterov momentum.

Theorem 3.5.1 (Convergence of stochastic distributed mini-batch SGD for non-convex functions (Ghadimi and Lan, 2016)). *Under Assumptions 1–3, after T mini-batch gradient updates, each using KB samples, the mini-batch SGD returns an iterate \mathbf{x} satisfying*

$$\mathbb{E}[\|\nabla f(\mathbf{x})\|^2] \leq \mathcal{O}\left(\frac{L(f(\mathbf{x}_0) - f^*)}{T} + \frac{\sigma\sqrt{L(f(\mathbf{x}_0) - f^*)}}{\sqrt{KB T}}\right).$$

The second term in the rate is asymptotically dominant as long as $KB = \mathcal{O}\left(\frac{\sigma^2 T}{L(f(\mathbf{x}_0) - f^*)}\right)$. In this regime, increasing the mini-batch size gives a linear speedup, as $T = \mathcal{O}\left(\frac{\sigma^2 L(f(\mathbf{x}) - f^*)}{KB \epsilon^2}\right)$ decreases in KB , as similarly pointed out by Wang and Srebro (2019). However, when we increase the mini-batch size beyond this critical point, the first term dominates the rate and increasing the mini-batch size further will have less impact on the convergence. This phenomenon has also been empirically verified in deep learning applications (Shallue et al., 2019).

In practice, mini-batch SGD with Nesterov momentum (Nesterov, 1983) is the state-of-the-art deep learning training scheme. However, previous theoretical analysis normally relies on the strong assumption of the bounded mini-batch gradients (Yan et al., 2018). Here we provide a better convergence analysis without such an assumption for distributed mini-batch SGD with

3.5. Theoretical Analysis of Nesterov Momentum and EXTRAP-SGD

Nesterov momentum following closely [Yu et al. \(2019a\)](#). The convergence rate is detailed in Theorem 3.5.2, and for the proof details we refer to Section 12.1 in Appendix.

Theorem 3.5.2 (Convergence of mini-batch SGD with Nesterov momentum for non-convex functions). *Under Assumption 1–3, for mini-batch SGD with Nesterov momentum, i.e. $\mathbf{x}_{t+\frac{1}{2}} = \mathbf{x}_t + u\mathbf{v}_t$, $\mathbf{v}_{t+1} = u\mathbf{v}_t - \frac{\eta}{KB} \sum_{k=1}^K \sum_{i \in \mathcal{S}_t^k} \nabla f_i(\mathbf{x}_{t+\frac{1}{2}})$, $\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{v}_{t+1}$, we can show that for optimally tuned stepsize (cf. Lemma 12.1.4) $\eta \leq \frac{2(1-u)^2}{L(u^3+1)}$, it holds*

$$\mathbb{E} \left[\|\nabla f(\mathbf{x})\|^2 \right] = \mathcal{O} \left(\frac{Lr_0(u^3+1)}{T(1-u)} + \sqrt{\frac{2Lr_0\sigma^2}{KBT(1-u)}} \right),$$

where here \mathbf{x} denotes a uniformly at random selected $\mathbf{x}_{t+\frac{1}{2}}$ iterate, i.e. $\frac{1}{T} \sum_{t=0}^{T-1} \|\nabla f(\mathbf{x}_{t+\frac{1}{2}})\|^2$, and $r_0 := f(\mathbf{x}_0) - f^*$.

The second term is asymptotically dominant as long as $KB = \mathcal{O} \left(\frac{(1-u)}{(u^3+1)^2} \frac{\sigma^2 T}{L(f(\mathbf{x}_0) - f^*)} \right)$ and for small K we can achieve the linear speedup where $T = \mathcal{O} \left(\frac{L\sigma^2(f(\mathbf{x}_0) - f^*)}{(1-u)KB\epsilon^2} \right)$.

Remark 3.5.3. [Yu et al. \(2019a\)](#) argue that a linear speedup $\mathcal{O}(1/\sqrt{KT})$ for SGD with the local Nesterov momentum can be achieved. However, this claim is only valid for large T and stepsize $\eta = \sqrt{K/T}$, cf. ([Yu et al., 2019a, Cor. 1](#)). We here tune the stepsize differently and show a tighter bound that holds for all T , providing a better critical mini-batch size analysis (diminishing linear speedup in terms of optimization) for mini-batch SGD with Nesterov momentum ([Shallue et al., 2019](#)).

3.5.2 Convergence of EXTRAP-SGD

In this subsection, we show the convergence analysis for our novel EXTRAP-SGD for non-convex functions. We also include the analysis for other noise variants of extrapolated SGD, which explains their potential limitations. The proof details can be found in Appendix (Section 12.2).

Theorem 3.5.4 (Convergence of EXTRAP-SGD for non-convex functions). *Under Assumption 1–3, and by defining $\bar{\mathbf{x}}_{t+\frac{1}{2}} := \frac{1}{K} \sum_{k=1}^K \mathbf{x}_{t+\frac{1}{2}}^k$, it holds for $\hat{\eta} \leq \frac{u^2}{(1-u)^2} \eta$ and $\eta \leq \frac{(1-u)^2}{L(1+3u+u^3)}$:*

$$\mathbb{E} \frac{1}{T} \sum_{t=0}^{T-1} \left\| \nabla f(\bar{\mathbf{x}}_{t+\frac{1}{2}}) \right\|^2 \leq \frac{2(1-u)}{\eta T} \mathbb{E} [f(\bar{\mathbf{x}}_0) - f^*] + \left(\frac{4\hat{\eta}^2 L^2}{B} + \frac{\eta L(1+3u)}{(1-u)^2 BK} \right) \sigma^2.$$

Remark 3.5.5. Using past local gradients for extrapolation in EXTRAP-SGD allows us to directly set $\hat{\eta} \approx \frac{\eta}{K}$ for EXTRAP-SGD, where the constraint of $\hat{\eta} \leq \frac{u^2}{(1-u)^2} \eta$ is normally satisfied.

Corollary 3.5.6. *Considering Theorem 3.5.4 and tuning the stepsize as in Lemma 12.1.4, with $\eta \leq \frac{(1-u)^2}{L(1+3u+u^3)}$ and $\hat{\eta} \leq \frac{\eta}{K}$, and $r_0 := f(\mathbf{x}_0) - f^*$, we can rewrite the convergence rate of Theorem 3.5.4*

as

$$\mathbb{E} \left[\frac{1}{T} \sum_{t=0}^{T-1} \left\| \nabla f(\bar{\mathbf{x}}_{t+\frac{1}{2}}) \right\|^2 \right] = \mathcal{O} \left(4(u^3 + 3u + 1) \frac{Lr_0}{T(1-u)} + 2\sqrt{\frac{(19u+1)}{(u^3+3u+1)} \frac{2Lr_0\sigma^2}{KBT(1-u)}} \right).$$

The second term is asymptotically dominant as long as $KB = \mathcal{O}\left(\frac{(19u+1)(1-u)}{(u^3+3u+1)^3} \frac{\sigma^2 T}{L(f(\mathbf{x}_0)-f^*)}\right)$ and for small K we can achieve the linear speedup where $T = \mathcal{O}\left(\frac{L\sigma^2(f(\mathbf{x}_0)-f^*)}{KB\epsilon^2} \frac{19u+1}{(u^3+3u+1)(1-u)}\right)$,

Remark 3.5.7. By setting $u = 0$, we recover the same rates for EXTRAP-SGD in non-convex cases (Theorem 3.5.2 and Corollary 3.5.6) as for standard mini-batch SGD (Theorem 3.5.1) but we cannot show an actual speedup over mini-batch SGD by setting $u > 0$. However, thus might not necessarily be a limitation of our approach as to the best of our knowledge, there exist so far no theoretical results for stochastic momentum methods that can show a speedup over mini-batch SGD.

The analysis below extends the proof of EXTRAP-SGD to the other cases of our extrapolated SGD framework.

Theorem 3.5.8. Under the extrapolation framework, IID random noise ζ_t^k (instead of the past local mini-batch gradients in Algorithm 1) is used for the local extrapolation, where $\mathbb{E}[\zeta_t^k] = 0$ and $\mathbb{E}[\|\zeta_t^k\|^2] \leq \hat{\sigma}^2$. Under Assumption 1–3, it holds for $\eta \leq \frac{(1-u)^2}{L(1+u+u^3)}$:

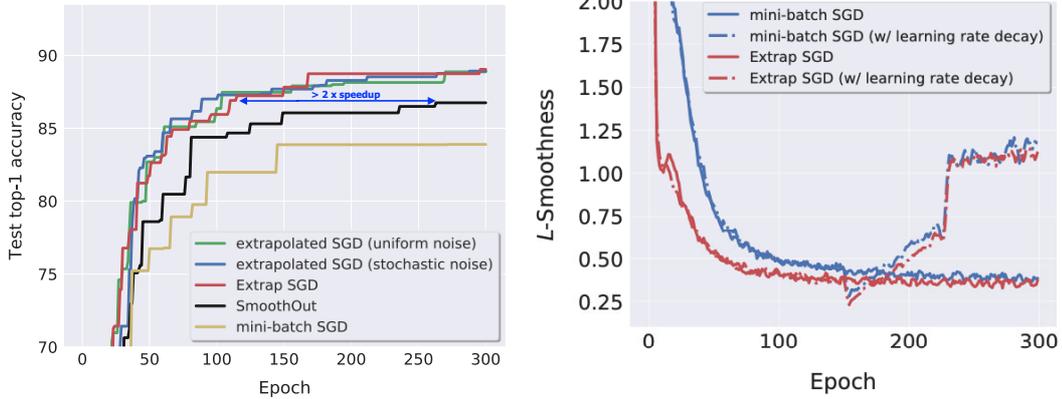
$$\mathbb{E} \frac{1}{T} \sum_{t=0}^{T-1} \left\| \nabla f(\bar{\mathbf{x}}_{t+\frac{1}{2}}) \right\|^2 \leq \frac{2(1-u)}{\eta T} \mathbb{E}[f(\bar{\mathbf{y}}_0) - f(\bar{\mathbf{y}}_T)] + \frac{\eta L(1+u)}{(1-u)^2 BK} \sigma^2 + (L^2 + \frac{(1-u)^2 L}{\eta u^3 K}) 2\hat{\eta}^2 T \hat{\sigma}^2.$$

Remark 3.5.9. The choice of using random noise for extrapolation in Theorem 3.5.8 requires the manual introduction of the noise distribution ($\hat{\sigma}^2$) for each problem setup. The dependence on the unknown relationship between $\hat{\sigma}^2$ and σ^2 results in the difficulty of providing a concise convergence analysis (e.g. exact convergence rate, the critical mini-batch size) for Theorem 3.5.8.

3.6 Experiments

We first briefly outline the general experimental setup below (for more details refer to Appendix 12.3) and then thoroughly evaluate our framework on various challenging large-batch training tasks. We limit our attention to three standard and representative benchmarking tasks, with a controlled epoch budget (for each task). We ensure the used mini-batch size is a significant fraction of the whole dataset. Performing experiments on a much larger dataset for the same demonstration purposes is out of our computational ability.³

³ E.g. we use the mini-batch size of 8,192 (roughly 16% of the total data), out of 50,000 samples for CIFAR as our main tool of justification. While for ImageNet (Russakovsky et al., 2015) with 1.28 million data samples in total, the same fraction would result in roughly 800 workers for local mini-batch of size 256.



(a) Learning curves for various methods (w/o learning rate decay).

(b) Better smoothness of EXTRAP-SGD.

Figure 3.2 – Understanding the learning behaviors of various methods on the large-batch training (with mini-batch size 8,192 on 32 workers) for training ResNet-20 on CIFAR-10. The visualization of smoothness in Figure 3.2(b) follows the idea in Santurkar et al. (2018); Haruki et al. (2019); it takes 8 additional steps (each with 30% of the update) in the direction of the update for each training step, and the smoothness of a training step is expressed by the maximum value of L (evaluated after local update steps) satisfying the Assumption 3. We use the learning rate scaling and warmup in Goyal et al. (2017) for the first 5 epochs, and η and $\bar{\eta}$ are fine-tuned for various methods.

3.6.1 Experimental Setup

Datasets. We evaluate all methods on the following three tasks: (1) Image Classification for CIFAR-10/100 (Krizhevsky and Hinton, 2009) (50K training samples and 10K testing samples with 10/100 classes) with the standard data augmentation and preprocessing scheme (He et al., 2016a; Huang et al., 2016); (2) Language Modeling for WikiText2 (Merity et al., 2017) (the vocabulary size is 33K, and its train and validation set have 2 million tokens and 217K tokens respectively); and (3) Neural Machine Translation for Multi30k (Elliott et al., 2016).

Models and training schemes. Several benchmarking models are used in our experimental evaluation. (1) ResNet-20 (He et al., 2016a) and VGG-11 (Simonyan and Zisserman, 2015) on CIFAR for image classification, (2) two-layer LSTM (Merity et al., 2018) with hidden dimension of size 128 on WikiText-2 for language modeling, and (3) a down-scaled transformer (factor of 2 w.r.t. the base model in Vaswani et al. (2017)) for neural machine translation. Weight initialization schemes for the three tasks follow Goyal et al. (2017); He et al. (2015), Merity et al. (2018) and Vaswani et al. (2017) respectively.

We use mini-batch SGD with a Nesterov momentum of 0.9 without dampening for image classification and language modeling tasks, and Adam for neural machine translation tasks. In the following experiment section, the term “mini-batch SGD” indicates the mini-batch SGD with Nesterov momentum unless mentioned otherwise.

For experiments on image classification and language modeling, unless mentioned otherwise the models are trained for 300 epochs; the local mini-batch sizes are set to 256 and 64 respectively. By default, all related experiments will use learning rate scaling and warmup scheme⁴ (Goyal et al., 2017; Hoffer et al., 2017). The learning rate is always gradually warmed up from a relatively small value for the first few epochs. Besides, the learning rate η in image classification task will be dropped by a factor of 10 when the model has accessed 50% and 75% of the total number of training samples (He et al., 2016a; Huang et al., 2017b). The LARS is only applied on image classification task⁵ (You et al., 2017c).

For experiments on neural machine translation, we use standard inverse square root learning rate schedule (Vaswani et al., 2017). The warmup step is set to 4000 for the mini-batch size of 64 and will be linearly scaled down by the global mini-batch size.⁶

We carefully tune the learning rate η and the trust term $\tilde{\eta}$ in You et al. (2017c). The tuning procedure ensures that the best hyperparameter lies in the middle of our search grids; otherwise, we extend our search grid. The procedure of hyperparameter tuning can be found in Appendix 12.3.1.

3.6.2 Evaluation on Large-batch Training

Superior performance of EXTRAP-SGD on various tasks. We evaluate our extrapolation framework and compare it with SOTA large-batch training methods on CIFAR-10 image classification (Figure 3.2) and WikiText2 language modeling (Figure 3.3). To better exhibit the optimization behaviors of various methods, for these two tasks in this section we do not decay the learning rate. *The extrapolated SGD framework in general significantly accelerates the optimization and leads to better test performance than the existing SOTA methods.* For example, the smaller gradient Lipschitz constant illustrated in Figure 3.2(b) demonstrates the improved optimization landscape, which explains the at least 2× speedup in terms of the convergence (after thorough hyperparameter tuning) in Figure 3.2(a) and Figure 3.3.

We further extend⁷ the extrapolation idea of EXTRAP-SGD to EXTRAP-ADAM and validate its effectiveness (compared with Adam) on neural machine translation with Transformer. The algorithmic description refers to Algorithm 13 in Appendix 12.4.2. Figure 3.4 shows the results of large-batch training (using 4% and 16% of the training data per mini-batch) and EXTRAP-ADAM again outperforms the Adam with at least 2× speedup.

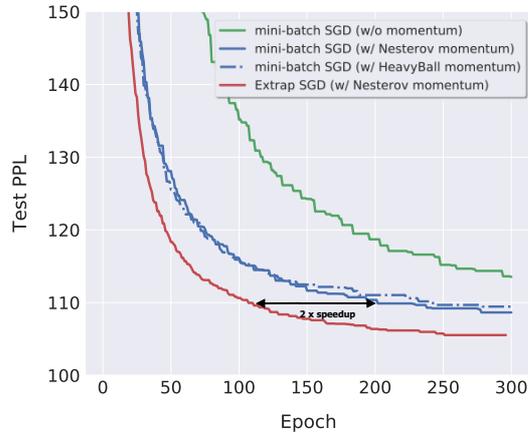
Optimization v.s. Generalization benefits. To better understand when and how EXTRAP-SGD (and extrapolated SGD) help, we switch our attention to the commonly accepted training

⁴ Since we will fine-tune the (to be scaled) learning rate, there is no difference between learning rate linear scaling (Goyal et al., 2017) and square root scaling (Hoffer et al., 2017) in our case.

⁵ Our implementation relies on the PyTorch extension of NVIDIA apex for mixed precision and distributed training.

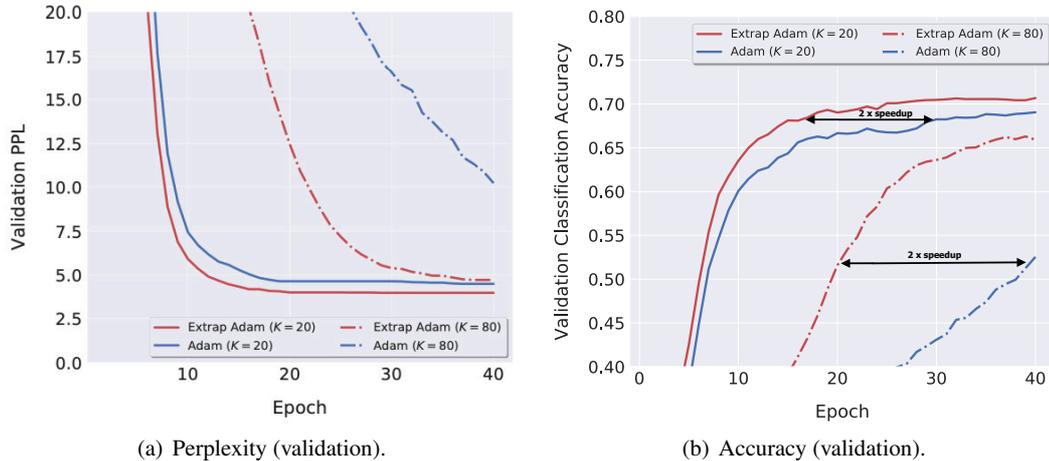
⁶ We follow the practical instruction from NVIDIA.

⁷ It is non-trivial to adapt Adam to the noise variants of the extrapolated SGD.



	$K=24$	$K=48$
mini-batch SGD	108.39 ± 0.31	110.16 ± 0.67
EXTRAP-SGD	105.86 ± 0.32	107.86 ± 0.50

Figure 3.3 – The perplexity (PPL, the lower the better) of training LSTM on WikiText-2. The global mini-batch size are 1,536 and 3,072 for $K=24$ and $K=48$ respectively, accounting for 2% and 4% of the total training data. We use the learning rate scaling and warmup in Goyal et al. (2017), and use constant learning rate after the warmup. We finetune the η for mini-batch SGD (and its momentum variants); EXTRAP-SGD reuses the hyperparameters from mini-batch SGD. The results of the inline table are averaged over three seeds. The displayed learning curves are based on $K=24$ and more details refer to Appendix 12.5.2.



(a) Perplexity (validation).

(b) Accuracy (validation).

Figure 3.4 – Training EXTRAP-ADAM on Multi30k with Transformer. The evaluations are performed on the validation dataset, for the training on $K=20$ and $K=80$ workers (corresponding to roughly 4% and 16% of the total training data). We use the standard inverse square root learning rate schedule (Vaswani et al., 2017) and scale the warmup step based on the number of workers.

Chapter 3. Extrapolation for Large-batch Training in Deep Learning

Table 3.1 – The performance comparison of various methods for large-batch training on CIFAR-10 (mini-batch of size 8,192 on $K=32$). Two neural architectures (ResNet-20 and VGG-11) are considered (with and without batch normalization). Unless mentioned otherwise each method will use the learning rate scaling and warmup in Goyal et al. (2017) and LARS in You et al. (2017c). We fine-tune η and $\hat{\eta}$ for mini-batch SGD and SmoothOut (with $\hat{\eta}$ additionally). For the results of extrapolated SGD, we reuse the optimal η and $\hat{\eta}$ tuned on mini-batch SGD; $\hat{\eta}$ is fine-tuned for noise-based extrapolation variants. The results are averaged over three seeds.

	mini-batch SGD (w/o LARS)	EXTRAP-SGD (w/o LARS)	mini-batch SGD	SmoothOut (Wen et al., 2018)	extrapolated SGD, uniform noise	extrapolated SGD, stochastic noise	EXTRAP-SGD
ResNet-20 on CIFAR-10	90.00 ± 0.48	90.47 ± 0.16	91.36 ± 0.19	91.55 ± 0.20	91.53 ± 0.25	91.66 ± 0.24	91.72 ± 0.11
VGG-11 on CIFAR-10	73.09 ± 9.35	76.79 ± 3.5	86.64 ± 0.10	86.92 ± 0.15	87.00 ± 0.31	86.04 ± 0.43	87.00 ± 0.26

practices: using an initial large learning rate and decaying when the training plateaus. The common beliefs⁸ (LeCun et al., 1991; Kleinberg et al., 2018) argue that, the initial large learning rate accelerates the transition from random initialization to convergence regions (optimization), and the decaying leads the convergence to local minimum (generalization).

Table 3.1 thoroughly evaluates the large-batch training performance (ResNet-20 on CIFAR-10 for mini-batch size 8,192, with learning rate decay schedule) for all related methods. Though the remarkable optimization improvements in Figure 3.2(a) justify *the effects of EXTRAP-SGD (and extrapolated SGD)*, i.e. *smooth the ill-conditioned loss landscape*, decaying the learning rate diminishes our advantages, as illustrated in Table 3.1 and Figure 12.1 in Appendix. *We argue that EXTRAP-SGD and its variants can smoothen the loss surface thus avoid some bad local minima regions, but it cannot guarantee to converge to a much better local minimum*⁹, contrary to the statements in Wen et al. (2018); Haruki et al. (2019). They use (limited) empirical generalization metrics as the main arguments of the “flatter minima”, ignoring the complex training dynamics; their ignored SOTA optimization techniques (e.g. LARS in some experiments) might also result in the improper understandings. Our empirical results provide insights: *the primary benefits of EXTRAP-SGD are on the optimization phase (early training phase) and will diminish in the later training phase*. It is also reflected in Figure 3.2(b) in terms of the smoothness.

Combining EXTRAP-SGD with post-local SGD. Given our new insights in the previous paragraph, here we try to understand the importance of better optimization brought by EXTRAP-SGD for the eventual generalization. We consider the post-local SGD in Lin et al. (2020d) (see Chapter 2), a known technique arguing to converge to flatter local minimum for better generalization. This choice comes from the noticeable optimization benefits of EXTRAP-SGD in the initial training phase while post-local SGD targeting to converge to “flatter minima” for the later training phase. Please refer to Algorithm 12 in Appendix 12.4.1 for training details.

We argue that EXTRAP-SGD biases the optimization trajectory towards a better-conditioned loss surface, where solutions with good generalization properties can be found more easily. Figure 3.5

⁸ Recent works (Li et al., 2019; You et al., 2020a) complement the understanding of this phenomenon from learning differing patterns via diverse learning rate scales; we leave the connection to this aspect for future work.

⁹ Additional experiments show that switching EXTRAP-SGD to mini-batch SGD after the first learning rate decay has similar test performance as using EXTRAP-SGD alone.

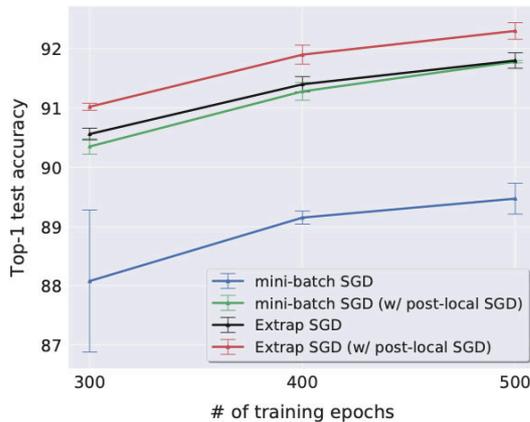


Figure 3.5 – The test top-1 accuracy of integrating EXTRAP-SGD with post-local SGD (Lin et al., 2020d). The performance of ResNet-20 on CIFAR-10 is evaluated with a global mini-batch size of 16,384 (33% of the total training data with 64 workers). By default we use the learning rate scaling and warmup in Goyal et al. (2017) and LARS in You et al. (2017c). We individually fine-tune η and $\tilde{\eta}$ for each base method; the local update step H is tuned and set to $H=4$. The learning rate is decayed by 10 when the model has accessed 50% and 75% of the total training samples. The results are averaged over three seeds.

Table 3.2 – The test top-1 accuracy of integrating EXTRAP-SGD with post-local SGD (Lin et al., 2020d). The performance of ResNet-20 on CIFAR-10/100 is evaluated with global mini-batch size 8,192 ($K=32$). By default we use the learning rate scaling and warmup in Goyal et al. (2017) and LARS in You et al. (2017c). We individually finetune η and $\tilde{\eta}$ for each method; the local update step H tuned from $\{4, 8, 16\}$ (as in Lin et al. (2020d)) in general improves the performance and we report the best performance with $H=8$. The results are averaged over three seeds.

	mini-batch SGD	mini-batch SGD (with post-local SGD)	EXTRAP-SGD	EXTRAP-SGD (with post-local SGD)
CIFAR-10	91.36 \pm 0.19	91.73 \pm 0.25	91.72 \pm 0.11	92.23 \pm 0.02
CIFAR-100	65.79 \pm 0.46	67.39 \pm 0.18	66.63 \pm 0.32	68.06 \pm 0.21

challenges the extreme large-batch training (mini-batch of size 16,384 on 64 workers, accounting for 33% of training data) for ResNet-20 on CIFAR-10 with various epoch budgets. We can notice that the improper optimization in the critical initial learning phase of the mini-batch SGD results in a significant generalization gap, which cannot be addressed by adding post-local SGD or increasing the number of training epochs alone. EXTRAP-SGD, on the contrary, avoids the regions containing bad local minima, complementing the ability of post-local SGD for converging to better solutions. Table 3.2 additionally reports a similar observation for mini-batch of size 8,192 on more datasets.

3.6.3 Ablation Study

EXTRAP-SGD for various local mini-batch sizes and number of workers. Table 12.1 in the Appendix 12.5.1 evaluates how the combinations of the local mini-batch size and the number of workers will impact the performance, for a given global mini-batch size (8,192 for ResNet-20 on CIFAR-10). *The benefits of EXTRAP-SGD can be further pronounced when increasing the*

Chapter 3. Extrapolation for Large-batch Training in Deep Learning

number of workers, which is the common practice for large-batch training. Similar observation can be found in Figure 3.4 for the increased number of workers (as well as the global mini-batch size).

Understanding the effect of momentum. Given the mixed effects of local extrapolation and momentum acceleration in EXTRAP-SGD, Figure 12.3 in Appendix 12.5.3 decouples these two factors for the training ResNet-20 on CIFAR-10 with mini-batch of size 8,192. We can witness that (1) EXTRAP-SGD can take advantages of both extragradient as well as the acceleration from the momentum, and thus can always be applied for accelerated and stabilized distributed optimization; (2) using extrapolation alone (no momentum) in EXTRAP-SGD still outperforms mini-batch SGD with tuned momentum factor; (3) tuning momentum factor for mini-batch SGD can marginally improve the optimization performance but cannot eliminate the optimization difficulty.

3.7 Conclusion

In this work, we adopt the idea of randomized smoothing to distributed training and propose EXTRAP-SGD to perform extrapolation with past local mini-batch gradients. The idea further extends to a unified framework, covering multiple noise extrapolation variants. We provide convergence guarantees for methods within this framework, and empirically justify the remarkable benefits of our methods on image classification, language modeling and neural machine translation tasks. We further investigate and understand the properties of our methods; the algorithms smoothen the ill-conditioned loss landscape for faster optimization and biases the optimization trajectory to well-conditioned regions. These insights further motivate us to combine our methods with post-local SGD for SOTA large-batch training.

4 Decentralized Deep Learning with Arbitrary Communication Compression

4.1 Preface

Contribution and sources. This chapter is part of [Koloskova et al. \(2020a\)](#). The design and evaluation of deep learning experiments, and practical efficient implementation of the algorithm, were all conducted by the author. The initial idea and theory parts were mostly carried out by Anastasia Koloskova. Detailed individual contributions:

Tao Lin (author): Conceptualization (20 %), Software, Experiments, Writing—original draft preparation (20 %).

Anastasia Koloskova: Conceptualization (80 %), Formal analysis, Writing—original draft preparation (80 %).

Sebastian U. Stich: Writing—review and editing.

Martin Jaggi: Supervision, Administration, Writing—review and editing.

Summary. Decentralized training of deep learning models is a key element for enabling data privacy and on-device learning over networks, as well as for efficient scaling to large compute clusters. As current approaches are limited by network bandwidth, we propose the use of communication compression in the decentralized training context. We show that CHOCO-SGD achieves linear speedup in the number of workers for arbitrary high compression ratios on general *non-convex* functions, and non-IID training data. We demonstrate the practical performance of the algorithm in two key scenarios: the training of deep learning models (i) over decentralized user devices, connected by a peer-to-peer network and (ii) in a datacenter.

4.2 Introduction

Distributed machine learning—i.e. the training of machine learning models using distributed optimization algorithms—has recently enabled many successful applications in research and industry. Such methods offer two of the key success factors: 1) *computational scalability* by leveraging the simultaneous computational power of many devices, and 2) *data-locality*, the ability to perform joint training while keeping each part of the training data local to each participating device. Recent theoretical results indicate that decentralized schemes can be as efficient as the centralized approaches, at least when considering convergence of training loss v.s. iterations (Scaman et al., 2017, 2018; Lian et al., 2017; Tang et al., 2018a; Koloskova et al., 2019; Assran et al., 2019).

Gradient compression techniques have been proposed for the standard distributed training case (Alistarh et al., 2017; Wen et al., 2017; Lin et al., 2018; Wangni et al., 2018; Stich et al., 2018), to reduce the amount of data that has to be sent over each communication link in the network. For decentralized training of deep neural networks, Tang et al. (2018a) introduce two algorithms (DCD, ECD) which allow for communication compression. However, both these algorithms are restrictive with respect to the used compression operators, only allowing for unbiased compressors and—more significantly—so far not supporting arbitrarily high compression ratios. We here study CHOCO-SGD—recently introduced for convex problems only (Koloskova et al., 2019)—which overcomes these constraints.

For the evaluation of our algorithm we in particular focus on the generalization performance (on the test-set) on standard machine learning benchmarks, hereby departing from previous work such as e.g. Tang et al. (2018a); Reisizadeh et al. (2019a); Tang et al. (2019); Reisizadeh et al. (2019b) that mostly considered training performance (on the train-set). We study two different scenarios: firstly, (i) training on a challenging peer-to-peer setting, where the training data is distributed over the training devices (and not allowed to move), similar to the federated learning setting (McMahan et al., 2017). We are again able to show speed-ups for CHOCO-SGD over the decentralized baseline (Lian et al., 2017) with much less communication overhead. Secondly, (ii) training in a datacenter setting, where decentralized communication patterns allow better scalability than centralized approaches. For this setting we show that communication efficient CHOCO-SGD can improve time-to-accuracy on large tasks, such as e.g. ImageNet training. However, when investigating the scaling of decentralized algorithms to larger number of nodes we observe that (all) decentralized schemes encounter difficulties and often do not reach the same (test and train) performance as centralized schemes. As these findings point out some deficiencies of current decentralized training schemes (and are not particular to our scheme) we think that reporting these results is a helpful contribution to the community to spur further research on decentralized training schemes that scale to large number of peers.

Contributions. Our contributions can be summarized as:

- On the theory side, we are the first to show that CHOCO-SGD converges at rate $\mathcal{O}(1/\sqrt{nT} + 1/(\rho^2\delta T)^{2/3})$

on non-convex smooth functions, where n denotes the number of nodes, T the number of iterations, ρ the spectral gap of the mixing matrix and δ the compression ratio. The main term, $\mathcal{O}(1/\sqrt{nT})$, matches with the centralized baselines with exact communication and shows a linear speedup in the number of workers n . Both ρ and δ only affect the asymptotically smaller second term.

- On the practical side, we present a version of CHOCO-SGD with momentum and analyze its practical performance on two relevant scenarios:
 - for *on-device training* over a realistic peer-to-peer social network, where lowering the bandwidth requirements of joint training is especially impactful
 - in a datacenter setting for *computational scalability* of training deep learning models for resource efficiency and improved time-to-accuracy
- Lastly, we systematically investigate performance of the decentralized schemes when scaling to larger number of nodes and we point out some (shared) difficulties encountered by current decentralized learning approaches.

4.3 Related Work

For the training in communication restricted settings a variety of methods have been proposed. For instance, decentralized schemes (Lian et al., 2017; Nedić et al., 2018; Koloskova et al., 2019), gradient compression (Seide et al., 2014; Strom, 2015; Alistarh et al., 2017; Wen et al., 2017; Lin et al., 2018; Wangni et al., 2018; Bernstein et al., 2018; Lin et al., 2018; Alistarh et al., 2018; Stich et al., 2018; Karimireddy et al., 2019), asynchronous methods (Recht et al., 2011; Assran et al., 2019) or performing multiple local SGD steps before averaging (Zhang et al., 2016; McMahan et al., 2017; Lin et al., 2020d). This especially covers learning over decentralized data, as extensively studied in the federated learning literature for the centralized algorithms (McMahan et al., 2016). In this paper we advocate for combining decentralized SGD schemes with gradient compression.

Decentralized SGD. We in particular focus on approaches based on gossip averaging (Kempe et al., 2003; Xiao and Boyd, 2004; Boyd et al., 2006) whose convergence rate typically depends on the spectral gap $\rho \geq 0$ of the mixing matrix (Xiao and Boyd, 2004). Lian et al. (2017) combine SGD with gossip averaging and show that the leading term in the convergence rate $\mathcal{O}(1/\sqrt{nT})$ is consistent with the convergence of the centralized mini-batch SGD (Dekel et al., 2012) and the spectral gap only affects the asymptotically smaller terms. Similar results have been observed very recently for related schemes (Scaman et al., 2017, 2018; Koloskova et al., 2019; Yu et al., 2019a).

Quantization. Communication compression with quantization has been popularized in the deep learning community by the reported successes in Seide et al. (2014); Strom (2015). Theoretical guarantees were first established for schemes with unbiased compression (Alistarh et al., 2017; Wen et al., 2017; Wangni et al., 2018) but soon extended to biased compression (Bernstein et al., 2018) as well. Schemes with error correction work often best in practice and give the best

Chapter 4. Decentralized Deep Learning with Arbitrary Communication Compression

theoretical guarantees (Lin et al., 2018; Alistarh et al., 2018; Stich et al., 2018; Karimireddy et al., 2019). Recently, also proximal updates and variance reduction have been studied in combination with quantized updates (Mishchenko et al., 2019; Horváth et al., 2019).

Decentralized Optimization with Quantization. It has been observed that gossip averaging can diverge (or not converge to the correct solution) in the presence of quantization noise (Xiao et al., 2005; Carli et al., 2007; Nedic et al., 2008; Dimakis et al., 2010; Carli et al., 2010b; Yuan et al., 2012). Reisizadeh et al. (2019a) propose an algorithm that can still converge, though at a slower rate than the exact scheme. Another line of work proposed adaptive schemes (with increasing compression accuracy) that converge at the expense of higher communication cost (Carli et al., 2010a; Doan et al., 2018; Berahas et al., 2019). For deep learning applications, Tang et al. (2018a) proposed the DCD and ECD algorithms that converge at the same rate as the centralized baseline though only for *constant* compression ratio. The CHOCO-SGD algorithm that we consider in this work can deal with *arbitrary* high compression, and has been introduced in Koloskova et al. (2019) but only been analyzed for convex functions. For non-convex functions we show a rate of $\mathcal{O}(1/\sqrt{nT} + 1/(\rho^2 \delta T)^{\frac{2}{3}})$, where $\delta > 0$ measures the compression quality. Simultaneous work of Tang et al. (2019) introduced DeepSqueeze, an alternative method which also converges with arbitrary compression ratio. In our experiments, under the same amount of tuning, CHOCO-SGD achieves higher test accuracy.

4.4 CHOCO-SGD

In this section we formally introduce the decentralized optimization problem, compression operators, and the gossip-based stochastic optimization algorithm CHOCO-SGD from Koloskova et al. (2019).

Distributed Setup. We consider optimization problems distributed across n nodes of the form

$$f^* := \min_{\mathbf{x} \in \mathbb{R}^d} \left[f(\mathbf{x}) := \frac{1}{n} \sum_{i=1}^n f_i(\mathbf{x}) \right], \quad f_i(\mathbf{x}) := \mathbb{E}_{\xi_i \sim D_i} F_i(\mathbf{x}, \xi_i), \quad \forall i \in [n], \quad (4.1)$$

where D_1, \dots, D_n are local distributions for sampling data which can be different on every node, $F_i: \mathbb{R}^d \times \Omega \rightarrow \mathbb{R}$ are possibly non-convex (and non-identical) loss functions. This setting covers the important case of empirical risk minimization in distributed machine learning and deep learning applications.

Communication. Every device is only allowed to communicate with its local neighbours defined by the network *topology*, given as a weighted graph $G = ([n], E)$, with edges E representing the communication links along which messages (e.g. model updates) can be exchanged. We assign a positive weight w_{ij} to every edge ($w_{ij} = 0$ for disconnected nodes $\{i, j\} \notin E$).

Assumption 4 (Mixing matrix). *We assume that $W \in [0, 1]^{n \times n}$, $(W)_{ij} = w_{ij}$ is a symmetric ($W = W^\top$) doubly stochastic ($W\mathbf{1} = \mathbf{1}, \mathbf{1}^\top W = \mathbf{1}^\top$) matrix with eigenvalues $1 = |\lambda_1(W)| > |\lambda_2(W)| \geq \dots \geq |\lambda_n(W)|$ and spectral gap $\rho := 1 - |\lambda_2(W)| \in (0, 1]$.*

In our experiments we set the weights based on the local node degrees: $w_{ij} = \max\{\deg(i), \deg(j)\}^{-1}$ for $\{i, j\} \in E$. This will not only guarantee $\rho > 0$ but these weights can easily be computed in a local fashion on each node (Xiao and Boyd, 2004).

Compression. We aim to only transmit *compressed* (e.g. quantized or sparsified) messages. We formalized this through the notion of compression operators that was e.g. also used in (Tang et al., 2018a; Stich et al., 2018).

Definition 4.4.1 (Compression operator). $Q: \mathbb{R}^d \rightarrow \mathbb{R}^d$ is a compression operator if it satisfies

$$\mathbb{E}_Q \|Q(\mathbf{x}) - \mathbf{x}\|^2 \leq (1 - \delta) \|\mathbf{x}\|^2, \quad \forall \mathbf{x} \in \mathbb{R}^d, \quad (4.2)$$

for a parameter $\delta > 0$. Here \mathbb{E}_Q denotes the expectation over the internal randomness of operator Q .

In contrast to the quantization operators used in e.g. Alistarh et al. (2017); Horváth et al. (2019), compression operators defined as in (4.2) are not required to be unbiased and therefore supports a larger class of compression operators. Some examples can be found in Koloskova et al. (2019) and we further discuss specific compression schemes in Section 4.6.

Algorithm. CHOCO-SGD is summarized in Algorithm 2.

Algorithm 2 CHOCO-SGD (Koloskova et al., 2019).

Requires: Initial value $\bar{\mathbf{x}}^{(-\frac{1}{2})} \in \mathbb{R}^d$; $\mathbf{x}_i^{(-\frac{1}{2})} = \bar{\mathbf{x}}^{(-\frac{1}{2})}$ on each node $i \in [n]$; consensus stepsize γ ; SGD stepsize η ; communication graph $G = ([n], E)$ and mixing matrix W ; initialize $\hat{\mathbf{x}}_i^{(0)} := \mathbf{0} \forall i \in [n]$.

```

1: procedure WORKER- $i$ 
2:   for  $t \in \{0, \dots, T-1\}$  do
3:      $\mathbf{x}_i^{(t)} := \mathbf{x}_i^{(t-\frac{1}{2})} + \gamma \sum_{j: \{i, j\} \in E} w_{ij} (\hat{\mathbf{x}}_j^{(t)} - \hat{\mathbf{x}}_i^{(t)})$             $\triangleright$  modified gossip averaging
4:      $\mathbf{q}_i^{(t)} := Q(\mathbf{x}_i^{(t)} - \hat{\mathbf{x}}_i^{(t)})$                                             $\triangleright$  compression
5:     for neighbors  $j: \{i, j\} \in E$  (including  $\{i\} \in E$ ) do
6:       Send  $\mathbf{q}_i^{(t)}$  and receive  $\mathbf{q}_j^{(t)}$                                         $\triangleright$  communication
7:        $\hat{\mathbf{x}}_j^{(t+1)} := \mathbf{q}_j^{(t)} + \hat{\mathbf{x}}_j^{(t)}$                                         $\triangleright$  local update
8:     Sample  $\xi_i^{(t)}$ , compute gradient  $\mathbf{g}_i^{(t)} := \nabla F_i(\mathbf{x}_i^{(t)}, \xi_i^{(t)})$ 
9:      $\mathbf{x}_i^{(t+\frac{1}{2})} := \mathbf{x}_i^{(t)} - \eta \mathbf{g}_i^{(t)}$                                         $\triangleright$  stochastic gradient update
10:  return  $\mathbf{x}^{(T-1+\frac{1}{2})}$ 

```

① {

② {

Every worker i stores its own private variable $\mathbf{x}_i \in \mathbb{R}^d$ that is updated by a stochastic gradient step in part ② and a modified gossip averaging step on line 2. This step is a key element of the algorithm as it preserves the averages of the iterates even in presence of quantization noise (the compression errors are not discarded, but aggregated in the local variables \mathbf{x}_i , see also Koloskova et al. (2019)). The nodes communicate with their neighbors in part ① and update the variables $\hat{\mathbf{x}}_j \in \mathbb{R}^d$ for all their neighbors $\{i, j\} \in E$ only using compressed updates. These $\hat{\mathbf{x}}_i$ are available to

all the neighbours of the node i and represent the ‘publicly available’ copies of the private \mathbf{x}_i , in general $\mathbf{x}_i \neq \hat{\mathbf{x}}_i$, due to the communication restrictions.

From an implementation aspect, it is worth highlighting that the communication part ① and the gradient computation part ② can both be executed in parallel because they are independent. Moreover, each node only needs to store 3 vectors at most, independent of the number of neighbors (this might not be obvious from the notation used here for additional clarity, for further details c.f. [Koloskova et al. \(2019\)](#)). We further propose a momentum-version of CHOCO-SGD in Algorithm 3 (see also Section 13.4 for further details).

4.5 Convergence of CHOCO-SGD on Smooth Non-Convex Problems

As the first main contribution, we here extend the analysis of CHOCO-SGD to non-convex problems. For this we make the following technical assumptions:

Assumption 5. Each function $f_i: \mathbb{R}^d \rightarrow \mathbb{R}$ for $i \in [n]$ is L -smooth, that is

$$\|\nabla f_i(\mathbf{y}) - \nabla f_i(\mathbf{x})\| \leq L \|\mathbf{y} - \mathbf{x}\|, \quad \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^d, i \in [n],$$

and the variance of the stochastic gradients is bounded on each worker:

$$\mathbb{E}_{\xi_i} \|\nabla F_i(\mathbf{x}, \xi_i) - \nabla f_i(\mathbf{x})\|^2 \leq \sigma_i^2, \quad \mathbb{E}_{\xi_i} \|\nabla F_i(\mathbf{x}, \xi_i)\|^2 \leq G^2, \quad \forall \mathbf{x} \in \mathbb{R}^d, i \in [n], \quad (4.3)$$

where $\mathbb{E}_{\xi_i}[\cdot]$ denotes the expectation over $\xi_i \sim \mathcal{D}_i$. We also denote $\bar{\sigma}^2 := \frac{1}{n} \sum_{i=1}^n \sigma_i^2$ for convenience.

Theorem 4.5.1. Under Assumptions 4–5 there exists a constant stepsize η and the consensus stepsize from [Koloskova et al. \(2019\)](#), $\gamma := \frac{\rho^2 \delta}{16\rho + \rho^2 + 4\beta^2 + 2\rho\beta^2 - 8\rho\delta}$ with $\beta = \|I - W\|_2 \in [0, 2]$, such that the averaged iterates $\bar{\mathbf{x}}^{(t)} := \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i^{(t)}$ of Algorithm 2 satisfy:

$$\frac{1}{T+1} \sum_{t=0}^T \left\| \nabla f(\bar{\mathbf{x}}^{(t)}) \right\|_2^2 = \mathcal{O} \left(\left(\frac{LF_0 \bar{\sigma}^2}{n(T+1)} \right)^{1/2} + \left(\frac{GLF_0}{c(T+1)} \right)^{2/3} + \frac{LF_0}{T+1} \right)$$

where $c := \frac{\rho^2 \delta}{82}$ denotes the convergence rate of the underlying consensus averaging scheme of [Koloskova et al. \(2019\)](#), $F_0 := f(\bar{\mathbf{x}}^{(0)}) - f^*$.

This result shows that CHOCO-SGD converges as $\mathcal{O}(1/\sqrt{nT} + 1/(\rho^2 \delta T)^{2/3})$. The first term shows a linear speed-up compared to SGD on a single node, while compression and graph topology affect only the higher order second term. By differently choosing the stepsize $\eta := \sqrt{n}/\sqrt{T+1}$ we can recover asymptotic convergence rate of $\mathcal{O}(1/\sqrt{nT} + n/(\rho^4 \delta^2 T))$. For the proofs and convergence of the individual iterates \mathbf{x}_i we refer to Appendix 13.1.

4.6 Comparison to Baselines for Various Compression Schemes

In this section we experimentally compare CHOCO-SGD to the relevant baselines for a selection of commonly used compression operators. For the experiments we further leverage momentum in all implemented algorithms. The newly developed momentum version of CHOCO-SGD is given as Algorithm 3.

Algorithm 3 CHOCO-SGD with Momentum.

Lines 3–8 in Algorithm 2 are left unmodified

Line 9 in Algorithm 2 is replaced with the following two lines

9: $\mathbf{v}_i^{(t+1)} := (\mathbf{g}_i^{(t)} + \lambda \mathbf{x}_i^{(t)}) + \beta \mathbf{v}_i^{(t)}$	▷ local momentum with weight decay
10: $\mathbf{x}_i^{(t+\frac{1}{2})} := \mathbf{x}_i^{(t)} - \eta \mathbf{v}_i^{(t+1)}$	▷ stochastic gradient update

Setup. In order to match the setting in Tang et al. (2018a) for our first set of experiments, we use a ring topology with $n = 8$ nodes and train the ResNet20 architecture (He et al., 2016a) on the Cifar10 dataset (50K/10K training/test samples) (Krizhevsky and Hinton, 2009). We randomly split the training data between workers and shuffle it after every epoch, following standard procedure as e.g. in Goyal et al. (2017). We implement DCD and ECD with momentum (Tang et al., 2018a), DeepSqueeze with momentum (Tang et al., 2019), CHOCO-SGD with momentum (Algorithm 3) and standard (all-reduce) mini-batch SGD with momentum and without compression (Dekel et al., 2012). Our implementations are open-source and available at <https://github.com/epfml/ChocoSGD>. The momentum factor is set to 0.9 without dampening. For all algorithms we fine-tune the initial learning rate and gradually warm it up from a relative small value (0.1) (Goyal et al., 2017) for the first 5 epochs. The learning rate is decayed by 10 twice, at 150 and 225 epochs, and stop training at 300 epochs. For CHOCO-SGD and DeepSqueeze the consensus learning rate γ is also tuned. The detailed hyperparameter tuning procedure refers to Appendix 13.6. Every compression scheme is applied to every layer of ResNet20 separately. We evaluate the top-1 test accuracy on every node separately over the whole dataset and report the average performance over all nodes.

Compression Schemes. We implement two *unbiased* compression schemes: (i) gsgd_b quantization that randomly rounds the weights to b -bit representations (Alistarh et al., 2017), and (ii) random_a sparsification, which preserves a randomly chosen a fraction of the weights and sets the other ones to zero (Wangni et al., 2018). Further two *biased* compression schemes: (iii) top_a , which selects the a fraction of weights with the largest magnitude and sets the other ones to zero (Alistarh et al., 2018; Stich et al., 2018), and (iv) sign compression, which compresses each weight to its sign scaled by the norm of the full vector (Bernstein et al., 2018; Karimireddy et al., 2019). We refer to Appendix 13.3 for exact definitions of the schemes.

DCD and ECD have been analyzed only for unbiased quantization schemes, thus the combination with the two biased schemes is not supported by theory. In converse, CHOCO-SGD and

Chapter 4. Decentralized Deep Learning with Arbitrary Communication Compression

Table 4.1 – Top-1 test accuracy for decentralized DCD, ECD, DeepSqueeze, and CHOCO-SGD with various compression schemes. Reported top-1 test accuracies are averaged over three runs with fine-tuned hyperparameters (learning rate, weight decay, consensus stepsize). The fine-tuned all-reduce baseline reaches accuracy 92.64, with 1.04 MB gradient transmission per iteration. (★ indicates that 2 out of 3 runs diverged).

Algorithm	Error-feedback	Quantization (QSGD)				Sparsification (random-%)		
		16 bits	8 bits	4 bits	2 bits	50%	10%	1%
transmitted data/iteration		0.52 MB	0.26 MB	0.13 MB	0.065 MB	1.04 MB	0.21 MB	0.031 MB
DCD-PSGD	✗	92.51 ± 0.05	92.36 ± 0.28	23.56 ± 2.97	diverges	92.05 ± 0.25	diverges	diverges
ECD-PSGD	✗	92.02 ± 0.14	59.11 ± 1.57	diverges	diverges	diverges	diverges	diverges
DeepSqueeze	✓	92.27 ± 0.21	91.83 ± 0.35	91.47 ± 0.21	90.96 ± 0.19	91.46 ± 0.09	90.96 ± 0.16	88.55 ± 0.11
CHOCO-SGD	✓	92.34 ± 0.19	92.30 ± 0.08	91.92 ± 0.27	91.41 ± 0.11	92.54 ± 0.26	91.87 ± 0.21	91.32 ± 0.17

Algorithm	Error-feedback	Sparsification (top-%)			Sign+Norm
		50%	10%	1%	-
transmitted data/iteration		1.04 MB	0.21 MB	0.031 MB	0.032 MB
DCD-PSGD	✗	92.40 ± 0.11	91.97 ± 0.14	89.79 ± 0.40	92.40 ± 0.14
ECD-PSGD	✗	17.03 ★	16.78 ★	18.03 ★	diverges
DeepSqueeze	✓	91.55 ± 0.28	91.31 ± 0.25	90.47 ± 0.17	91.38 ± 0.19
CHOCO-SGD	✓	92.54 ± 0.26	92.29 ± 0.05	91.73 ± 0.11	92.46 ± 0.10

DeepSqueeze has been studied only for biased schemes according to Definition 4.2. However, both unbiased compression schemes can be scaled down in order to meet the specification (cf. discussions in [Stich et al. \(2018\)](#); [Koloskova et al. \(2019\)](#)) and we adopt this for the experiments.

Results. The results are summarized in Table 4.1. For unbiased compression schemes, ECD and DCD only achieve good performance when the compression ratio is small, and sometimes even diverge when the compression ratio is high. This is consistent¹ with the theoretical and experimental results in [Tang et al. \(2018a\)](#). We further observe that the performance of DCD with the biased top_a sparsification is much better than with the unbiased random_a counterpart, though this operator is not yet supported by theory.

CHOCO-SGD can generalize reasonably well in all scenarios (at most 1.65% accuracy drop) for fixed training budget. The sign compression achieves state-of-the-art accuracy and requires approximately 32× less bits per weight than the full precision baseline.

4.7 Use Case I: On-Device Peer-to-Peer Learning

We now shift our focus to challenging real-world scenarios which are intrinsically decentralized, i.e. each part of the training data remains local to each device, and thus centralized methods either fail or are inefficient to implement. Typical scenarios comprise e.g. sensor networks, or mobile devices or hospitals which jointly train a machine learning model. Common to these applications is that i) each device has only access to locally stored or acquired data, ii) communication

¹ [Tang et al. \(2018a\)](#) only consider absolute bounds on the quantization error. Such bounds might be restrictive (i.e. allowing only for low compression) when the input vectors are unbounded. This might be the reason for the instabilities observed here and also in ([Tang et al., 2018a](#), Figure 4), ([Koloskova et al., 2019](#), Figures. 5–6).

4.7. Use Case I: On-Device Peer-to-Peer Learning

Table 4.2 – Summary of communication topologies.

Topology	max. node degree	spectral gap ρ			
		$n = 4$	$n = 16$	$n = 36$	$n = 64$
ring	2	0.67	0.05	0.01	0.003
torus	4	0.67	0.4	0.2	0.12
fully-connected	d	1	1	1	1

bandwidth is limited (either physically, or artificially for e.g. metered connections), iii) the global network topology is typically unknown to a single device, and iv) the number of connected devices is typically large. Additionally, this fully decentralized setting is also strongly motivated by privacy aspects, enabling to keep the training data private on each device at all times.

Modeling. To simulate this scenario, we permanently split the training data between the nodes, i.e. the data is never shuffled between workers during training, and every node has distinct part of the dataset. To the best of our knowledge, no prior works studied this scenario for decentralized deep learning. For the centralized approach, gathering methods such as all-reduce are not efficiently implementable in this setting, hence we compare to the centralized baseline where all nodes route their updates to a central coordinator for aggregation. For the comparison we consider CHOCO-SGD with sign compression (this combination achieved the compromise between accuracy and compression level in Table 4.1)), decentralized SGD without compression (Lian et al., 2017), and centralized SGD without compression.

Scaling to Large Number of Nodes. To study the scaling properties of CHOCO-SGD, we examine the quality of decentralized training with compressed communication on various number of nodes, namely $\{4, 16, 36, 64\}$. We consider decentralized algorithms on two different topologies: (1) *ring*, which is the worst possible topology, and (2) *torus*, the one with much larger spectral gap. A summary of communication topologies are listed in Table 4.2. We train ResNet8 (He et al., 2016a) (78K parameters) on Cifar10 dataset (50K/10K training/test samples) (Krizhevsky and Hinton, 2009). For the sake of simplicity, we keep the learning rate constant and tune it individually for each method. We tune the consensus learning rate for CHOCO-SGD.

The results are summarized in Figure 4.1 (and Figure 13.1, Table 13.4–13.5 in Appendix 13.7). We begin by comparing the testing accuracy reached after 300 epochs (Figure 4.1, *Left*). CentralizedSGD has an excellent performance (over other decentralized methods) for varying numbers of nodes evaluated. The convergence of CHOCO-SGD experiences a slowing down, due to the influence of graph topology (*Decentralized curve*) and the communication ρ compression (*CHOCO curve*). The former observations on graph topology are consistent with the spectral gaps order (see Table 4.2), while the latter issue caused by communication compression applied to both ring and torus. It is worth noting that the training performance matches the test curves

Chapter 4. Decentralized Deep Learning with Arbitrary Communication Compression

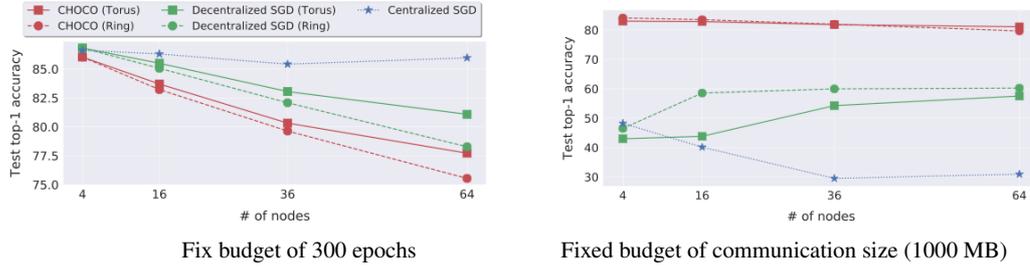


Figure 4.1 – Scaling of CHOCO-SGD with sign compression to large number of devices on Cifar10 dataset. *Left*: best testing accuracy of the algorithms reached after 300 epochs. *Right*: best testing accuracy reached after communicating 1000 MB.

in Figure 4.1, implying that the source of performance degradation is a hindered convergence (Theorem 4.5.1) rather than a potential generalization issue. The performance of decentralized schemes may improve as the number of epochs is increased; however, even with 10 times more epochs, we were unable to completely bridge the gap between centralized and decentralized algorithms for both train and test performance.

In the realistic decentralized scenario, the goal is to reduce the amount of communication rather than the number of epochs to lower the cost of the user’s mobile data. We therefore fix the number of total transmitted bits to 1000 MB and compare the best test accuracy reached (Figure 4.1, *Right*). CHOCO-SGD outperforms other decentralized schemes and only suffers a slight degradation as the number of nodes grows. When the number of nodes is large, the torus topology is beneficial because of its good mixing properties; however, there is little difference between these two topologies for small networks—the benefit of a large spectral gap is offset by the increased communication due to the larger node degree in torus topology.

Experiments on a Real Social Network Graph. We simulate training models on user devices (e.g. mobile phones), connected by a real social network. We use Davis Southern women social network (Davis et al., 1941) with 32 nodes. We train ResNet20 (0.27 million parameters) model on the Cifar10 dataset (50K/10K training/test samples) (Krizhevsky and Hinton, 2009) for image classification and a three-layer LSTM architecture (Hochreiter and Schmidhuber, 1997b) (28.95 million parameters) for a language modeling task on WikiText-2 (600 training and 60 validation articles with a total of 2’088’628 and 217’646 tokens respectively) (Merity et al., 2017). The depicted curves of the training loss are the averaged local loss over all workers (local model with fixed local data); the test performance uses the mean of the evaluations for local models on whole test dataset. For more detailed experimental setup we refer to Appendix 13.6.

The results are summarized in Figures 4.2–4.3 and in Table 4.3. For the image classification task, when comparing the training accuracy reached after the same number of epochs, we observe that the decentralized algorithm performs best, follows by the centralized and lastly the quantized decentralized. However, the test accuracy is highest for the centralized scheme. When comparing

4.7. Use Case I: On-Device Peer-to-Peer Learning

Table 4.3 – Summary of performance when training with the same epoch budget (as centralized SGD).

Algorithm	max. connections/node	ResNet-20 (Figure 4.2)		LSTM (Figure 4.3)	
		data/gradient	top-1 test acc.	data/gradient	test perplexity
Centralized SGD	32	1.04 MB	93.00	110.43 MB	89.39
Exact Decentralized SGD	14	1.04 MB	92.12	110.43 MB	91.38
CHOCO-SGD (Sign + Norm)	14	0.032 MB	91.80	3.45 MB	86.58

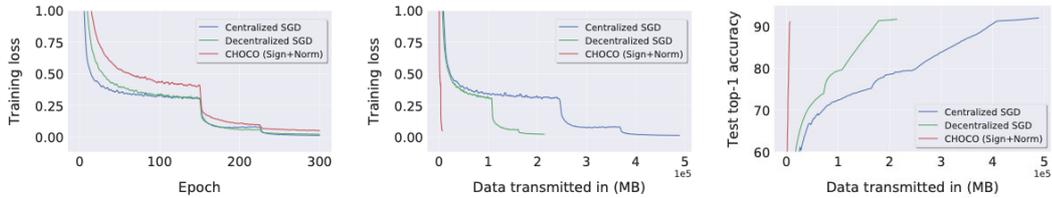


Figure 4.2 – Image classification: ResNet-20 on CIFAR-10 on social network topology.

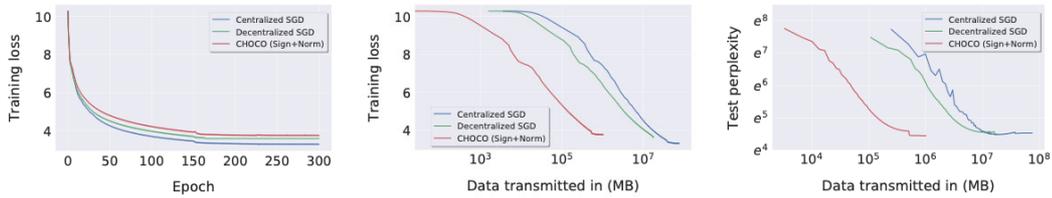


Figure 4.3 – Language modeling: LSTM on WikiText-2 on social network topology.

the test accuracy reached for the same transmitted data², CHOCO-SGD significantly outperforms the exact decentralized scheme, with the centralized performing worst. We note a slight accuracy drop, i.e. after the same number of epochs (but much less transmitted data), CHOCO-SGD does not reach the same level of test accuracy than the baselines.

For the language modeling task, both decentralized schemes suffer a drop in the training loss when the evaluation reaching the epoch budget; while our CHOCO-SGD outperforms the centralized SGD in test perplexity. When considering perplexity for a fixed data volume (middle and right subfigure of Figure 4.3), CHOCO-SGD performs best, followed by the exact decentralized and centralized algorithms.

In Figure 4.4 we additionally depict the test accuracy of the averaged model $\bar{\mathbf{x}}^{(t)} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i^{(t)}$ (left) and averaged distance of the local models from the averaged model (right), for CHOCO-SGD on image classification task. Towards the end of the optimization, the local models reach consensus (Figure 4.4, right), and their individual test performances are the same as performance of averaged model. Interestingly, before decreasing the stepsize at the epoch 225, the local models are in general diverging from the averaged model, while decreasing only when the stepsize decreases. The same behavior was also reported in Assran et al. (2019).

² The figure reports the transmitted data on the busiest node, i.e. on the max-degree node (degree 14) node for decentralized schemes, and degree 32 for the centralized one.

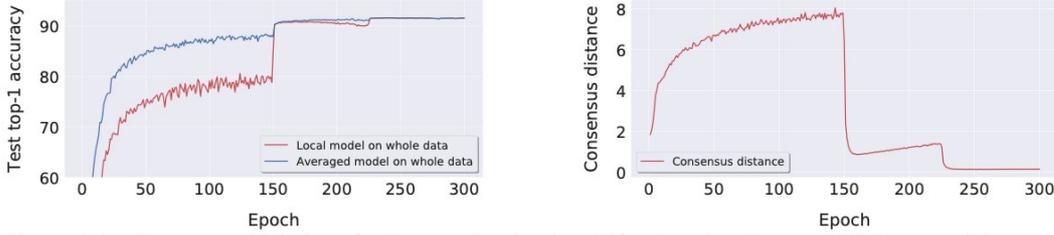


Figure 4.4 – Parameter deviations for Resnet20 trained on Cifar10 (using CHOCO-SGD) on social network topology (32 workers). (Left) performance of the averaged model compared to the average of performances of local models. (Right) parameters divergence: averaged L_2 consensus distance between local models \mathbf{x}_i and the averaged model $\bar{\mathbf{x}} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i$, i.e. $\frac{1}{n} \sum_{i=1}^n \|\mathbf{x}_i - \bar{\mathbf{x}}\|_2^2$.

4.8 Use Case II: Efficient Large-Scale Training in a Datacenter

Decentralized optimization methods offer a way to address scaling issues even for well connected devices, such as e.g. in datacenter with fast InfiniBand (100Gbps) or Ethernet (10Gbps) connections. Lian et al. (2017) describe scenarios when decentralized schemes can outperform centralized ones, and recently, Assran et al. (2019) presented impressive speedups for training on 256 GPUs, for the setting when all nodes can access all training data. The main differences of their algorithm to CHOCO-SGD are the asynchronous gossip updates, time-varying communication topology and most importantly exact communication, making their setup not directly comparable to ours. We note that these properties of asynchronous communication and changing topology for faster mixing are orthogonal to our contribution, and offer promise to be combined.

Setup. We train ImageNet-1k (1.28M/50K training/validation) (Deng et al., 2009) with Resnet-50 (He et al., 2016a). We perform our experiments on 8 machines (n1-standard-32 from Google Cloud with Intel Ivy Bridge CPU platform), where each of machines has 4 Tesla P100 GPUs and each machine interconnected via 10Gbps Ethernet. Within one machine communication is fast and we rely on the local data parallelism to aggregate the gradients for the later gradients communication (over the machines). Between different machines we consider centralized (fully connected topology) and decentralized (ring topology) communication, with and without compressed communication (sign compression). Several methods categorized by communication schemes are evaluated: (i) centralized SGD (full-precision communication), (ii) error-feedback centralized SGD with compressed communications (Karimireddy et al., 2019) through sign compression, (iii) decentralized SGD (Lian et al., 2017) with parallelized forward pass and gradients communication (full-precision communication), and (iv) CHOCO-SGD with sign compressed communications. The mini-batch size on each GPU is 128, and we follow the general SGD training scheme in Goyal et al. (2017) and directly use all their hyperparameters for all evaluated methods. Due to the limitation of the computational resource, we did not heavily tune the consensus stepsize for CHOCO-SGD.³

³ We estimate the consensus stepsize by running CHOCO-SGD with various values for the first 3 epochs.

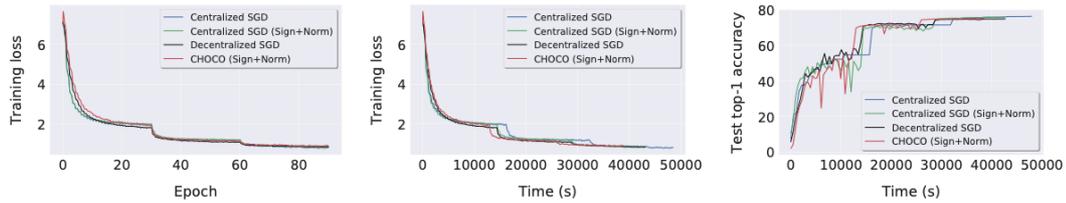


Figure 4.5 – Large-scale training: Resnet-50 on ImageNet-1k in the datacenter setting. The topology has 8 nodes (each accesses 4 GPUs). We use sign as the compression scheme, for CHOCO-SGD and Centralized SGD. For centralized SGD baseline without compression, we use all-reduce to aggregate the gradients; we use all-gather for centralized SGD with sign gradients quantization. The benefits of CHOCO-SGD can be further pronounced when scaling to more nodes.

Results. We depict the training loss and top-1 test accuracy in terms of epochs and time in Figure 4.5. CHOCO-SGD benefits from its decentralized and parallel structure and takes less time than all-reduce to perform the same number of epochs, while having only a slight 1.5% accuracy loss.⁴ In terms of time per epoch, our speedup does not match that of Assran et al. (2019), as the used hardware and the communication pattern⁵ are very different. Their scheme is orthogonal to our approach and could be integrated for better training efficiency. Nevertheless, we still demonstrate a time-wise 20% gain over the common all-reduce baseline, on our used commodity hardware cluster.

4.9 Conclusion

We propose the use of CHOCO-SGD (and its momentum version) for enabling decentralized deep learning training in bandwidth-constrained environments. We provide theoretical convergence guarantees for the non-convex setting and show that the algorithm enjoys linear speedup in the number of nodes. We empirically study the performance of the algorithm in a variety of settings on the image classification (ImageNet-1k, Cifar10) and on the language modeling task (WikiText-2). Whilst previous work successfully demonstrated that decentralized methods can be a competitive alternative to centralized training schemes when no communication constraints are present (Lian et al., 2017; Assran et al., 2019), our main contribution is to enable training in strongly communication-restricted environments, and while respecting the challenging constraint of locality of the training data. We theoretically and practically demonstrate the performance of decentralized schemes for arbitrary high communication compression, and under data-locality, and thus significantly expand the reach of potential applications of fully decentralized deep learning.

⁴ Centralized SGD with full precision gradients achieved test accuracy of 76.37%, v.s. 76.03% for centralized SGD (with sign compression), v.s. 74.92% for plain decentralized SGD, and v.s. 75.15% for CHOCO-SGD (with sign compression).

⁵ We consider undirected communication, contrary to the directed 1-peer communication (every node sends and receives one message at every iteration) in Assran et al. (2019).

5 Consensus Control for Decentralized Deep Learning

5.1 Preface

Contribution and sources. This chapter builds on the work done in [Kong et al. \(2021a\)](#). The author proposed the initial idea and led the whole project; the author and Lingjing Kong conducted most of the experiments and identified the key insights. The theoretical insights were derived under the help of Anastasia Koloskova and Sebastian U. Stich. Detailed individual contributions: *Tao Lin (author)*: Conceptualization, Software & Experiments (50 %), Empirical analysis (50 %), Writing—original draft preparation (45 %).

Lingjing Kong: Software & Experiments (50 %), Empirical analysis (50 %), Writing—original draft preparation (45 %).

Anastasia Koloskova: Theoretical analysis (80 %), Writing—original draft preparation (10 %).

Martin Jaggi: Supervision, Writing—review and editing.

Sebastian U. Stich: Theoretical analysis (20 %), Supervision, Administration, Writing—review and editing.

Summary. Decentralized training of deep learning models enables on-device learning over networks, as well as efficient scaling to large compute clusters. Experiments in earlier works reveal that, even in a data-center setup, decentralized training often suffers from the degradation in the quality of the model: the training and test performance of models trained in a decentralized fashion is in general worse than that of models trained in a centralized fashion, and this performance drop is impacted by parameters such as network size, communication topology and data partitioning.

In this chapter, we identify the changing consensus distance between devices as a key parameter to explain the gap between centralized and decentralized training. We show in theory that when the training consensus distance is lower than a critical quantity, decentralized training converges as fast as the centralized counterpart. We empirically validate that the relation between

generalization performance and consensus distance is consistent with this theoretical observation. Our empirical insights allow the principled design of better decentralized training schemes that mitigate the performance drop. To this end, we provide practical training guidelines and exemplify its effectiveness on the data-center setup as the important first step.

5.2 Introduction

The impressive successes of machine learning, witnessed in the last decade, have been accompanied by a steady increase in the size, complexity, and computational requirements of training systems. In response to these challenges, distributed training algorithms (i.e. data-parallel large mini-batch SGD) have been developed for the use in data-centers (Goyal et al., 2017; You et al., 2018; Shallue et al., 2019). These state-of-the-art (SOTA) training systems rely on the All-Reduce communication primitive to perform exact averaging on the local mini-batch gradients computed on subsets of the data, for the later synchronized model update. However, exact averaging with All-Reduce is sensitive to the communication hardware of the training system, causing the bottleneck in efficient deep learning training. To address this issue, decentralized training has become an indispensable training paradigm for efficient large scale training in data-centers (Assran et al., 2019), alongside its orthogonal benefits on preserving users’ privacy for edge AI (Bellet et al., 2018; Kairouz et al., 2021).

Decentralized SGD (D-SGD) implementations trade off the exactness of the averaging provided by All-Reduce, with more efficient, but inexact, communication over sparse topologies. However, this often results in a severe drop in the training and/or test performance (i.e. generalization gap), even after hyperparameter fine-tuning (see our Table 5.1 as well as Tables 1–3 in Assran et al., 2019). This phenomenon is poorly understood even in relatively straightforward i.i.d. data distribution scenarios (i.e. the data-center case), to which very few works are dedicated (in fact none of them provide insights into the generalization performance).

In this work, we investigate the trade-off between the train/test performance and the exactness of the averaging, measured in terms of consensus distance, i.e. the average discrepancy between each node and the mean of model parameters over all machines. We identify this consensus distance as the key parameter that captures the joint effect of decentralization.

Table 5.1 – **Significant generalization gap for decentralized training** on a sparse ring topology (ResNet-20 on CIFAR-10 with $n \in \{16, 32, 64\}$ workers). Decentralized SGD (D-SGD) communicates model parameters through the gossip averaging. Test top-1 accuracies averaged over three seeds with fine-tuned learning rates.

	AllReduce (complete)	D-SGD (ring)
n=16	92.91 ± 0.12	92.40 ± 0.10
n=32	92.82 ± 0.27	91.81 ± 0.09
n=64	92.71 ± 0.11	89.58 ± 0.20

While one might suspect that a smaller consensus distance would improve performance in any case, we identify several interesting phenomena. (i) We identify a *diminishing return* phenomenon: if the consensus distance stays below a critical value (critical consensus distance), decreasing the consensus distance further does not yield any additional performance gains. For the main interests of this work, deep learning training, we (ii) identify the pivotal initial training phase where the critical consensus distance matters and the training consensus distance heavily influences the final training and generalization performance, and (iii) large consensus distance in later training phases can even be beneficial.

Our findings have far-reaching consequences for practice: By (iv) using consensus control as a principled tool to find, adaptively during training, the appropriate trade-off between targeted generalization performance and affordable communication resources, it is possible to exploit the efficiency benefits of decentralized methods without sacrificing quality. While our numerical study, on Computer Vision (CV) tasks (CIFAR-10 and ImageNet-32) as well as Natural Language Processing (NLP) tasks (transformer models for machine translation), mainly focuses on the data-center setting with homogeneous nodes, our findings also apply to decentralized training over time-varying topologies and the more difficult heterogeneous setting alike.

5.3 Related Work

5.3.1 Decentralized Learning

For general decentralized optimization, common algorithms are either gradient-based methods with gossip averaging steps (Kempe et al., 2003; Xiao and Boyd, 2004; Boyd et al., 2006), or problem-structure dependent methods, such as primal-dual methods (Hong et al., 2017; Sun and Hong, 2019). In this work, we focus on non-convex decentralized deep learning problems and only consider gradient-based methods with gossip averaging—methods that do not support stochastic gradients (not suitable for deep learning) are omitted for the discussion.

The convergence rate of gossip averaging towards the consensus among the nodes can be expressed in terms of the (expected) spectral gap of the mixing matrix. Lian et al. (2017) combine SGD with gossip averaging for deep learning and show that the leading term in the convergence rate $\mathcal{O}(\frac{1}{n\epsilon^2})$ is consistent with the convergence of the centralized mini-batch SGD (Dekel et al., 2012) and the spectral gap only affects the asymptotically smaller terms. Similar results have been observed very recently for related schemes (Scaman et al., 2017, 2018; Koloskova et al., 2019, 2020a,b; Vogels et al., 2020). To reduce the communication overhead (number of peer-to-peer communications), sparse topologies have been studied recently (Assran et al., 2019; Wang et al., 2019b, 2020c; Nadiradze et al., 2021). Whilst a few recent works focus on the impact of the topology on the optimization performance (Luo et al., 2019b; Neglia et al., 2020), we here identify the consensus distance as a more canonical parameter that can characterize the overall effect of decentralized learning, beyond only the topology. Through this, we are able to provide deeper understanding of the more fine-grained impact of the evolution of the actual consensus

distance on the optimization/generalization performance of deep learning.

Prior works propose to either perform a constant number of gossip steps every round (Tsianos and Rabbat, 2016; Scaman et al., 2017; Jiang et al., 2017, 2021; Sharma et al., 2019) to increase the averaging quality, or choose carefully tuned learning rates (Tsitsiklis, 1984; Nedic and Ozdaglar, 2009; Duchi et al., 2011; Yuan et al., 2016) to improve the convergence. However, these works do not analyze the varying effect of consensus distance in the phases of training explicitly. In contrast, we identify the existence of a *critical* consensus distance, *adapt* gossip step numbers to the target distance on the fly, and provide insights into how consensus distance at various training phases impacts the decentralized deep learning.

Appendix 14.2.1 further details the relationship between consensus distance and other training metrics influential to the final performance (e.g. gradient diversity in (Yin et al., 2018; Johnson et al., 2020)). Besides, we connect the insights into better generalization (Lin et al., 2020d) (as in Chapter 2) with other interpretations in Izmailov et al. (2018); Gupta et al. (2020).

5.3.2 Critical Learning Phase in Deep Learning

The connection between optimization and generalization of deep learning training is not fully understood. A line of work on understanding the early phase of training has recently emerged as a promising avenue for studying this connection. For instance, Keskar et al. (2017); Sagun et al. (2018); Achille et al. (2019); Golatkar et al. (2019); Frankle et al. (2020) point out the existence of a “critical phase” for regularizing deep networks, which is decisive for the final generalization ability. Achille et al. (2019); Jastrzebski et al. (2019); Fort and Ganguli (2019); Jastrzebski et al. (2020) empirically demonstrate the rapid change in the local shape of the loss surface in the initial training phase.

In this work, we reach a similar conclusion for decentralized deep learning: we identify the importance of the initial training phase through the lens of consensus distance.

5.4 Theoretical Understanding

In this section, we study the trade-off between training performance and the exactness of parameter averaging, and establish the notion of critical consensus distance.

For the sake of simplicity, we consider decentralized stochastic gradient descent (D-SGD) without momentum in this section, and focus on the optimization difficulty in our theoretical analysis. Theoretically analyzing the generalization performance for deep learning is an open problem and not intended in this work. Instead we provide extensive empirical evaluation, addressing generalization for both D-SGD with and without momentum in Section 5.5. All proofs are deferred to Appendix 14.3.

5.4.1 Notation and Setting

The agents are tasked to solve a sum-structured optimization problem $f: \mathbb{R}^d \rightarrow \mathbb{R}$ of the form

$$f^* := \min_{\mathbf{x} \in \mathbb{R}^d} [f(\mathbf{x}) := \frac{1}{n} \sum_{i=1}^n f_i(\mathbf{x})], \quad (5.1)$$

where the components $f_i: \mathbb{R}^d \rightarrow \mathbb{R}$ are distributed among the n nodes and are given in stochastic form: $f_i(\mathbf{x}) := \mathbb{E}_{\xi \sim \mathcal{D}_i} [F_i(\mathbf{x}, \xi)]$, where \mathcal{D}_i denotes the local data distribution on node $i \in [n]$. For data-center settings, where data is re-shuffled periodically among nodes, these distributions are identical, but in other scenarios there can be differences between nodes. In D-SGD, each agent $i \in [n]$ maintains local parameters $\mathbf{x}_i^{(t)} \in \mathbb{R}^d$, and updates them as:

$$\mathbf{x}_i^{(t+1)} = \sum_{j=1}^n w_{ij} \left(\mathbf{x}_j^{(t)} - \eta \nabla F_j(\mathbf{x}_j^{(t)}, \xi_j^{(t)}) \right), \quad (\text{D-SGD})$$

that is, by a stochastic gradient step based on a sample $\xi_i^{(t)} \sim \mathcal{D}_i$, followed by gossip averaging with neighboring nodes in the network encoded by the mixing weights w_{ij} . As parameters can vary across nodes, we define $\bar{\mathbf{x}} := \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i$ and $\mathbf{X} := [\mathbf{x}_1, \dots, \mathbf{x}_n] \in \mathbb{R}^{d \times n}$, and $\bar{\mathbf{X}} := [\bar{\mathbf{x}}, \dots, \bar{\mathbf{x}}] \equiv \mathbf{X} \frac{1}{n} \mathbf{1} \mathbf{1}^\top$.

Assumption 6 (Mixing matrix). *Every sample of the (possibly randomized) mixing matrix $\mathbf{W} = \{w_{ij}\} \in \mathbb{R}^{n \times n}$ is doubly stochastic and there exists a parameter $p > 0$ s.t.*

$$\mathbb{E}_{\mathbf{W}} \|\mathbf{X}\mathbf{W} - \bar{\mathbf{X}}\|_F^2 \leq (1-p) \|\mathbf{X} - \bar{\mathbf{X}}\|_F^2, \forall \mathbf{X} \in \mathbb{R}^{d \times n}. \quad (5.2)$$

This assumption covers a broad variety of settings (see e.g. [Koloskova et al., 2020b](#)), such as D-SGD with fixed (constant) mixing matrix with spectral gap ρ , with parameter $p = 1 - (1 - \rho)^2 = \Theta(\rho)$, but also for randomly chosen mixing matrices, for instance random matchings.

Assumption 7 (L -smoothness). *Each function $f_i(\mathbf{x}): \mathbb{R}^d \rightarrow \mathbb{R}$, $i \in [n]$ is differentiable and there exists a constant $L \geq 0$ such that for each $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$: $\|\nabla f_i(\mathbf{x}) - \nabla f_i(\mathbf{y})\| \leq L \|\mathbf{x} - \mathbf{y}\|$.*

Assumption 8 (Bounded noise σ and diversity ζ). *There exists constants σ^2, ζ^2 s.t. $\forall \mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^d$*

$$\begin{aligned} \frac{1}{n} \sum_{i=1}^n \mathbb{E}_{\xi_i} \|\nabla F_i(\mathbf{x}_i, \xi_i) - \nabla f_i(\mathbf{x}_i)\|_2^2 &\leq \sigma^2, \\ \frac{1}{n} \sum_{i=1}^n \|\nabla f_i(\mathbf{x}_i) - \nabla f(\mathbf{x}_i)\|_2^2 &\leq \zeta^2. \end{aligned} \quad (5.3)$$

5.4.2 Decentralized Consensus Optimization

Under the above standard assumptions in decentralized optimization, the convergence rate of (D-SGD) has been shown as follows:

Theorem 5.4.1 ([Koloskova et al. \(2020b\)](#)). *Let f_i be L -smooth and stepsize $\eta \leq \eta_{\max} = \mathcal{O}(\frac{p}{L})$.*

Chapter 5. Consensus Control for Decentralized Deep Learning

Then there exists an optimal stepsize $\eta \leq \eta_{\max}$ such that $\frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E} \|\nabla f(\bar{\mathbf{x}}^{(t)})\|_2^2 \leq \varepsilon$ for

$$T = \mathcal{O} \left(\frac{\sigma^2}{n\varepsilon^2} + \frac{\sqrt{p}\sigma + \zeta}{p\varepsilon^{3/2}} + \frac{1}{p\varepsilon} \right) \cdot L(f(\mathbf{x}_0) - f^*).$$

In comparison, for centralized mini-batch SGD (C-SGD) we are allowed to choose a potentially much larger stepsize $\eta'_{\max} = \mathcal{O}(\frac{1}{L})$, and can bound the number of iterations by $\mathcal{O}(\frac{\sigma^2}{n\varepsilon^2} + \frac{1}{\varepsilon})$. While asymptotically both these rates are equivalent, they vary in the low accuracy setting when ε is not too small. That is, especially in the first phase of optimization where the lower order terms matter (Bottou and Bousquet, 2007; Stich et al., 2021).

As our first theoretical contribution, we show that if the individual iterates of the agents stay sufficiently close, then D-SGD can converge as fast as C-SGD. To measure this difference between agents, we use the *consensus distance*

$$\Xi_t^2 := \frac{1}{n} \sum_{i=1}^n \|\bar{\mathbf{x}}^{(t)} - \mathbf{x}_i^{(t)}\|^2.$$

Proposition 5.4.2 (Critical Consensus Distance (CCD)). *If the consensus distance is bounded by*

$$\Xi_t^2 \leq \left(\frac{1}{Ln} \eta \sigma^2 + \frac{1}{8L^2} \|\nabla f(\bar{\mathbf{x}}^{(t)})\|^2 =: \eta_t^2 \right) \quad (5.4)$$

for all t , then in D-SGD we may choose larger stepsizes $\eta \leq \eta'_{\max} = \mathcal{O}(\frac{1}{L})$ and recover the convergence rate of C-SGD, that is $\mathcal{O}(\frac{\sigma^2}{n\varepsilon^2} + \frac{1}{\varepsilon})$ (Dekel et al., 2012; Bottou et al., 2018). We refer to η_t^2 as critical consensus distance (CCD).

Note that the CCD does not depend on the graph topology and that $\eta_t^2 > 0$, which means that we do not need perfect consensus between agents to recover the C-SGD rate, but we allow consensus distance $\Xi_t^2 \geq 0$ (i.e. the $\Xi_t^2 = 0 \forall t$, as we have for centralized optimization, is sufficient but not necessary). In Section 5.5, we empirically examine the existence of the critical consensus distance Ξ_t^2 in decentralized deep learning, as we cannot compute the critical consensus distance in a closed-form (through L and σ^2).

We now estimate the magnitude of the consensus distance in D-SGD and compare it to the CCD.

Proposition 5.4.3 (Typical consensus distance). *Let $\phi_t^2 := \frac{1}{n} \sum_{i=1}^n \|\nabla f_i(\mathbf{x}_i^{(t)})\|^2$. Then under the assumption that η, p are constant, and the ϕ_t does not change too fast between iterations, i.e. not decreasing faster than exponentially: $\phi_t^2 \leq (1 + p/4)\phi_{t+1}^2$, the consensus distance in D-SGD satisfies*

$$\Xi_t^2 = (1 - p)\eta^2 \cdot \mathcal{O} \left(\frac{\phi_t^2}{p^2} + \frac{\sigma^2}{p} \right). \quad (5.5)$$

While these assumptions do not hold in epochs with learning rate decay, we observe in practice

5.5. Inspecting Consensus Distance for Decentralized Training

that during epochs of a constant learning rate the gradients indeed do not change too fast (see Figure 14.2(b)). Thus these assumptions are reasonable approximations to capture the practical behavior.

5.4.3 Controlling the Consensus Distance

We now investigate scenarios where the typical consensus distance derived in Proposition 5.4.3 *can* be smaller than the critical value (CCD). This reveals two orthogonal strategies to control the consensus distance in D-SGD. We here assume diversity $\zeta = 0$ as with i.i.d. training data, and that the stepsize $\eta \leq \mathcal{O}(\frac{1}{L})$ as for C-SGD, and give a more refined discussion in Appendix 14.3.3.

Learning rate decay (changing η). We observe that when $\eta = \mathcal{O}(\frac{p}{nL})$ then $\Xi_t^2 \leq \eta_t^2$ (if the noise σ is small, especially for $\sigma = 0$, then the weaker assumption $\eta = \mathcal{O}(\frac{p}{L})$ is sufficient). However, choosing too small stepsizes can impact performance in practice. In C-SGD the constraint on the stepsize is loose ($\eta \leq \frac{1}{L}$). Yet, after sufficient learning rate decay, the desired CCD can be reached.

More gossip iterations (changing p). We observe that when $\frac{1}{1-p} = \mathcal{O}(1 + \eta Ln)$, then $\Xi_t^2 \leq \eta_t^2$ (again, when the noise σ is small, especially when $\sigma^2 = 0$, a weaker condition $\frac{1}{1-p} = \mathcal{O}(1 + \eta L)$ is sufficient). Whilst designing new mixing topologies to control p might not be possible due to practical constraints (fixed network, denser graphs increase latency, etc.), a simple and commonly used strategy is to use repeated gossip steps in every round.

Lemma 5.4.4 (Repeated gossip (Xiao and Boyd, 2004; Boyd et al., 2006)). *Suppose $\mathbf{W} = \mathbf{W}_k \dots \mathbf{W}_1$, for k (possibly randomized) mixing matrices with parameter p each. Then the mixing parameter for \mathbf{W} is at least $p_{\mathbf{W}} \geq 1 - (1 - p)^k$.*

From this, we see that the mixing parameter can be improved exponentially when applying more gossip steps. To ensure $p_{\mathbf{W}} \geq 1 - \frac{1}{1 + \eta Ln}$, at most $k \leq \frac{\ln(1 + \eta Ln)}{p} = \tilde{\mathcal{O}}(\frac{1}{p})$ repetitions are required.

5.5 Inspecting Consensus Distance for Decentralized Training

Our analysis in Section 5.4 shows that we can—at least in theory—recover the convergence behavior of C-SGD by controlling the consensus distance. Now, we direct our focus on generalization in decentralized deep learning training. We show, empirically (not theoretically, see also Appendix 14.2.2), that the critical consensus distance is an important metric to capture the connection between optimization and generalization in deep learning—e.g. Figure 5.2 in Section 5.5.3 showcases that by addressing the optimization difficulty in the critical initial training phase (Figure 5.2(a) and Figure 5.2(b)), the final generalization gap can be perfectly closed (Figure 5.2(c), Table 5.2 and Table 5.3).

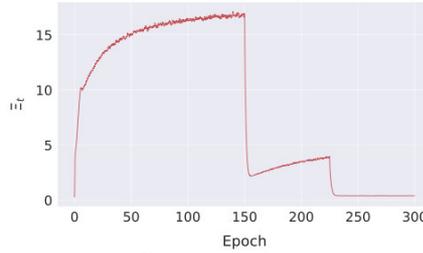


Figure 5.1 – Evolution of the consensus distance Ξ for ResNet-20 on CIFAR-10 ($n = 32$) with ring topology.

First we introduce and justify our experimental design in Section 5.5.1. We describe the implementation in Section 5.5.2. In Section 5.5.3, we present our findings on image classification benchmark with standard SGD optimizer, which is the main focus of this work; a preliminary study on Transformer with Adam optimizer and inverse square root learning rate schedule can be found in Section 5.5.4.

5.5.1 Experiment Design: Controlled Training Phases

Phase-wise training. Since the consensus distance evolves throughout training, identifying its impact at every training step is infeasible. However, as the consensus distance and critical consensus distance (CCD) both significantly depend on the learning rate (Propositions 5.4.2 and 5.4.3), we expect rather consistent observations during phases in which the learning rate is kept fixed and more drastic changes between such phases. On CV tasks, stage-wise learning rate schedule is the common practice for SOTA distributed training as described in Section 5.5.2: thus the training can be naturally divided into phases through the learning rate decay¹, in each of which training dynamics are significantly different from the others, such as Ξ_t (Figure 14.2(a)), ϕ_t (Figure 14.2(b)) and L -smoothness (Figure 14.2(c)). The transformer (NLP task) has no well-defined training phases due to the conventional inverse square root learning rate, thus for the sake of simplicity, we consider the entire transformer training as one phase as a preliminary study.

Individual phase investigation. In order to eliminate the coupling of effects from other phases, in each experiment we place only one phase under consensus distance control (the control refers to perform multiple gossip steps as in Section 5.4.3 to reach certain distance targets), while performing exact averaging (All-Reduce for all nodes) on model parameters for the other unstudied phases. We demonstrate in Table 5.5 of Section 5.5.3 that the decentralization impacts on various phases are rather orthogonal, which justifies our design of examining one phase at a time.

For the ease of presentation, the term “phase- x ” refers to a training phase between $(x-1)$ -th and x -th learning rate decay. The notation “dec-phase- x ” indicates that only in “phase- x ” the

¹ The learning rate warmup is only over a very small fraction of training epochs (e.g. 5 out of 300 epochs on CIFAR-10). To simplify the analysis, we do not consider it as a separate phase.

5.5. Inspecting Consensus Distance for Decentralized Training

model is trained with a decentralized communication topology, while for other phases we perform All-Reduce on model parameters. We compare the result of each individually decentralized phase with that of All-Reduce centralized training (on all training phases), so as to identify when (which phase) and how decentralized training influences final generalization performance.

5.5.2 Experimental Setup

Datasets and models. We empirically study the decentralized training behavior on the following two tasks, on convolutional neural networks and transformer architectures: (1) Image Classification for CIFAR-10 (Krizhevsky and Hinton, 2009) and ImageNet-32 (i.e. image resolution of 32, Chrabaszcz et al., 2017), with the standard data augmentation and preprocessing scheme (He et al., 2016a); and (2) Neural Machine Translation for the Multi30k dataset (Elliott et al., 2016). For Image Classification, ResNet-20 (He et al., 2016a) with various widths are used on CIFAR (default width of 1) and ImageNet-32 (width factor of 3).² For Neural Machine Translation, a down-scaled transformer architecture (by 2 w.r.t. the base model in Vaswani et al., 2017) is used. Weight initialization schemes follow Goyal et al. (2017); He et al. (2015) and Vaswani et al. (2017) respectively. Unless mentioned otherwise, our experiments are repeated over three random seeds.

Training schemes. We use mini-batch SGD with a Nesterov momentum of 0.9 without dampening for image classification task (we confirm our findings in Section 5.5.3 for SGD without momentum), and Adam is used for neural machine translation task. Unless mentioned otherwise we use the optimal learning rate (lr) from centralized training for our decentralized experiments³ in order to observe the impact of *decentralization* on normal *centralized* training.

- For image classification experiments, unless mentioned otherwise, the models are trained for 300 and 90 epochs for CIFAR-10 and ImageNet-32 respectively; the local mini-batch size are set to 32 and 64. By default, all experiments follow the SOTA learning rate scheme in the distributed deep learning literature (Goyal et al., 2017; He et al., 2019a) with learning rate scaling and warmup scheme. The learning rate is always gradually warmed up from a relatively small value (i.e. 0.1) for the first 5 epochs. Besides, the learning rate will be divided by 10 when the model has accessed specified fractions of the total number of training samples (He et al., 2016a); we use $\{\frac{1}{2}, \frac{3}{4}\}$ and $\{\frac{1}{3}, \frac{2}{3}, \frac{8}{9}\}$ for CIFAR and ImageNet respectively. All results in tables are test top-1 accuracy.
- For experiments on neural machine translation, we use standard inverse square root learning

² It takes ~ 7 h to for 1 round of standard ImageNet-32 training with $n=16$ V100 on a ring, and the cost increases to 12h for our consensus distance controlled experiments. Experiments on datasets of larger scales are beyond our computation budget.

³ We find that fine-tuning the learning rate for decentralized experiments does not change our conclusions. I.e. no significant difference can be found for the curves at phase-1 for “ring (fine-tuned lr)” and “dec-phase-1 (Ξ_{\max})” in Figure 5.2(a) and 5.2(b). We have similar observations in Table 14.6 after the sufficient learning rate tuning on phase-1.

rate schedule (Vaswani et al., 2017) with local mini-batch size 64. The warm-up step is set to 4000 for the mini-batch size of 64 and is linearly scaled down by the global mini-batch size.

Consensus distance control. For consensus control, we adopt the “more gossip iterations” strategy introduced in Section 5.4.3. That is, we perform multiple gossip steps (if needed) until reaching the desired target consensus distance value. Two metrics are considered to set the consensus distance target value during the specified training phase:

- constant target distance (main approach⁴): the target consensus distance Ξ for a phase is the *maximum consensus distance* Ξ_{\max} of the *current phase* in normal (uncontrolled) decentralized training, multiplied by a factor. For a given topology, the smaller the factor, the tighter the consensus.
- adaptive target distance (in Appendix 14.5.3): the target consensus distance Ξ for the current step is the averaged local gradient norm ϕ_t^{avg} scaled by a factor. For stability, we use the exponentially moving averaged value ϕ_t^{ema} of ϕ_t^{avg} (accumulated during the corresponding phase).

We use a ring as the main decentralized communication topology, as it is a particularly hard instance with a small spectral gap (cf. Table 14.3) which allows us to study a wide range of target consensus distances by modifying the number of gossip steps (in appendix we show consistent findings on time varying exponential topology in Table 14.11 and 14.12).

5.5.3 Findings on Computer Vision Tasks

In this section we present our empirical findings and provide insights into how consensus distance at various phases impacts the training generalization for CV tasks (i.e. CIFAR-10, Imagenet-32).

Critical consensus distance exists in the initial training phase—consensus distance below this critical threshold ensures good optimization and generalization. In the initial training phase, both training and generalization performance are heavily impacted by the consensus distance (“dec-phase-1” in Figure 5.2 and Table 5.2). A smaller consensus distance in the early phase results in considerably faster optimization (training loss) and higher generalization performance (test accuracy), and these advantages persist over the entire training.

When the consensus distance is larger (i.e. $1/2 \Xi_{\max}$ for CIFAR-10), the optimization (training performance) can eventually catch up with the centralized convergence (c.f. Figure 5.2(a) and 5.2(b)) but a considerable generalization gap still remains (92.36 v.s. 92.82 for the setup in Figure 5.2) as shown in Table 5.2. A consistent pattern can be found in ImageNet-32 experiments⁵, as shown

⁴ We use this one primarily since we can directly regulate the magnitude of consensus distance. In experiments, target $\Xi = \Xi_{\max}$ refers to the normal (i.e. uncontrolled) decentralized training.

⁵ $1/2 \Xi_{\max}$ has already been tight enough to recover the centralized performance for ImageNet-32 ($n=32$), while a significant performance drop can be observed between Ξ_{\max} and $1/2 \Xi_{\max}$.

5.5. Inspecting Consensus Distance for Decentralized Training

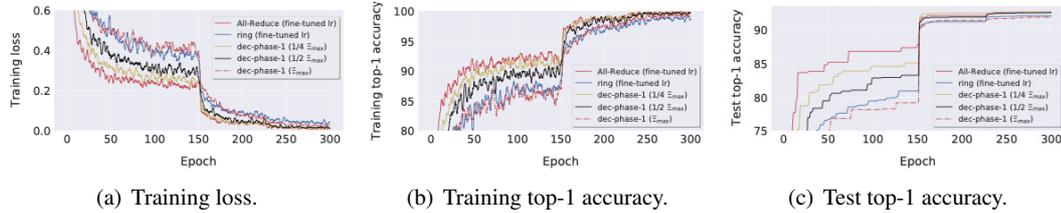


Figure 5.2 – Learning curves for ResNet-20 on CIFAR-10 ($n=32$). We compare fine-tuned normal (w/o control) decentralized training (i.e. “ring”) with dec-phase-1 on various target consensus distances.

in Table 5.3. These observations to some extent are consistent with the insights of the critical learning phase described in Golatkar et al. (2019); Jastrzebski et al. (2020); Frankle et al. (2020) for centralized training, where it is argued that the initial learning phase is crucial for the final generalization.

Notably, perfect consensus distance is not required to recover the centralized training performance. For instance, $1/4 \Xi_{\max}$ is sufficient in CIFAR-10 experiments to approach the optimal centralized training performance in both optimization and *generalization* at the end. Smaller distances (e.g. $1/8 \Xi_{\max}$, $1/16 \Xi_{\max}$) do not bring significant gain (92.77 and 92.72 respectively in Table 14.10). The performance saturates (c.f. 92.74 for $1/4 \Xi_{\max}$) with significantly increased communication overhead (e.g. Figure 14.6 of Appendix 14.5.1). This confirms that our analysis and discovery in Section 5.4 are sensible in the initial training phase: *there exists a critical consensus distance for the training, below which the impact of decentralization is negligible*.

A non-negligible consensus distance at middle phases can improve generalization over centralized training. Surprisingly, it is not always the case that the generalization performance improves with a shrinking consensus distance. We observe that at the phase right after the initial training plateaus (e.g. phase-2 for CIFAR-10, phase-3 for Imagenet-32), a non-negligible consensus distance⁶ actually boosts the generalization performance over the centralized training which has been deemed optimal. In CIFAR-10 dec-phase-2 experiments (Table 5.2), the generalization performance increases monotonically with the evaluated consensus distance and is consistently superior to that of the centralized training (e.g. 93.04, 92.99, 92.87 over 92.82 for $n=32$). Analogous observation can be obtained in Imagenet-32 dec-phase-3 experiments (Table 5.3).

This coincides with the observations firstly introduced in post-local SGD (Lin et al., 2020d) (see Chapter 2), where for better generalization, consensus distance is created among local machines by less frequent model parameter synchronization (All-Reduce) in late training phases (e.g. phase-2, phase-3 for CIFAR). Thus non-negligible consensus distance at middle phases can be viewed as a means of injecting proper noise as argued in Lin et al. (2020d), which reduces communication cost and in the meanwhile benefits generalization.

⁶ Table 14.12 of Appendix 14.5.3 shows that there exists optimal consensus distance at middle phases, beyond which the gain in generalization (brought by noise injection) starts to diminish.

Chapter 5. Consensus Control for Decentralized Deep Learning

Table 5.2 – **The impact of consensus distance of various phases on generalization performance** (test top-1 accuracy) of training ResNet-20 on CIFAR-10 on ring. The All-Reduce performance for $n=32$ and $n=64$ are 92.82 ± 0.27 and 92.71 ± 0.11 respectively. The fine-tuned normal (w/o control) decentralized training performance for $n=32$ and $n=64$ are 91.74 ± 0.15 and 89.87 ± 0.12 respectively.

# nodes	target Ξ	dec-phase-1			dec-phase-2			dec-phase-3		
	Ξ_{\max}	$1/2 \Xi_{\max}$	$1/4 \Xi_{\max}$	Ξ_{\max}	$1/2 \Xi_{\max}$	$1/4 \Xi_{\max}$	Ξ_{\max}	$1/2 \Xi_{\max}$	$1/4 \Xi_{\max}$	
$n=32$	91.78 ± 0.35	92.36 ± 0.21	92.74 ± 0.10	93.04 ± 0.01	92.99 ± 0.30	92.87 ± 0.11	92.60 ± 0.00	92.82 ± 0.21	92.85 ± 0.24	
$n=64$	90.31 ± 0.12	92.18 ± 0.07	92.45 ± 0.17	93.14 ± 0.04	92.94 ± 0.10	92.79 ± 0.07	92.23 ± 0.12	92.50 ± 0.09	92.60 ± 0.10	

Table 5.3 – **The impact of consensus distances on generalization for various phases** of training ResNet-20-3 on ImageNet-32 on ring. The centralized baseline performances for $n=16$ and $n=32$ are 51.74 ± 0.06 and 51.98 ± 0.37 respectively, while those of decentralized training (on a fixed ring) are 51.04 ± 0.06 and 50.17 ± 0.04 . The reported test top-1 accuracies are over two seeds.

# nodes	target Ξ	dec-phase-1			dec-phase-2			dec-phase-3			dec-phase-4		
	Ξ_{\max}	$1/2 \Xi_{\max}$	$1/4 \Xi_{\max}$										
$n=16$	51.22 ± 0.08	51.79 ± 0.10	51.71 ± 0.03	51.59 ± 0.02	51.67 ± 0.01	51.65 ± 0.13	51.80 ± 0.10	51.81 ± 0.13	51.81 ± 0.04	51.72 ± 0.02	51.76 ± 0.01	51.74 ± 0.06	
$n=32$	50.76 ± 0.18	51.27 ± 0.07	51.60 ± 0.21	51.39 ± 0.07	51.59 ± 0.04	51.66 ± 0.12	51.79 ± 0.06	51.73 ± 0.10	51.77 ± 0.10	51.70 ± 0.02	51.71 ± 0.02	51.70 ± 0.02	

At the last phase of training, the consensus distance only marginally impacts the generalization performance. Similar to the initial training phase, the final convergence phase seems to favor small consensus distances in CIFAR-10 experiments. However, its impact is less prominent in comparison: for dec-phase-3, performance of a smaller consensus distance ($1/4 \Xi_{\max}$) is only 0.25% and 0.37% higher than that of Ξ_{\max} for $n=32, 64$ respectively (Table 5.2). In Imagenet-32 experiments, dec-phase-3 performance is not even affected by changes in consensus.

Quality propagation across phases. Our previous experiments only consider a single phase of decentralized training. We now evaluate the lasting impact of consensus across the sequence of multiple phases. In Table 5.5, we control the consensus distance for both phase-1 and phase-2 when training on CIFAR-10. Our previous findings hold when we view each controlled phase separately. For instance, when we apply $1/2 \Xi_{\max}$ consensus control to phase-2 (the middle column in Table 5.5), we can still observe that a smaller consensus distance in phase-1 results in a higher performance as in our previous finding. Hence our previous findings are valid in more general cases of decentralized training.

Longer training cannot close the generalization gap caused by large consensus distances in the initial training phase. As discussed above, large consensus distances in the initial phase can result in significant generalization loss. Table 5.6 investigates whether a prolonged training on the initial phase can address this difficulty: we prolong the phase-1 for CIFAR-10 with a range of consensus distances and leave the other training phases centralized. We can observe that although longer training is beneficial for each consensus distance, it cannot recover the generalization gap resulting from large consensus distance. For instance, the maximum gain (among all evaluated cases) of increasing the epoch number from 150 to 250 is 0.31% at $1/2 \Xi_{\max}$, which is lower than the average gain (around 0.6%) of merely reducing the consensus distance from Ξ_{\max} to $1/2 \Xi_{\max}$. Table 14.7 in Appendix 14.5.2 evaluates cases where dec-phase-2 and dec-phase-3 are prolonged. We find longer training in these two phases brings about negligible performance gain.

5.5. Inspecting Consensus Distance for Decentralized Training

Table 5.4 – **The impact of consensus distance on generalization performance with vanilla SGD (without momentum)** (test top-1 accuracy) of training ResNet-20 on CIFAR-10 on ring. The All-Reduce performance for $n=32$ and $n=64$ are 90.64 ± 0.19 and 90.58 ± 0.26 respectively. The fine-tuned normal (w/o control) decentralized training performance for $n=32$ and $n=64$ are 90.30 ± 0.14 and 88.92 ± 0.23 respectively. We repeat experiments for $n=32$ for 3 seeds and $n=64$ for 2 seeds.

# nodes	target Ξ	dec-phase-1			dec-phase-2		
	Ξ_{\max}	$1/2 \Xi_{\max}$	$1/4 \Xi_{\max}$	Ξ_{\max}	$1/2 \Xi_{\max}$	$1/4 \Xi_{\max}$	
$n=32$	90.51 ± 0.05	90.74 ± 0.14	90.88 ± 0.37	90.64 ± 0.18	90.55 ± 0.19	90.57 ± 0.17	
$n=64$	88.80 ± 0.03	89.89 ± 0.03	90.43 ± 0.05	90.63 ± 0.37	90.46 ± 0.15	90.63 ± 0.25	

Table 5.5 – **Quality propagation across training phases with various consensus distances** on ResNet-20 for CIFAR-10 (Ring with $n=32$). In phase-1 and phase-2, the model parameters reach inexact consensus of various target consensus distance Ξ , while phase-3 performs All-Reduce on model parameters.

phase-1 \ phase-2	Ξ_{\max}	$1/2 \Xi_{\max}$	$1/4 \Xi_{\max}$
$1/2 \Xi_{\max}$	92.48 ± 0.19	92.46 ± 0.11	92.31 ± 0.23
$1/4 \Xi_{\max}$	92.73 ± 0.11	92.66 ± 0.08	92.69 ± 0.19
$1/8 \Xi_{\max}$	93.10 ± 0.22	92.88 ± 0.15	92.91 ± 0.06

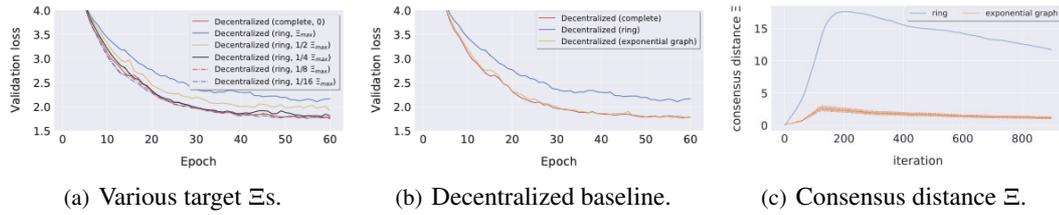


Figure 5.3 – Learning curves for training Transformer on Multi30k ($n=32$).

Consistent findings on decentralized SGD without momentum. To validate the coherence between our theory and experiments, we perform similar consensus distance control experiments on vanilla SGD optimizer (i.e. without momentum) for dec-phase-1 and dec-phase-2 on CIFAR-10. The patterns illustrated in Table 5.4 are consistent with our previous observations in Table 5.2 and Table 5.3, supporting the claim on the relation between consensus distance and generalization performance (which stands regardless of the use of momentum).

5.5.4 Preliminary Study on Training Transformer Models

The critical consensus distance also exists in NLP tasks. Figure 5.3(a) demonstrates that $1/4 \Xi_{\max}$ target control on a ring is sufficient to recover the centralized training performance. Besides, the target consensus distance in this case can be reached by exponential graph (and thus target test performance, as shown in Figure 5.3(b) and 5.3(c)). These justify the importance of designing an efficient communication topology/scheme in practice so as to effectively reach the CCD.

Chapter 5. Consensus Control for Decentralized Deep Learning

Table 5.6 – **The impact of various numbers of training epochs (at phase-1)** on generalization, for training ResNet-20 on CIFAR-10 (dec-phase-1 with $n=32$). The number of epochs at phase-1 is chosen from $\{150, 200, 250\}$, while the other training setting is identical to that of dec-phase-1 in Table 5.2.

target Ξ	training epochs at phase-1		
	150	200	250
Ξ_{\max}	91.78 ± 0.35	91.91 ± 0.19	92.04 ± 0.14
$1/2 \Xi_{\max}$	92.36 ± 0.21	92.55 ± 0.07	92.67 ± 0.13
$1/4 \Xi_{\max}$	92.74 ± 0.10	92.91 ± 0.15	92.84 ± 0.20

Table 5.7 – **The importance of phase-1** for training ResNet-20 on CIFAR-10 ($n=32$), in terms of (1) **target consensus distance** and (2) **the number of training epochs**. In phase-2 and phase-3, we perform *decentralized* training (w/o consensus distance control).

# of epochs	target Ξ	Ξ_{\max}	$1/2 \Xi_{\max}$	$1/4 \Xi_{\max}$	$1/8 \Xi_{\max}$	$0 \Xi_{\max}$
	150		91.74 ± 0.15	92.31 ± 0.12	92.81 ± 0.22	92.91 ± 0.15
200		91.81 ± 0.22	92.88 ± 0.20	93.00 ± 0.18	93.01 ± 0.10	92.90 ± 0.17
250		92.09 ± 0.23	92.74 ± 0.11	93.15 ± 0.26	92.99 ± 0.24	93.31 ± 0.06

5.6 Impact on Practice

Practical guidelines: prioritizing the initial training phase. Apart from effectiveness (generalization/test performance), efficiency (time) stands as the other crucial goal in deep learning, and thus how to allocate communication resource over the training becomes a relevant question.

As indicated by our first empirical finding (and theory in Section 5.4), the initial training phase bears the greatest importance over all other training phases; therefore the communication expenditure should be concentrated on the initial phase to maintain a consensus distance lower than the CCD. We suggest a list of communication topologies with superior spectral properties, i.e. exponential graph (Assran et al., 2019) and random matching (Nadiradze et al., 2021) in Figure 5.4 (the definition of the topology is detailed in Appendix 14.5.1), which can be utilized to achieve fast convergence in gossip averaging.

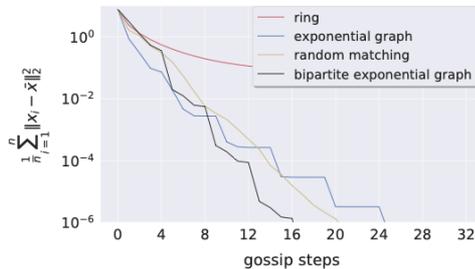


Figure 5.4 – **Consensus distance evolution against the number of gossip steps** on different topologies ($n=32$). The initial x_i 's are sampled uniformly from $[0, 10]$. Results on various topology scales are deferred to Appendix 14.5.1.

The late training phases should be less prioritized for communication resources, due to the generalization benefits from a reasonable consensus distance in the middle phases. Providing a rigorous way to quantify the optimal consensus distance is non-trivial, and is left as future work.

In Table 5.7 we show that the above-mentioned guideline is practically feasible: as long as the quality of the initial phase is ensured, we can afford to slacken the consensus control for later phases, in particular the middle phase. For instance, when the number of epochs is 150, a consensus control of $1/4 \Xi_{\max}$ in the initial phase with uncontrolled middle and final phase is adequate to recover the centralized training performance (92.81 v.s. 92.82). Note that here the noise injection from the uncontrolled middle phase also contributes positively to the performance. Table 14.11 in Appendix 14.5.3 additionally justifies the applicability of applying this guideline on exponential graphs.

Practical implementation of Consensus Control in Data-Centers. Computing the exact consensus distance requires the average of all model parameters in \mathbb{R}^d , which is prohibitively expensive (All-Reduce). We propose therefore to use the following efficient estimator

$$\Theta_t^2 := \frac{1}{n} \sum_{i=1}^n \theta_i^{(t)} \quad \text{with} \quad \theta_i^{(t)} := \left\| \sum_{j=1}^n w_{ij} \mathbf{x}_j^{(t)} - \mathbf{x}_i^{(t)} \right\|_2^2,$$

instead (in Lemma 14.1.1 we prove that $\Xi_t \leq \frac{2}{p} \Theta_t$ is an upper-bound of consensus distance and thus a valid control parameter, see also Section 14.1.2 for numerical validation). The values $\theta_i^{(t)} \in \mathbb{R}$ can be computed *locally* on each node when updating the parameters at negligible cost (compared to gradient computations), and computing Θ_t requires only averaging of scalars. While this can be implemented efficiently in data-centers (the cost of averaging these scalar values is negligible compared to averaging high-dimensional parameter vectors in the gossip steps), this might not be efficient over arbitrary decentralized network.

Table 14.1 and 14.2 in Appendix 14.1.2 show the feasibility of integrating the control of Θ_t with our practical guidelines for efficient training in data-centers, which serves as a strong starting point for designing decentralized training algorithms with a desired balance between communication cost and training performance.

5.7 Conclusion

In this work, we theoretically identify the consensus distance as an essential factor for decentralized training. We show the existence of a critical consensus distance, below which the consensus distance does not hinder optimization. Our deep learning experiments validate our theoretical findings and extend them to the generalization performance. Based on these insights, we propose practical guidelines for favorable generalization performance with low communication expenses, on arbitrary communication networks.

While we focused in this work on data-center training with iid data as an important first step, consensus control may be of even greater importance in non-i.i.d. scenarios (such as in [Hsieh et al., 2020](#)).

Learning Efficiency: Computational Efficiency **Part II**

6 Dynamic Model Pruning with Feedback

6.1 Preface

Contribution and sources. This chapter builds on the work done in [Lin et al. \(2020c\)](#). The author proposed the initial idea and led the whole project, including developing and implementing the algorithms, designing experiments, and conducting some analyses for deep learning. The theoretical analysis was mostly provided by Sebastian U. Stich. Daniil Dmitriev assisted in running some experiments, and Luis Barba contributed to the paper writing.

Summary. Deep neural networks often have millions of parameters. This can hinder their deployment to low-end devices, not only due to high memory requirements but also because of increased latency at inference. We propose a novel model compression method that generates a sparse trained model without additional overhead: by allowing (i) dynamic allocation of the sparsity pattern and (ii) incorporating feedback signal to reactivate prematurely pruned weights we obtain a performant sparse model in one single training pass (retraining is not needed, but can further improve the performance). We evaluate our method on CIFAR-10 and ImageNet, and show that the obtained sparse models can reach the state-of-the-art performance of dense models. Moreover, their performance surpasses that of models generated by all previously proposed pruning schemes.

6.2 Introduction

Highly overparametrized deep neural networks show impressive results on machine learning tasks. However, with the increase in model size comes also the demand for memory and computer power at inference stage—two resources that are scarcely available on low-end devices. Pruning techniques have been successfully applied to remove a significant fraction of the network weights while preserving test accuracy attained by dense models. In some cases, the generalization of compressed networks has even been found to be better than with full models (Han et al., 2015, 2017; Mocanu et al., 2018).

The *sparsity* of a network is the number of weights that are identically zero, and can be obtained by applying a *sparsity mask* on the weights. There are several approaches to find sparse models. For instance, *one-shot pruning* strategies find a suitable sparsity mask by inspecting the weights of a pretrained network (Mozer and Smolensky, 1989; LeCun et al., 1989; Han et al., 2017). While these algorithms achieve a substantial size reduction of the network with little degradation in accuracy, they are computationally expensive (training and refinement on the dense model), and they are outperformed by algorithms that explore various sparsity masks instead of a single one. In *dynamic pruning* methods, the sparsity mask is readjusted during training according to various criteria (Mostafa and Wang, 2019; Mocanu et al., 2018). However, these methods require fine-tuning of many hyperparameters.

We propose a new pruning approach to obtain sparse neural networks with state-of-the-art test accuracy. Our compression scheme uses a new saliency criterion that identifies important weights in the network throughout training to propose candidate masks. As a key feature, our algorithm not only evolves the pruned sparse model alone, but jointly also a (closely related) dense model that is used in a natural way to correct for pruning errors during training. This results in better generalization properties on a wide variety of tasks, since the simplicity of the scheme allows us further to study it from a theoretical point of view, and to provide further insights and interpretation. We do not require tuning of additional hyperparameters, and no retraining of the sparse model is needed (though can further improve performance).

Contributions.

- A novel dynamic pruning scheme, that incorporates an error feedback in a natural way and finds a trained sparse model in one training pass. Sec. 6.4 and Sec. 6.6
- We demonstrate state-of-the-art performance (in accuracy and sparsity), outperforming all previously proposed pruning schemes. Sec. 6.6
- We complement our results by an ablation study that provides further insights and convergence analysis for convex and non-convex objectives. Sec. 6.7 and Sec. 6.5

6.3 Related Work

Previous works on obtaining pruned networks can (loosely) be divided into three main categories.

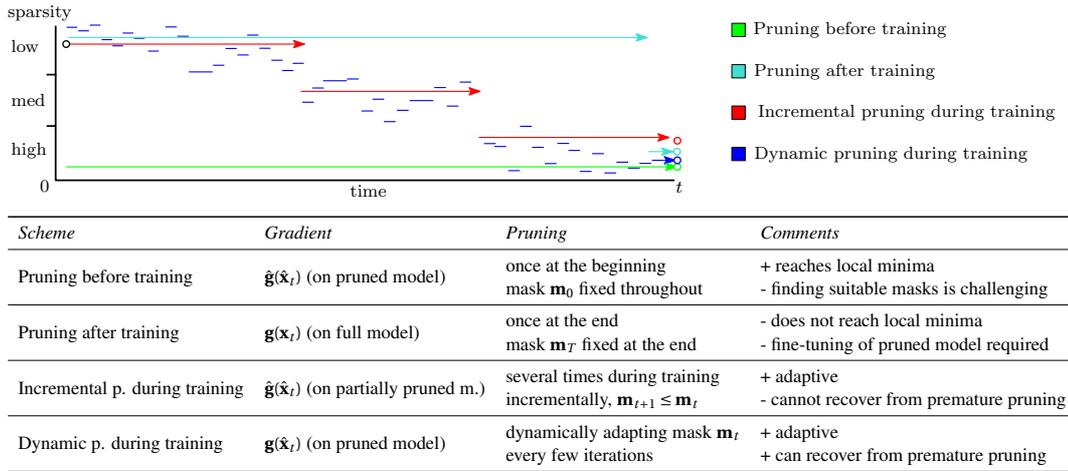


Figure 6.1 – Schematic view of various pruning methodologies and their properties.

Pruning after training. Training approaches to obtain sparse networks usually include a three stage pipeline—training of a dense model, *one-shot pruning* and fine-tuning—e.g. (Han et al., 2015). Their results (i.e. moderate sparsity level with minor quality loss) made them the standard method for network pruning and led to several variations (Alvarez and Salzmann, 2016; Guo et al., 2016; Alvarez and Salzmann, 2017; Carreira-Perpinán and Idelbayev, 2018).

Pruning during training. Zhu and Gupta (2017) propose the use of magnitude-based pruning and to gradually increase the sparsity ratio while training the model from scratch. A pruning schedule determines when the new masks are computed (extending and simplifying (Narang et al., 2017)). SFP (He et al., 2018) prunes entire filters of the model at the end of each epoch, but allow the pruned filters to be updated when training the model. Deep Rewiring (DeepR) (Bellec et al., 2018) allows for even more adaptivity by performing pruning and regrowth decisions periodically. This approach is computationally expensive and challenging to apply to large networks and datasets. Sparse evolutionary training (SET) (Mocanu et al., 2018) simplifies prune–regrowth cycles by using heuristics for random growth at the end of each training epoch and NeST (Dai et al., 2019) by inspecting gradient magnitudes.

Dynamic Sparse Reparameterization (DSR) (Mostafa and Wang, 2019) implements a prune–redistribute–regrowth cycle where target sparsity levels are redistributed among layers, based on loss gradients (in contrast to SET, which uses fixed, manually configured, sparsity levels). Sparse Momentum (SM) (Dettmers and Zettlemoyer, 2019) follows the same cycle but instead using the mean momentum magnitude of each layer during the redistribute phase. SM outperforms DSR on ImageNet for unstructured pruning by a small margin but has no performance difference on CIFAR experiments. Our approach also falls in the dynamic category but we use error compensation mechanisms instead of hand crafted redistribute–regrowth cycles.

Pruning before training. Recently—spurred by the lottery ticket hypothesis (LT) (Frankle and Carbin, 2019)—methods which try to find a sparse mask that can be trained from scratch have attracted increased interest. For instance, Lee et al. (2019) propose SNIP to find a pruning mask by inspecting connection sensitivities and identifying structurally important connections in the network for a given task. Pruning is applied at initialization, and the sparsity mask remains fixed throughout training. Note that Frankle and Carbin (2019); Frankle et al. (2019) do not propose an efficient pruning scheme to find the mask, instead they rely on iterative pruning, repeated for several full training passes.

Further Approaches. Srinivas et al. (2017); Louizos et al. (2018) learn gating variables (e.g. through ℓ_0 regularization) that minimize the number of nonzero weights, recent parallel work studies filter pruning for pre-trained models (You et al., 2019b). Gal et al. (2017); Neklyudov et al. (2017); Molchanov et al. (2017) prune from Bayesian perspectives to learn dropout probabilities during training to prune and sparsify networks as dropout weight probabilities reach 1. Gale et al. (2019) extensively study recent unstructured pruning methods on large-scale learning tasks, and find that complex techniques (Molchanov et al., 2017; Louizos et al., 2018) perform inconsistently. Simple magnitude pruning approaches achieve comparable or better results (Zhu and Gupta, 2017).

6.4 Method

We consider the training of a non-convex loss function $f: \mathbb{R}^d \rightarrow \mathbb{R}$. We assume for a weight vector $\mathbf{x} \in \mathbb{R}^d$ to have access to a stochastic gradient $\mathbf{g}(\mathbf{x}) \in \mathbb{R}^d$ such that $\mathbb{E}[\mathbf{g}(\mathbf{x})] = \nabla f(\mathbf{x})$. This corresponds to the standard machine learning setting with $\mathbf{g}(\mathbf{x})$ representing a (mini-batch) gradient of one (or several) components of the loss function. Stochastic Gradient Descent (SGD) computes a sequence of iterates by the update rule

$$\mathbf{x}_{t+1} := \mathbf{x}_t - \gamma_t \mathbf{g}(\mathbf{x}_t), \tag{SGD}$$

for some learning rate γ_t . To obtain a sparse model, a general approach is to *prune* some of the weights of \mathbf{x}_t , i.e. to set them to zero. Such pruning can be implemented by applying a *mask* $\mathbf{m} \in \{0, 1\}^d$ to the weights, resulting in a sparse model $\hat{\mathbf{x}}_t := \mathbf{m} \odot \mathbf{x}_t$, where \odot denotes the entry-wise (Hadamard) product. The mask could potentially depend on the weights \mathbf{x}_t (e.g. smallest magnitude pruning), or depend on t (e.g. the sparsity is incremented over time).

Before we introduce our proposed dynamic pruning scheme, we formalize the three main existing types of pruning methodologies (summarized in Figure 6.1). These approaches vary in the way the mask is computed, and the moment when it is applied.¹

¹The method introduced in Section 6.3 typically follow one of these broad themes loosely, with slight variations in detail. For the sake of clarity we omit a too technical and detailed discussion here.

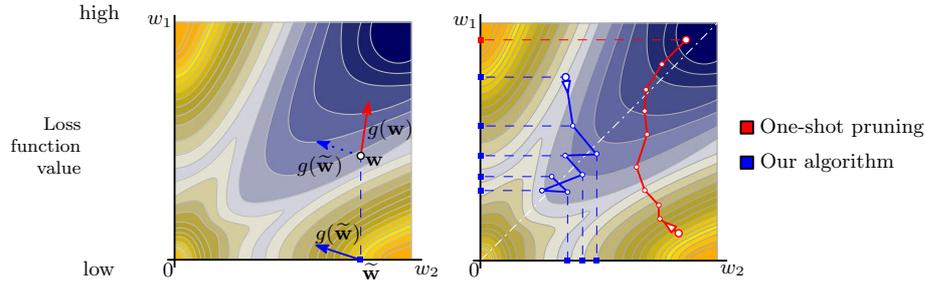


Figure 6.2 – **Left:** One-shot pruning (red) computes a stochastic gradient at \mathbf{x} and takes a step towards the best dense model. In contrast, DPF (blue) computes a stochastic gradient at the pruned model $\tilde{\mathbf{x}}$ (here obtained by smallest magnitude pruning), and takes a step that best suits the compressed model. **Right:** One-shot pruning commits to a single sparsity mask and might obtain sparse models that generalize poorly (without retraining). DPF explores various available sparsity patterns and finds better sparse models.

Pruning before training. A mask \mathbf{m}_0 (depending on e.g. the initialization \mathbf{x}_0 or the network architecture of f) is applied and (SGD) is used for training on the resulting sub-network $\hat{f}(\mathbf{x}) := f(\mathbf{m}_0 \odot \mathbf{x})$ with the advantage that only pruned weights need to be stored and updated², and that by training with SGD a local minimum of the subnetwork \hat{f} (but not of f —the original training target) can be reached. In practice however, it remains a challenge to efficiently determine a good mask \mathbf{m}_0 and a wrongly chosen mask at the beginning strongly impacts the performance.

Pruning after training (one-shot pruning). A dense model is trained, and pruning is applied to the trained model \mathbf{x}_T . As the pruned model $\hat{\mathbf{x}}_T = \mathbf{m}_T \odot \mathbf{x}_T$ is very likely not at a local optimum of f , fine-tuning (retraining with the fixed mask \mathbf{m}_T) is necessary to improve performance.

Pruning during training (incremental and dynamic pruning). Dynamic schemes change the mask \mathbf{m}_t every (few) iterations based on observations during training (i.e. by observing the weights and stochastic gradients). Incremental schemes monotonically increase the sparsity pattern, fully dynamic schemes can also reactivate previously pruned weights. In contrast to previous dynamic schemes that relied on elaborated heuristics to adapt the mask \mathbf{m}_t , we propose a simpler approach:

Dynamic pruning with feedback (DPF, Algorithm 16). Our scheme evaluates a stochastic gradient at the *pruned* model $\hat{\mathbf{x}}_t = \mathbf{m}_t \odot \mathbf{x}_t$ and applies it to the (simultaneously maintained) dense model \mathbf{x}_t :

$$\mathbf{x}_{t+1} := \mathbf{x}_t - \gamma_t \mathbf{g}(\mathbf{m}_t \odot \mathbf{x}_t) = \mathbf{x}_t - \gamma_t \mathbf{g}(\hat{\mathbf{x}}_t). \quad (\text{DPF})$$

Applying the gradient to the full model allows to recover from “errors”, i.e. prematurely masking out important weights: when the accumulated gradient updates from the following steps drastically change a specific weight, it can become activated again (in contrast to incremental pruning approaches that have to stick to sub-optimal decisions). For illustration, observe that (DPF) can

²When training on $\hat{f}(\mathbf{x})$, it suffices to access stochastic gradients of $\hat{f}(\mathbf{x})$, denoted by $\hat{\mathbf{g}}(\mathbf{x})$, which can potentially be cheaper be computed than by naively applying the mask to $\mathbf{g}(\mathbf{x})$ (note $\hat{\mathbf{g}}(\mathbf{x}) = \mathbf{m}_0 \odot \mathbf{g}(\mathbf{x})$).

Algorithm 4 The detailed training procedure of DPF.

Requires: uncompressed model weights $\mathbf{x} \in \mathbb{R}^d$, pruned weights: $\hat{\mathbf{x}}$, mask: $\mathbf{m} \in \{0, 1\}^d$; reparametrization period: p ; training iterations: T .

```

1: procedure
2:   for  $t \in \{1, \dots, T\}$  do
3:     if  $p \mid t$  (i.e. trigger mask update, per default every  $p = 16$  iterations) then
4:       compute mask  $\mathbf{m} \leftarrow \mathbf{m}_t(\mathbf{x}_t)$  ▷ by arbitrary pruning scheme
5:        $\hat{\mathbf{x}}_t \leftarrow \mathbf{m} \odot \mathbf{x}_t$  ▷ apply (precomputed) mask
6:       compute (mini-batch) gradient  $\mathbf{g}(\hat{\mathbf{x}}_t)$  ▷ forward/backward with pruned weights  $\hat{\mathbf{x}}_t$ 
7:        $\mathbf{x}_{t+1} \leftarrow$  gradient update  $\mathbf{g}(\hat{\mathbf{x}}_t)$  to  $\mathbf{x}_t$  ▷ via arbitrary optimizer
8:   return  $\mathbf{x}_T$  and  $\hat{\mathbf{x}}_T$ 

```

equivalently be written as

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \gamma_t \mathbf{g}(\mathbf{x}_t + \mathbf{e}_t),$$

where $\mathbf{e}_t := \hat{\mathbf{x}}_t - \mathbf{x}_t$ is the *error* produced by the compression. This provides a different intuition of the behavior of (DPF), and connects it with the concept of *error-feedback* (Stich et al., 2018; Karimireddy et al., 2019).³ We illustrate this principle in Figure 6.2 and give detailed pseudocode in Algorithm 4 and further implementation details in Appendix 15.1. The DPF scheme can also be seen as an instance of a more general class of schemes that apply (arbitrary) perturbed gradient updates to the dense model. For instance straight-through gradient estimators (Bengio et al., 2013) that are used to empirically simplify the backpropagation can be seen as such perturbations. Our stronger assumptions on the structure of the perturbation allow to derive non-asymptotic convergence rates in the next section, though our analysis could also be extended to the setting in Yin et al. (2019) if the perturbations can be bounded.

6.5 Convergence Analysis

We now present convergence guarantees for (DPF). For the purposes of deriving theoretical guarantees, we assume that the training objective is smooth, that is $\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq L \|\mathbf{x} - \mathbf{y}\|$, $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, for a constant $L > 0$, and that the stochastic gradients are bounded $\mathbb{E} \|\mathbf{g}(\hat{\mathbf{x}}_t)\|^2 \leq G^2$ for every pruned model $\hat{\mathbf{x}}_t = \mathbf{m}_t(\mathbf{x}_t) \odot \mathbf{x}_t$. The *quality* of this pruning is defined as the parameter $\delta_t \in [0, 1]$ such that

$$\delta_t := \|\mathbf{x}_t - \hat{\mathbf{x}}_t\|^2 / \|\mathbf{x}_t\|^2. \tag{6.1}$$

Pruning without information loss corresponds to $\hat{\mathbf{x}}_t = \mathbf{x}_t$, i.e. $\delta_t = 0$, and in general $\delta_t \leq 1$.

Convergence on Convex functions. We first consider the case when f is in addition μ -strongly convex, that is $\langle \nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle \leq f(\mathbf{x}) - f(\mathbf{y}) - \frac{\mu}{2} \|\mathbf{x} - \mathbf{y}\|^2$, $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^d$. While it is clear that this assumption does not apply to neural networks, it eases the presentation as strongly convex

³Our variable \mathbf{x}_t corresponds to $\tilde{\mathbf{x}}_t$ in the notation of Karimireddy et al. (2019). Their error-fixed SGD algorithm evaluates gradients at perturbed iterates $\mathbf{x}_t := \tilde{\mathbf{x}}_t + \mathbf{e}_t$, which correspond precisely to $\tilde{\mathbf{x}}_t = \mathbf{x}_t + \mathbf{e}_t$ in our notation. This shows the connection of these two methods.

functions have a unique (global) minimizer $\mathbf{x}^* := \operatorname{argmin}_{\mathbf{x} \in \mathbb{R}^d} f(\mathbf{x})$.

Theorem 6.5.1. *Let f be μ -strongly convex and learning rates given as $\gamma_t = \frac{4}{\mu(t+2)}$. Then for a randomly chosen pruned model $\hat{\mathbf{u}}$ of the iterates $\{\hat{\mathbf{x}}_0, \dots, \hat{\mathbf{x}}_T\}$ of DPF, concretely $\hat{\mathbf{u}} = \hat{\mathbf{x}}_t$ with probability $p_t = \frac{2(t+1)}{(T+1)(T+2)}$, it holds—in expectation over the stochasticity and the selection of $\hat{\mathbf{u}}$:*

$$\mathbb{E} f(\hat{\mathbf{u}}) - f(\mathbf{x}^*) = \mathcal{O} \left(\frac{G^2}{\mu T} + L \mathbb{E} [\delta_t \|\mathbf{x}_t\|^2] \right). \quad (6.2)$$

The rightmost term in (6.2) measures the average quality of the pruning. However, unless $\delta_t \rightarrow 0$ or $\|\mathbf{x}_t\| \rightarrow 0$ for $t \rightarrow \infty$, the error term never completely vanishes, meaning that the method converges only to a neighborhood of the optimal solution (this not only holds for the pruned model, but also for the jointly maintained dense model, as we will show in the appendix). This behavior is expected, as the global optimal model \mathbf{x}^* might be dense and cannot be approximated well by a sparse model.

For one-shot methods that only prune the final (SGD) iterate \mathbf{x}_T at the end, we have instead:

$$\mathbb{E} f(\hat{\mathbf{x}}_T) - f(\mathbf{x}^*) \leq 2\mathbb{E} (f(\mathbf{x}_T) - f(\mathbf{x}^*)) + L\delta_T \mathbb{E} \|\mathbf{x}_T\|^2 = \mathcal{O} \left(\frac{LG^2}{\mu^2 T} + L \mathbb{E} [\delta_T \|\mathbf{x}_T\|^2] \right),$$

as we show in the appendix. First, we see from this expression that the estimate is very sensitive to δ_T and \mathbf{x}_T , i.e. the quality of the pruning the final model. This could be better or worse than the average of the pruning quality of all iterates. Moreover, one loses also a factor of the condition number $\frac{L}{\mu}$ in the asymptotically decreasing term, compared to (6.2). This is due to the fact that standard convergence analysis only achieves optimal rates for an average of the iterates (but not the last one). This shows a slight theoretical advantage of DPF over rounding at the end.

Convergence on Non-Convex Functions to Stationary Points. Secondly, we consider the case when f is a non-convex function and show convergence (to a neighborhood) of a stationary point.

Theorem 6.5.2. *Let learning rate be given as $\gamma = \frac{c}{\sqrt{T}}$, for $c = \sqrt{\frac{f(\mathbf{x}_0) - f(\mathbf{x}^*)}{LG^2}}$. Then for pruned model $\hat{\mathbf{u}}$ chosen uniformly at random from the iterates $\{\hat{\mathbf{x}}_0, \dots, \hat{\mathbf{x}}_T\}$ of DPF, concretely $\hat{\mathbf{u}} := \hat{\mathbf{x}}_t$ with probability $p_t = \frac{1}{T+1}$, it holds—in expectation over the stochasticity and the selection of $\hat{\mathbf{u}}$:*

$$\mathbb{E} \|\nabla f(\hat{\mathbf{u}})\|^2 = \mathcal{O} \left(\frac{\sqrt{L(f(\mathbf{x}_0) - f(\mathbf{x}^*))}G}{\sqrt{T}} + L^2 \mathbb{E} [\delta_t \|\mathbf{x}_t\|^2] \right). \quad (6.3)$$

Extension to Other Compression Schemes. So far we put our focus on simple mask pruning schemes to achieve high model sparsity. However, the pruning scheme in Algorithm 16 could be replaced by an arbitrary compressor $\mathcal{C}: \mathbb{R}^d \rightarrow \mathbb{R}^d$, i.e. $\hat{\mathbf{x}}_t = \mathcal{C}(\mathbf{x}_t)$. Our analysis extends to compressors as e.g. defined in Karimireddy et al. (2019); Stich and Karimireddy (2020), whose quality is also measured in terms of the δ_t parameters as in (6.1). For example, if our objective was not to obtain a sparse model, but to produce a quantized neural network where inference could be computed faster on low-precision numbers, then we could define \mathcal{C} as a quantized compressor.

One variant of this approach is implemented in the Binary Connect algorithm (BC) (Courbariaux et al., 2015) with prominent results, see also Li et al. (2017a) for further insights and discussion.

6.6 Experiments

We evaluated DPF together with its competitors on a wide range of neural architectures and sparsity levels. **DPF exhibits consistent and noticeable performance benefits over its competitors.**

6.6.1 Experimental Setup

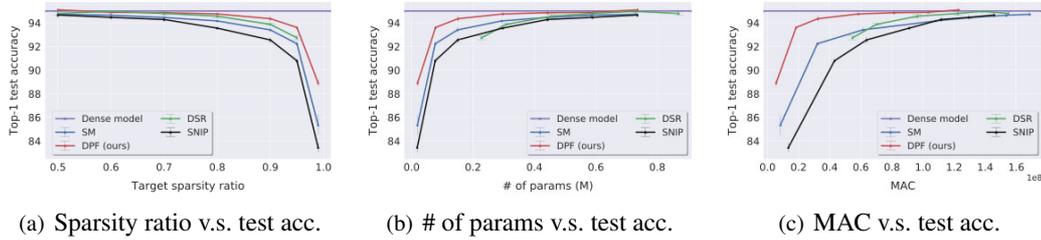
Datasets. We evaluated DPF on two image classification benchmarks: (1) CIFAR-10 (Krizhevsky and Hinton, 2009) (50K/10K training/test samples with 10 classes), and (2) ImageNet (Russakovsky et al., 2015) (1.28M/50K training/validation samples with 1000 classes). We adopted the standard data augmentation and preprocessing scheme from He et al. (2016a); Huang et al. (2016); for further details refer to Appendix 15.2.

Models. Following the common experimental setting in related work on network pruning (Liu et al., 2019d; Gale et al., 2019; Dettmers and Zettlemoyer, 2019; Mostafa and Wang, 2019), our main experiments focus on ResNet (He et al., 2016a) and WideResNet (Zagoruyko and Komodakis, 2016). However, DPF can be effectively extended to other neural architectures, e.g. VGG (Simonyan and Zisserman, 2015), DenseNet (Huang et al., 2017b). We followed the common definition in He et al. (2016a); Zagoruyko and Komodakis (2016) and used ResNet- a , WideResNet- a - b to represent neural network with a layers and width factor b .

Baselines. We considered the state-of-the-art model compression methods presented in the table below as our strongest competitors. We omit the comparison to other dynamic reparameterization methods, as DSR can outperform DeepR (Bellec et al., 2018) and SET (Mocanu et al., 2018) by a noticeable margin (Mostafa and Wang, 2019).

<i>Scheme</i>	<i>Reference</i>	<i>Pruning</i>	<i>How the mask(s) are found</i>
Lottery Ticket (LT) SNIP	2019, FDRC 2019, LAT	before training	10-30 successive rounds of (full training + pruning). By inspecting properties/sensitivity of the network.
One-shot + fine-tuning (One-shot P+FT)	2015, HPDT	after training	Saliency criterion (prunes smallest weights).
Incremental pruning + fine-tuning (Incremental)	2017, ZG	incremental	Saliency criterion. Sparsity is gradually incremented.
Dynamic Sparse Reparameterization (DSR)	2019, MW	dynamic	Prune–redistribute–regrowth cycle.
Sparse Momentum (SM)	2019, DZ		Prune–redistribute–regrowth + mean momentum.
DPF	<i>ours</i>		Reparameterization via error feedback.

Implementation of DPF. Compared to other dynamic reparameterization methods (e.g. DSR and SM) that introduced many extra hyperparameters, our method has trivial hyperparameter tuning overhead. We perform pruning across all neural network layers (no layer-wise pruning) using magnitude-based unstructured weight pruning (inherited from Han et al. (2015)). We found



(a) Sparsity ratio v.s. test acc. (b) # of params v.s. test acc. (c) MAC v.s. test acc.

Figure 6.3 – Top-1 test accuracy of **WideResNet-28-2** on **CIFAR-10** for unstructured weight pruning. The original model has 1.47M parameters with 216M MACs (Multiplier-ACcumulator). We varied the sparsity ratio from 50% to 99%. The complete numerical test accuracy values refer to Table 15.1 in Appendix 15.3.1. The lower # of params and MACs the model has, the higher sparsity ratio it uses. All results are averaged over three runs. Note that methods might consider various types of layers and thus the same pruning sparsity ratio might result in the slight difference for both of # of params and MACs. The DSR cannot converge when using the extreme high sparsity ratio (99%).

the best performance when updating the mask every 16 iterations (see also Figure 15.5) and we keep this value fixed for all experiments (independent of the architecture or task).

Unlike our competitors that may ignore some layers (e.g. the first convolution and downsampling layers in DSR), we applied DPF (as well as the One-shot P+FT and Incremental baselines) to all convolutional layers while keeping the last fully-connected layer⁴, biases and batch normalization layers dense. Lastly, our algorithm gradually increases the sparsity s_t of the mask from 0 to the desired sparsity using the same scheduling as in [Zhu and Gupta \(2017\)](#); see Appendix 15.2.

Training schedules. For all competitors, we adapted their open-sourced code and applied a consistent (and standard) training scheme over various methods to ensure a fair comparison. Following the standard training setup for CIFAR-10, we trained ResNet- a for 300 epochs and decayed the learning rate by 10 when accessing 50% and 75% of the total training samples ([He et al., 2016a](#); [Huang et al., 2017b](#)); and we trained WideResNet- $a-b$ as [Zagoruyko and Komodakis \(2016\)](#) for 200 epochs and decayed the learning rate by 5 when accessing 30%, 60% and 80% of the total training samples. For ImageNet training, we used the training scheme in [Goyal et al. \(2017\)](#) for 90 epochs and decayed learning rate by 10 at 30, 60, 80 epochs. For all datasets and models, we used mini-batch SGD with Nesterov momentum (factor 0.9) with fine-tuned learning rate for DPF. We reused the tuned (or recommended) hyperparameters for our baselines (DSR and SM), and fine-tuned the optimizer and learning rate for One-shot P+FT, Incremental and SNIP. The mini-batch size is fixed to 128 for CIFAR-10 and 1024 for ImageNet regardless of datasets, models and methods.

⁴ The last fully-connected layer normally makes up only a very small fraction of the total MACs, e.g. 0.05% for ResNet-50 on ImageNet and 0.0006% for WideResNet-28-2 on CIFAR-10.

Chapter 6. Dynamic Model Pruning with Feedback

Table 6.1 – Top-1 test accuracy of SOTA DNNs on **CIFAR-10** for unstructured weight pruning. We considered unstructured pruning and the \star indicates the method cannot converge. The results are averaged for three runs. The results we presented for each model consider some reasonable pruning ratios (we prune more aggressively for deeper and wider neural networks), and readers can refer to Table 15.1 (in Appendix 15.3.1) for a complete overview.

Model	Baseline on dense model	Methods				Target Pr. ratio
		SNIP (L ⁺ , 2019)	SM (DZ, 2019)	DSR (MW, 2019)	DPF	
VGG16-D	93.74 ± 0.13	93.04 ± 0.26	93.59 ± 0.17	-	93.87 ± 0.15	95%
ResNet-20	92.48 ± 0.20	91.10 ± 0.22	91.98 ± 0.01	92.00 ± 0.19	92.42 ± 0.14	70%
		90.53 ± 0.27	91.54 ± 0.16	91.78 ± 0.28	92.17 ± 0.21	80%
		88.50 ± 0.13	89.76 ± 0.40	87.88 ± 0.04	90.88 ± 0.07	90%
		84.91 ± 0.25	83.03 ± 0.74	\star	88.01 ± 0.30	95%
ResNet-32	93.83 ± 0.12	90.40 ± 0.26	91.54 ± 0.18	91.41 ± 0.23	92.42 ± 0.18	90%
		87.23 ± 0.29	88.68 ± 0.22	84.12 ± 0.32	90.94 ± 0.35	95%
ResNet-56	94.51 ± 0.20	91.43 ± 0.34	92.73 ± 0.21	93.78 ± 0.20	93.95 ± 0.11	90%
		\star	90.96 ± 0.40	92.57 ± 0.09	92.74 ± 0.08	95%
WideResNet-28-2	95.01 ± 0.04	92.58 ± 0.22	93.41 ± 0.22	93.88 ± 0.08	94.36 ± 0.24	90%
		90.80 ± 0.04	92.24 ± 0.14	92.74 ± 0.17	93.62 ± 0.05	95%
		83.45 ± 0.38	85.36 ± 0.80	\star	88.92 ± 0.29	99%
WideResNet-28-4	95.69 ± 0.10	93.62 ± 0.17	94.45 ± 0.14	94.63 ± 0.08	95.38 ± 0.04	95%
		92.06 ± 0.38	93.80 ± 0.24	93.92 ± 0.16	94.98 ± 0.08	97.5%
		89.49 ± 0.20	92.18 ± 0.04	92.50 ± 0.07	93.86 ± 0.20	99%
WideResNet-28-8	96.06 ± 0.06	95.49 ± 0.21	95.67 ± 0.14	95.81 ± 0.10	96.08 ± 0.15	90%
		94.92 ± 0.13	95.64 ± 0.07	95.55 ± 0.12	95.98 ± 0.10	95%
		94.11 ± 0.19	95.31 ± 0.20	95.11 ± 0.07	95.84 ± 0.04	97.5%
		92.04 ± 0.11	94.38 ± 0.12	94.10 ± 0.12	95.63 ± 0.16	99%
		74.50 ± 2.23	\star	88.65 ± 0.36	91.76 ± 0.18	99.9%

6.6.2 Experiment Results

CIFAR-10. Figure 6.3 shows a comparison of various methods for WideResNet-28-2. For low sparsity level (e.g. 50%), DPF outperforms even the dense baseline, which is in line with regularization properties of network pruning. Furthermore, DPF can prune the model up to a very high level (e.g. 99%), and still exhibit viable performance. This observation is also present in Table 6.1, where the results of training various state-of-the-art DNN architectures with higher sparsities are depicted. DPF shows reasonable performance even with extremely high sparsity level on large models (e.g. WideResNet-28-8 with sparsity ratio 99.9%), while other methods either suffer from significant quality loss or even fail to converge.

Because simple model pruning techniques sometimes show better performance than complex techniques (Gale et al., 2019), we further consider these simple models in Table 6.2. While DPF outperforms them in almost all settings, it faces difficulties pruning smaller models to extremely high sparsity ratios (e.g. ResNet-20 with sparsity ratio 95%). This seems however to be an artifact of fine-tuning, as DPF with extra fine-tuning convincingly outperforms all other methods regardless of the network size. This comes to no surprise as schemes like One-shot P+FT and Incremental do not benefit from extra fine-tuning, since it is already incorporated in their training

Table 6.2 – Top-1 test accuracy of SOTA DNNs on **CIFAR-10** for unstructured weight pruning via some simple pruning techniques. This table complements Table 6.1 and evaluates the performance of model compression under One-shot P+FT and Incremental, as well as how extra fine-tuning (FT) impact the performance of Incremental and our DPF. Note that One-shot P+FT prunes the dense model and uses extra fine-tuning itself. The Dense, Incremental and DPF train with the same number of epochs from scratch. The extra fine-tuning procedure considers the model checkpoint at the end of the normal training, uses the same number of training epochs (60 epochs in our case) with tuned optimizer and learning rate. Detailed hyperparameters tuning procedure refers to Appendix 15.2.

Model	Baseline on dense model	Methods					Target pr. ratio
		One-shot + FT (H ⁺ , 2015)	Incremental (ZG, 2017)	Incremental + FT	DPF	DPF + FT	
ResNet-20	92.48 ± 0.20	90.18 ± 0.12	90.55 ± 0.38	90.54 ± 0.25	90.88 ± 0.07	91.76 ± 0.12	90%
		86.91 ± 0.16	89.21 ± 0.10	89.24 ± 0.28	88.01 ± 0.30	90.34 ± 0.31	95%
ResNet-32	93.83 ± 0.12	91.72 ± 0.15	91.69 ± 0.12	91.76 ± 0.14	92.42 ± 0.18	92.61 ± 0.11	90%
		89.31 ± 0.18	90.86 ± 0.17	90.93 ± 0.18	90.94 ± 0.35	92.18 ± 0.14	95%
ResNet-56	94.51 ± 0.20	93.26 ± 0.06	93.14 ± 0.23	93.09 ± 0.16	93.95 ± 0.11	93.95 ± 0.17	90%
		91.61 ± 0.07	92.14 ± 0.10	92.50 ± 0.25	92.74 ± 0.08	93.25 ± 0.15	95%

procedure and they might become stuck in local minima. On the other hand, dynamic pruning methods, and in particular DPF, work on a different paradigm, and can still heavily benefit from fine-tuning.⁵

Figure 15.7 in Appendix 15.3.4 depicts another interesting property of DPF. When we search for a subnetwork with a (small) predefined number of parameters for a fixed task, it is much better to run DPF on a large model (e.g. WideResNet-28-8) than on a smaller one (e.g. WideResNet-28-2). That is, DPF performs structural exploration more efficiently in larger parametric spaces.

ImageNet. We compared DPF to other dynamic reparameterization methods as well as the strong Incremental baseline in Table 6.3. For both sparsity levels (80% and 90%), DPF shows a significant improvement of top-1 test accuracy with fewer or equal number of parameters.

6.7 Discussion

Besides the theoretical guarantees, a straightforward benefit of DPF over one-shot pruning in practice is its *fine-tuning free* training process. Figure 15.6 in the appendix (Section 15.3.3) demonstrates the trivial computational overhead (considering the dynamic reparameterization cost) of involving DPF to train the model from scratch. Small number of hyperparameters compared to other dynamic reparameterization methods (e.g. DSR and SM) is another advantage of DPF and Figure 15.5 further studies how various setups of DPF impact the final performance. Notice also that for DPF, inference is done only at sparse models, an aspect that could be leveraged for more efficient computations.

Empirical difference between one-shot pruning and DPF. From the Figure 6.2 one can see

⁵ Besides that a large fraction of the mask elements already converge during training (see e.g. Figure 6.4 below), not all mask elements converge. Thus DPF can still benefit from fine-tuning on the fixed sparsity mask.

Chapter 6. Dynamic Model Pruning with Feedback

Table 6.3 – Top-1 test accuracy of **ResNet-50** on **ImageNet** for unstructured weight pruning. The # of parameters for the full model is 25.56 M. We used the results of DSR from [Mostafa and Wang \(2019\)](#) as we use the same (standard) training/data augmentation scheme for the same neural architecture. Note that the evaluated methods prune various types of layers and result in diverse # of parameters for the same target pruning ratio. We also directly (and fairly) compare with the results of Incremental ([Zhu and Gupta, 2017](#)) reproduced and fine-tuned by [Gale et al. \(2019\)](#), where they consider layer-wise sparsity ratio and fine-tune both the sparsity warmup schedule and label-smoothing for better performance.

Method	Top-1 accuracy			Top-5 accuracy			Pruning ratio		remaining # of params
	Dense	Pruned	Difference	Dense	Pruned	Difference	Target	Reached	
Incremental (ZG, 2017)	75.95	74.25	-1.70	92.91	91.84	-1.07	80%	73.5%	6.79 M
DSR (MW, 2019)	74.90	73.30	-1.60	92.40	92.40	0	80%	71.4%	7.30 M
SM (DZ, 2019)	75.95	74.59	-1.36	92.91	92.37	-0.54	80%	72.4%	7.06 M
DPF	75.95	75.48	-0.47	92.91	92.59	-0.32	80%	73.5%	6.79 M
Incremental (Gale et al., 2019)	76.69	75.58	-1.11	-	-	-	80%	79.9%	5.15 M
DPF	75.95	75.13	-0.82	92.91	92.52	-0.39	80%	79.9%	5.15 M
Incremental (ZG, 2017)	75.95	73.36	-2.59	92.91	91.27	-1.64	90%	82.6%	4.45 M
DSR (MW, 2019)	74.90	71.60	-3.30	92.40	90.50	-1.90	90%	80.0%	5.10 M
SM (DZ, 2019)	75.95	72.65	-3.30	92.91	91.26	-1.65	90%	82.0%	4.59 M
DPF	75.95	74.55	-1.44	92.91	92.13	-0.78	90%	82.6%	4.45 M

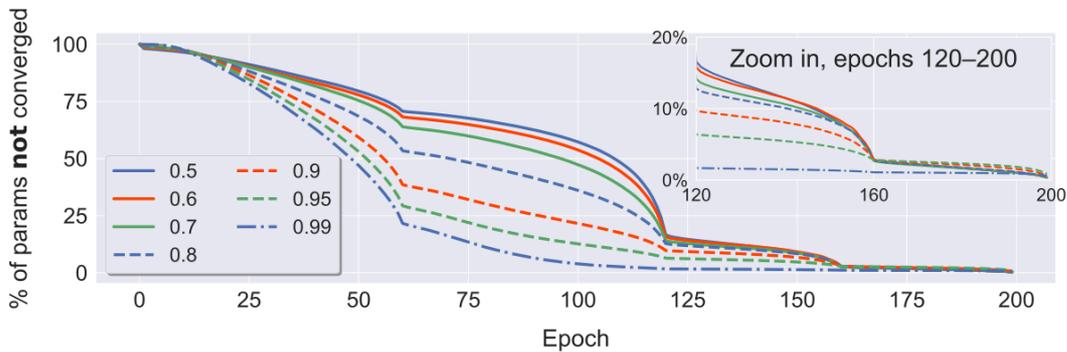


Figure 6.4 – Convergence of the pruning mask \mathbf{m}_t of DPF for various target sparsity levels (see legend). The y-axis represent the percentage of mask elements that still change **after** a certain epoch (x-axis). The illustrated example are from WideResNet-28-2 on CIFAR-10. We decayed the learning rate at 60,120,160 epochs.

that DPF tends to oscillate among several local minima, whereas one-shot pruning, even with fine-tuning, converges to a single solution, which is not necessarily close to the optimum. We believe that the wider exploration of DPF helps to find a better local minima (which can be even further improved by fine-tuning, as shown in Table 6.2). We empirically analyzed how drastically the mask changes between each reparameterization step, and how likely it is for some pruned weight to become non-zero in the later stages of training. Figure 6.4 shows at what stage of the training each element of the final mask becomes fixed. For each epoch, we report how many mask elements were flipped starting from this epoch. As an example, we see that for sparsity ratio 95%, after epoch 157 (i.e. for 43 epochs left), only 5% of the mask elements were changing. This suggests that, except for a small percentage of weights that keep oscillating, the mask has converged early in the training. In the final epochs, the algorithm keeps improving accuracy, but the masks are only being fine-tuned. A similar mask convergence behavior can be found in

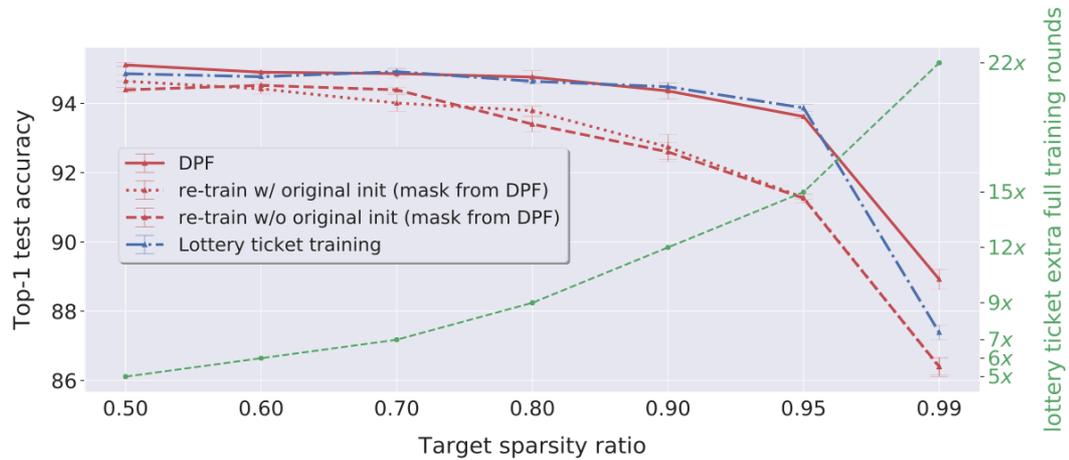
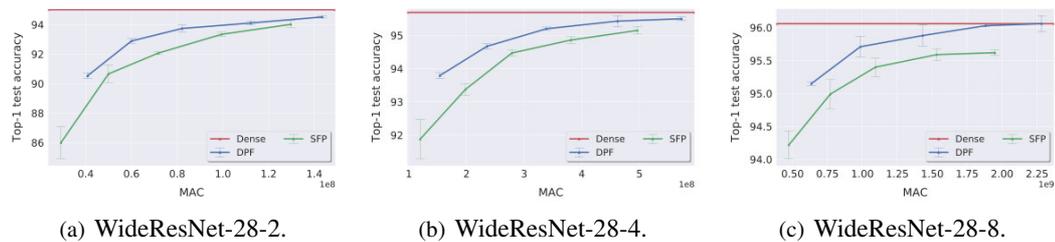


Figure 6.5 – Top-1 test accuracy for various target sparsity levels (on WideResNet-28-2 with CIFAR-10, unstructured pruning). DPF reaches comparable accuracy than the LT training method (and better for 99% target sparsity), but involves much less computation (**right y-axis, green**). Training the sparse models found by DPF from scratch does not reach the same performance (hence our sparse models are not lottery tickets).



(a) WideResNet-28-2.

(b) WideResNet-28-4.

(c) WideResNet-28-8.

Figure 6.6 – MAC v.s. top-1 test accuracy, for training WideResNet-28 (with various widths) on CIFAR-10. The reported results are averaged over three runs. The WideResNet-28-2 has 216M MACs, WideResNet-28-4 has 848M MACs and WideResNet-28-8 has 3366M MACs. Other detailed information refers to the Appendix 15.4.1, e.g. the # of params v.s. top-1 test accuracy in Figure 15.8, and the numerical test accuracy score in Table 15.3.

Appendix (Figure 15.1) for training ResNet-20 on CIFAR-10.

DPF does not find a lottery ticket. The LT hypothesis (Frankle and Carbin, 2019) conjectures that for every desired sparsity level there exists a sparse submodel that can be trained to the same or better performance as the dense model. In Figure 6.5 we show that the mask found by DPF is not a LT, i.e. training the obtained sparse model from scratch does not recover the same performance. The (expensive) procedure proposed in Frankle and Carbin (2019); Frankle et al. (2019) finds various masks and achieves the same performance as DPF for mild sparsity levels, but DPF is much better for extremely sparse models (99% sparsity).

Extension to structured pruning. The current state-of-the-art dynamic reparameterization meth-

ods only consider unstructured weight pruning. Structured filter pruning⁶ is either ignored (Bellec et al., 2018; Mocanu et al., 2018; Dettmers and Zettlemoyer, 2019) or shown to be challenging (Mostafa and Wang, 2019) even for the CIFAR dataset. In Table 15.3 below we presented some preliminary results on CIFAR-10 to show that our DPF can also be applied to structured filter pruning schemes. DPF *outperforms the current filter-norm based state-of-the-art method for structured pruning (e.g. SFP (He et al., 2018)) by a noticeable margin*. Figure 15.10 in Appendix 15.4.3 displays the transition procedure of the sparsity pattern (of layers) for WideResNet-28-2 with various sparsity levels. DPF can be seen as a particular neural architecture search method, as it gradually learns to prune entire layers under the guidance of the feedback signal.

We followed the common experimental setup as mentioned in Section 6.6 with ℓ_2 norm based filter selection criteria for structured pruning extension on CIFAR-10. We do believe a better filter selection scheme (Ye et al., 2018; He et al., 2019b; Lym et al., 2019) could further improve the results but we leave this exploration for the future work.

⁶ Lym et al. (2019) consider structured filter pruning and reconfigure the large (but sparse) model to small (but dense) model during the training for better training efficiency. Note that they perform model update on a gradually reduced model space, and it is completely different from the dynamic reparameterization methods (e.g. DSR, SM and our scheme) that perform reparameterization under original (full) model space.

7 Masking as an Efficient Alternative to Fine-tuning

7.1 Preface

Contribution and sources. This chapter reproduces [Zhao et al. \(2020b\)](#). The initial idea was proposed by the author. The author and Mengjie Zhao developed and implemented the algorithm, conducted extensive numerical experiments, and provided some empirical understanding. Detailed individual contributions:

Tao Lin (author): Conceptualization, Methodology (50 %), Software (50 %), Experiments (50 %), Writing—original draft preparation (50 %).

Mengjie Zhao: Methodology (50 %), Software (50 %), Experiments (50 %), Writing—original draft preparation (50 %).

Martin Jaggi: Supervision, Administration, Writing—review and editing.

Hinrich Schütze: Supervision, Administration, Writing—review and editing.

Summary. In this chapter, we present an efficient method of utilizing pretrained language models, where we learn selective binary masks for pretrained weights in lieu of modifying them through fine-tuning. Extensive evaluations of masking BERT, RoBERTa, and DistilBERT on eleven diverse NLP tasks show that our masking scheme yields performance comparable to fine-tuning, yet has a much smaller memory footprint when several tasks need to be inferred. Intrinsic evaluations show that representations computed by our binary masked language models encode information necessary for solving downstream tasks. Analyzing the loss landscape, we show that masking and fine-tuning produce models that reside in minima that can be connected by a line segment with nearly constant test accuracy. This confirms that masking can be utilized as an efficient alternative to fine-tuning.

7.2 Introduction

Fine-tuning a large pretrained language model like BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019c), and XLNet (Yang et al., 2019) often yields competitive or even state-of-the-art results on NLP benchmarks (Wang et al., 2018, 2019a). Given an NLP task, standard fine-tuning stacks a linear layer on top of the pretrained language model and then updates all parameters using mini-batch SGD. Various aspects like brittleness (Dodge et al., 2020) and adaptiveness (Peters et al., 2019) of this two-stage transfer learning NLP paradigm (Dai and Le, 2015; Howard and Ruder, 2018) have been studied.

Despite the simplicity and impressive performance of fine-tuning, the prohibitively large number of parameters to be fine-tuned, e.g. 340 million in BERT-large, is a major obstacle to wider deployment of these models. The large memory footprint of fine-tuned models becomes more prominent when multiple tasks need to be solved—several copies of the millions of fine-tuned parameters have to be saved for inference.

Recent work (Gaier and Ha, 2019; Zhou et al., 2019) points out the potential of searching neural architectures within a fixed model, as an alternative to optimizing the model weights for downstream tasks. Inspired by these results, we present *masking*, a simple yet efficient scheme for utilizing pretrained language models. Instead of directly updating the pretrained parameters, we propose to *select* weights important to downstream NLP tasks while *discarding* irrelevant ones. The selection mechanism consists of a set of binary masks, one learned per downstream task through end-to-end training.

We show that masking, when being applied to pretrained language models like BERT, RoBERTa, and DistilBERT (Sanh et al., 2019), achieves performance comparable to fine-tuning in tasks like part-of-speech tagging, named-entity recognition, sequence classification, and reading comprehension. This is surprising in that a simple subselection mechanism that does not change any weights is competitive with a training regime—fine-tuning—that can change the value of every single weight. We conduct detailed analyses revealing important factors and possible reasons for the desirable performance of masking.

Masking is parameter-efficient: only a set of 1-bit binary masks needs to be saved per task after training, instead of all 32-bit float parameters in fine-tuning. This small memory footprint enables deploying pretrained language models for solving multiple tasks on edge devices. The compactness of masking also naturally allows parameter-efficient ensembles of pretrained language models.

Our contributions: (i) We introduce *masking*, a new scheme for utilizing pretrained language models by learning selective masks for pretrained weights, as an efficient alternative to fine-tuning. We show that masking is applicable to models like BERT/RoBERTa/DistilBERT, and produces performance on par with fine-tuning. (ii) We carry out extensive empirical analysis of masking, shedding light on factors critical for achieving good performance on eleven diverse NLP tasks.

(iii) We study the binary masked language models’ loss landscape and language representations, revealing potential reasons why masking has task performance comparable to fine-tuning.

7.3 Related Work

Two-stage NLP paradigm. Pretrained language models (Peters et al., 2018; Devlin et al., 2019; Liu et al., 2019c; Yang et al., 2019; Radford et al., 2019) advance NLP with contextualized representation of words. Fine-tuning a pretrained language model (Dai and Le, 2015; Howard and Ruder, 2018) often delivers competitive performance partly because pretraining leads to a better initialization across various downstream tasks than training from scratch (Li et al., 2018). However, fine-tuning on individual NLP tasks is not parameter-efficient. Each fine-tuned model, typically consisting of hundreds of millions of floating point parameters, needs to be saved individually. Stickland and Murray (2019) use projected attention layers with multi-task learning to improve efficiency of fine-tuning BERT. Houlsby et al. (2019) insert adapter modules to BERT to improve memory efficiency. The inserted modules alter the forward pass of BERT, hence need to be carefully initialized to be close to identity.

We propose to directly pick parameters appropriate to a downstream task, by learning selective binary masks via end-to-end training. Keeping the pretrained parameters untouched, we solve several downstream NLP tasks with minimal overhead.

Binary networks and network pruning. Binary masks can be trained using the “straight-through estimator” (Bengio et al., 2013; Hinton, 2012). Hubara et al. (2016); Rastegari et al. (2016); Hubara et al. (2017), *inter alia*, apply this technique to train efficient binarized neural networks. We use this estimator to train selective masks for pretrained language model parameters.

Investigating the lottery ticket hypothesis (Frankle and Carbin, 2019) of network pruning (Han et al., 2015; He et al., 2018; Liu et al., 2019d; Lee et al., 2019; Lin et al., 2020c), Zhou et al. (2019) find that applying binary masks to a neural network is a form of training the network. Gaier and Ha (2019) propose to search neural architectures for reinforcement learning and image classification tasks, without any explicit weight training. This work inspires our masking scheme (which can be interpreted as implicit neural architecture search (Liu et al., 2019d)): applying the masks to a pretrained language model is similar to fine-tuning, yet is much more parameter-efficient.

Perhaps the closest work, Mallya et al. (2018) apply binary masks to CNNs and achieve good performance in computer vision. We learn selective binary masks for pretrained language models in NLP and shed light on factors important for obtaining good performance. Mallya et al. (2018) explicitly update weights in a task-specific classifier layer. In contrast, we show that end-to-end learning of selective masks, consistently for both the pretrained language model and a randomly initialized classifier layer, achieves good performance. Radiya-Dixit and Wang (2020) investigate

fine-tuning of BERT by employing a number of techniques, including what they call sparsification, a method similar to masking. Their focus is analysis of fine-tuning BERT whereas our goal is to provide an efficient alternative to fine-tuning.

7.4 Method

7.4.1 Background on Transformer and Fine-tuning

The encoder of the Transformer architecture (Vaswani et al., 2017) is ubiquitously used when pretraining large language models. We briefly review its architecture and then present our masking scheme. Taking BERT-base as an example, each one of the 12 transformer blocks consists of (i) four linear layers¹ \mathbf{W}_K , \mathbf{W}_Q , \mathbf{W}_V , and \mathbf{W}_{AO} for computing and outputting the self attention among input wordpieces (Wu et al., 2016). (ii) two linear layers \mathbf{W}_I and \mathbf{W}_O feeding forward the word representations to the next transformer block.

More concretely, consider an input sentence $\mathbf{X} \in \mathbb{R}^{N \times d}$ where N is the maximum sentence length and d is the hidden dimension size. \mathbf{W}_K , \mathbf{W}_Q , and \mathbf{W}_V are used to compute transformations of \mathbf{X} :

$$\mathbf{K} = \mathbf{X}\mathbf{W}_K, \mathbf{Q} = \mathbf{X}\mathbf{W}_Q, \mathbf{V} = \mathbf{X}\mathbf{W}_V,$$

and the self attention of \mathbf{X} is computed as:

$$\text{Attention}(\mathbf{K}, \mathbf{Q}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{K}\mathbf{Q}^T}{\sqrt{d}}\right)\mathbf{V}.$$

The attention is then transformed by \mathbf{W}_{AO} , and subsequently fed forward by \mathbf{W}_I and \mathbf{W}_O to the next transformer block.

When fine-tuning on a downstream task like sequence classification, a linear classifier layer \mathbf{W}_T , projecting from the hidden dimension to the output dimension, is randomly initialized. Next, \mathbf{W}_T is stacked on top of a pretrained linear layer \mathbf{W}_P (the *pooler layer*). All parameters are then updated to minimize the task loss such as cross-entropy.

7.4.2 Learning the Mask

Given a pretrained language model, we do not fine-tune, i.e. we do not update the pretrained parameters. Instead, we *select* a subset of the pretrained parameters that is critical to a downstream task while *discarding* irrelevant ones with binary masks. We associate each linear layer $\mathbf{W}^l \in \{\mathbf{W}_K^l, \mathbf{W}_Q^l, \mathbf{W}_V^l, \mathbf{W}_{AO}^l, \mathbf{W}_I^l, \mathbf{W}_O^l\}$ of the l -th transformer block with a real-valued matrix \mathbf{M}^l that is randomly initialized from a uniform distribution and has the same size as \mathbf{W}^l . We then pass \mathbf{M}^l through an element-wise thresholding function (Hubara et al., 2016; Mallya et al., 2018), i.e. a

¹We omit the bias terms for brevity.

binarizer, to obtain a binary mask $\mathbf{M}_{\text{bin}}^l$ for \mathbf{W}^l :

$$(m_{\text{bin}}^l)_{i,j} = \begin{cases} 1 & \text{if } m_{i,j}^l \geq \tau \\ 0 & \text{otherwise} \end{cases}, \quad (7.1)$$

where $m_{i,j}^l \in \mathbf{M}^l$, i, j indicate the coordinates of the 2-D linear layer and τ is a global thresholding hyperparameter.

In each forward pass of training, the binary mask $\mathbf{M}_{\text{bin}}^l$ (derived from \mathbf{M}^l via (7.1)) selects weights in a pretrained linear layer \mathbf{W}^l by Hadamard product:

$$\hat{\mathbf{W}}^l := \mathbf{W}^l \odot \mathbf{M}_{\text{bin}}^l.$$

In the corresponding backward pass of training, with the associated loss function \mathcal{L} , we cannot back-propagate through the binarizer, since (7.1) is a hard thresholding operation and the gradient with respect to \mathbf{M}^l is zero almost everywhere. Similar to the treatment² in Bengio et al. (2013); Hubara et al. (2016); Lin et al. (2020c), we use $\frac{\partial \mathcal{L}(\hat{\mathbf{W}}^l)}{\partial \mathbf{M}_{\text{bin}}^l}$ as a noisy estimator of $\frac{\partial \mathcal{L}(\hat{\mathbf{W}}^l)}{\partial \mathbf{M}^l}$ to update \mathbf{M}^l , i.e.:

$$\mathbf{M}^l \leftarrow \mathbf{M}^l - \eta \frac{\partial \mathcal{L}(\hat{\mathbf{W}}^l)}{\partial \mathbf{M}_{\text{bin}}^l}, \quad (7.2)$$

where η refers to the step size. Hence, the whole structure can be trained end-to-end.

We learn a set of binary masks for an NLP task as follows. Recall that each linear layer \mathbf{W}^l is associated with a \mathbf{M}^l to obtain a masked linear layer $\hat{\mathbf{W}}^l$ through (7.1). We randomly initialize an additional linear layer with an associated \mathbf{M}^l and stack it on top of the pretrained language model. We then update each \mathbf{M}^l through (7.2) with the task objective during training.

After training, we pass each \mathbf{M}^l through the binarizer to obtain $\mathbf{M}_{\text{bin}}^l$, which is then saved for future inference. Since $\mathbf{M}_{\text{bin}}^l$ is binary, it takes only $\approx 3\%$ of the memory compared to saving the 32-bit float parameters in a fine-tuned model. Also, we will show that many layers—in particular the embedding layer—do not have to be masked. This further reduces memory consumption of masking.

7.4.3 Configuration of Masking

Our masking scheme is motivated by the observation: the pretrained weights form a good initialization (Li et al., 2018), yet a few steps of adaptation are still needed to produce competitive performance for a specific task. However, not every pretrained parameter is necessary for achieving reasonable performance, as suggested by the field of neural network pruning (LeCun

² Bengio et al. (2013); Hubara et al. (2016) describe it as the “straight-through estimator”, and Lin et al. (2020c) (as in Chapter 6) provide convergence guarantee with error feedback interpretation.

et al., 1989; Hassibi and Stork, 1992; Han et al., 2015). We now investigate two configuration choices that affect how many parameters are “eligible” for masking.

Initial sparsity of M_{bin}^l . As we randomly initialize our masks from uniform distributions, the sparsity of the binary mask M_{bin}^l in the mask initialization phase controls how many pretrained parameters in a layer W^l are assumed to be irrelevant to the downstream task. Various initial sparsity rates entail diverse optimization behaviors.

It is crucial to better understand how the initial sparsity of a mask impacts the training dynamics and final model performance, so as to generalize our masking scheme to broader domains and tasks. In Section 7.6.1, we investigate this aspect in detail. In practice, we fix τ in (7.1) while adjusting the uniform distribution to achieve a target initial sparsity.

Which layers to mask. Various layers of pretrained language models capture distinct aspects of a language during pretraining, e.g. Tenney et al. (2019) find that information on part-of-speech tagging, parsing, named-entity recognition, semantic roles, and coreference is encoded on progressively higher layers of BERT. It is hard to know a priori which types of NLP tasks have to be addressed in the future, making it non-trivial to decide layers to mask. We study this factor in Section 7.6.2.

We do not learn a mask for the lowest embedding layer, i.e. the uncontextualized wordpiece embeddings are completely “selected”, for all tasks. The motivation is two-fold. (i) The embedding layer weights take up a large part, e.g. almost 21% (23m/109m) in BERT-base-uncased, of the total number of parameters. Not having to learn a selective mask for this layer reduces memory consumption. (ii) Pretraining has effectively encoded context-independent general meanings of words in the embedding layer (Zhao et al., 2020a). Hence, learning a selective mask for this layer is unnecessary. Also, we do not learn masks for biases and layer normalization parameters as we did not observe a positive effect on performance.

7.5 Datasets and Experimental Setup

Datasets. We present results for masking BERT, RoBERTa, and DistilBERT in part-of-speech tagging, named-entity recognition, sequence classification, and reading comprehension.

We experiment with **part-of-speech tagging** (POS) on Penn Treebank (Marcinkiewicz, 1994), using Collins (2002)’s train/dev/test split. For **named-entity recognition** (NER), we conduct experiments on the CoNLL-2003 NER shared task (Tjong Kim Sang and De Meulder, 2003).

For **sequence classification**, the following GLUE tasks (Wang et al., 2018) are evaluated: Stanford Sentiment Treebank (SST2) (Socher et al., 2013), Microsoft Research Paraphrase Corpus (MRPC) (Dolan and Brockett, 2005), Corpus of Linguistic Acceptability (CoLA) (Warstadt

et al., 2019), Recognizing Textual Entailment (RTE) (Dagan et al., 2005), and Question Natural Language Inference (QNLI) (Rajpurkar et al., 2016).

In addition, we experiment on sequence classification datasets that have publicly available test sets: the 6-class question classification dataset TREC (Voorhees et al., 1999), the 4-class news classification dataset AG News (AG) (Zhang et al., 2015), and the binary Twitter sentiment classification task SemEval-2016 4B (SEM) (Nakov et al., 2016).

We experiment with **reading comprehension** on SWAG (Zellers et al., 2018) using the official data splits. We report Matthew’s correlation coefficient (MCC) for CoLA, micro-F1 for NER, and accuracy for the other tasks.

Setup. Due to resource limitations and in the spirit of environmental responsibility (Strubell et al., 2019; Schwartz et al., 2020), we conduct our experiments on the base models: BERT-base-uncased, RoBERTa-base, and DistilBERT-base-uncased. Thus, the BERT/RoBERTa models we use have 12 transformer blocks (0–11 indexed) producing 768-dimension vectors; the DistilBERT model we use has the same dimension but contains 6 transformer blocks (0–5 indexed). We implement our models in PyTorch (Paszke et al., 2017) with the HuggingFace framework (Wolf et al., 2019).

Throughout all experiments, we limit the maximum length of a sentence (pair) to be 128 after wordpiece tokenization. Following Devlin et al. (2019), we use the Adam (Kingma and Ba, 2015) optimizer of which the learning rate is a hyperparameter while the other parameters remain default. We carefully tune the learning rate for each setup: the tuning procedure ensures that the best learning rate does not lie on the border of our search grid, otherwise we extend the grid accordingly. The initial grid is {1e-5, 3e-5, 5e-5, 7e-5, 9e-5}.

For sequence classification and reading comprehension, we use [CLS] as the representation of the sentence (pair). Following Devlin et al. (2019), we formulate NER as a tagging task and use a linear output layer, instead of a conditional random field layer. For POS and NER experiments, the representation of a tokenized word is its last wordpiec (Liu et al., 2019b; He and Choi, 2020). Note that a 128 maximum length of a sentence for POS and NER means that some word-tag annotations need to be excluded.

7.6 Experiments

7.6.1 Initial Sparsity of Binary Masks

We first investigate how initial sparsity percentage (i.e. fraction of zeros) of the binary mask $\mathbf{M}_{\text{bin}}^l$ influences performance of a binary masked language model on downstream tasks. We experiment on four tasks, with initial sparsities in {1%, 3%, 5%, 10%, 15%, 20%, ..., 95%}. All other hyperparameters are controlled: learning rate is fixed to 5e-5; batch size is 32 for relatively small datasets (RTE, MRPC, and CoLA) and 128 for SST2. Each experiment is repeated four times

Chapter 7. Masking as an Efficient Alternative to Fine-tuning

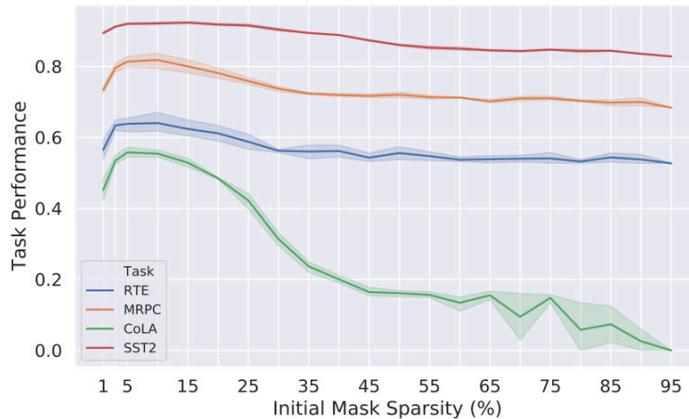


Figure 7.1 – Dev set performance of masking BERT when selecting various amounts of pretrained parameters.

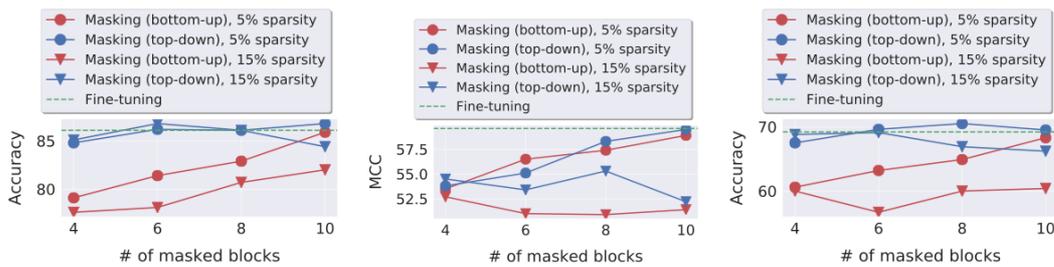


Figure 7.2 – The impact of masking varied transformer blocks of BERT for MRPC (left), CoLA (middle), and RTE (right). The number of masked blocks is shown on the x-axis; that number is either masked “bottom-up” or “top-down”. More precisely, a bottom-up setup (red) masking 4 blocks means we mask the transformer blocks $\{0, 1, 2, 3\}$; a top-down setup (blue) masking 4 blocks means we mask the transformer blocks $\{8, 9, 10, 11\}$. \mathbf{W}_P and \mathbf{W}_T are always masked.

with various random seeds $\{1, 2, 3, 4\}$. In this experiment, all transformer blocks, the pooler layer, and the classifier layer are masked.

Figure 7.1 shows that masking achieves decent performance without hyperparameter search. Specifically, (i) a large initial sparsity removing most pretrained parameters, e.g. 95%, leads to bad performance for the four tasks. This is due to the fact that the pretrained knowledge is largely discarded. (ii) Gradually decreasing the initial sparsity improves task performance. Generally, an initial sparsity in 3% ~ 10% yields reasonable results across tasks. Large datasets like SST2 are less sensitive than small datasets like RTE. (iii) Selecting almost all pretrained parameters, e.g. 1% sparsity, hurts task performance. Recall that a pretrained model needs to be adapted to a downstream task; masking achieves adaptation by learning selective masks—preserving too many pretrained parameters in initialization impedes the optimization.

7.6.2 Layer-wise Behaviors

Neural network layers present heterogeneous characteristics (Zhang et al., 2019a) when being applied to tasks. For example, syntactic information is better represented at lower layers while semantic information is captured at higher layers in ELMo (Peters et al., 2018). As a result, simply masking all transformer blocks (as in Section 7.6.1) may not be ideal.

We investigate the task performance when applying various choices of masks to BERT layers. Figure 7.2 presents the optimal task performance when masking only a subset of BERT’s transformer blocks on MRPC, CoLA, and RTE. Various amounts and indices of transformer blocks are masked: “bottom-up” and “top-down” indicate to mask the targeted amount of transformer blocks, either from bottom or top of BERT.

We can observe that (i) in most cases, top-down masking outperforms bottom-up masking when initial sparsity and the number of masked layers are fixed. Thus, it is reasonable to select all pretrained weights in lower layers, since they capture general information helpful and transferable to various tasks (Liu et al., 2019b; Howard and Ruder, 2018). (ii) For bottom-up masking, increasing the number of masked layers gradually improves performance. This observation illustrates dependencies between BERT layers and the learning dynamics of masking: provided with selected pretrained weights in lower layers, higher layers need to be given flexibility to select pretrained weights accordingly to achieve good task performance. (iii) In top-down masking, CoLA performance increases when masking a growing number of layers while MRPC and RTE are not sensitive. Recall that CoLA tests linguistic acceptability that typically requires both syntactic and semantic information.³ All of BERT layers are involved in representing this information, hence allowing more layers to change should improve performance.

7.6.3 Comparing Fine-tuning and Masking

We have investigated two factors—initial sparsity (Section 7.6.1) and layer-wise behaviors (7.6.2)—that are important in masking pretrained language models. Here, we compare the performance and memory consumption of masking and fine-tuning.

Based on observations in Section 7.6.1 and Section 7.6.2, we use 5% initial sparsity when applying masking to BERT, RoBERTa, and DistilBERT. We mask the transformer blocks 2–11 in BERT/RoBERTa and 2–5 in DistilBERT. \mathbf{W}_P and \mathbf{W}_T are always masked. Note that this global setup is surely suboptimal for some model-task combinations, but our goal is to illustrate the effectiveness and the generalization ability of masking. Hence, conducting extensive hyperparameter search is unnecessary.

For AG and QNLI, we use batch size 128. For the other tasks we use batch size 32. We search the

³For example, to distinguish acceptable caused-motion constructions (e.g. “the professor talked us into a stupor”) from unacceptable ones (e.g. “water talked it into red”), both syntactic and semantic information need to be considered (Goldberg, 1995).

Chapter 7. Masking as an Efficient Alternative to Fine-tuning

Table 7.1 – Dev set task performances (%) of masking and fine-tuning. Each experiment is repeated four times with various random seeds and we report mean and standard deviation. Numbers below dataset name (second row) are the size of training set. For POS and NER, we report the number of sentences.

		MRPC (3.5k)	SST2 (67k)	CoLA (8.5k)	RTE (2.5k)	QNLI (108k)	SEM (4.3k)	TREC (4.9k)	AG (96k)	POS (38k)	NER (15k)	SWAG (113k)
BERT	Fine-tuning	86.1 ± 0.8	93.3 ± 0.2	59.6 ± 0.8	69.2 ± 2.7	91.0 ± 0.6	86.6 ± 0.3	96.4 ± 0.2	94.4 ± 0.1	97.7 ± 0.0	94.6 ± 0.2	80.9 ± 1.7
	Masking	86.8 ± 1.1	93.2 ± 0.5	59.5 ± 0.1	69.5 ± 3.0	91.3 ± 0.4	85.9 ± 0.5	96.0 ± 0.4	94.2 ± 0.0	97.7 ± 0.0	94.5 ± 0.1	80.3 ± 0.1
RoBERTa	Fine-tuning	89.8 ± 0.5	95.0 ± 0.3	62.1 ± 1.7	78.2 ± 1.1	92.9 ± 0.2	90.2 ± 0.5	96.2 ± 0.4	94.7 ± 0.0	98.1 ± 0.0	94.9 ± 0.1	83.4 ± 0.8
	Masking	88.5 ± 1.1	94.5 ± 0.3	60.3 ± 1.3	69.2 ± 2.1	92.4 ± 0.1	90.1 ± 0.1	95.9 ± 0.5	94.5 ± 0.1	98.0 ± 0.0	93.9 ± 0.1	82.1 ± 0.2
DistilBERT	Fine-tuning	85.4 ± 0.5	91.6 ± 0.4	55.1 ± 0.3	62.2 ± 3.0	89.0 ± 0.8	85.9 ± 0.2	95.7 ± 0.6	94.2 ± 0.1	97.6 ± 0.0	94.1 ± 0.1	72.5 ± 0.2
	Masking	86.0 ± 0.3	91.3 ± 0.3	53.1 ± 0.7	61.6 ± 1.5	89.2 ± 0.2	86.6 ± 0.6	95.9 ± 0.6	94.2 ± 0.1	97.6 ± 0.0	94.1 ± 0.2	71.0 ± 0.0

Table 7.2 – Error rate (%) on test set and model size comparison. Single: the averaged performance of four models with various random seeds. Ensemble: ensemble of the four models.

		SEM	TREC	AG	POS	NER	Memory (MB)
Masking	Single	12.03	3.30	5.62	2.34	9.85	447
	Ensemble	11.52	3.20	5.28	2.12	9.19	474
Fine-tuning	Single	11.87	3.80	5.66	2.34	9.85	438
	Ensemble	11.73	2.80	5.17	2.29	9.23	1752
Sun et al. (2019)		n/a	2.80	5.25	n/a	n/a	n/a
Palogiannidi et al. (2016)		13.80	n/a	n/a	n/a	n/a	n/a

optimal learning rate per task as described in Section 7.5, and they are shown in Appendix 16.1.4.

Performance Comparison. Table 7.1 reports performance of masking and fine-tuning on the dev set for the eleven NLP tasks. We observe that applying masking to BERT/RoBERTa/DistilBERT yields performance comparable to fine-tuning. We observe a performance drop⁴ on RoBERTa-RTE. RTE has the smallest dataset size (train: 2.5k; dev: 0.3k) among all tasks—this may contribute to the imperfect results and large variances.

Our BERT-NER results are slightly worse than Devlin et al. (2019). This may be due to the fact that “maximal document context” is used by Devlin et al. (2019) while we use sentence-level context of 128 maximum sequence length.⁵

Rows “Single” in Table 7.2 compare performance of masking and fine-tuning BERT on the test set of SEM, TREC, AG, POS, and NER. The same setup and hyperparameter searching as Table 7.1 are used, the best hyperparameters are picked on the dev set. Results from Sun et al. (2019); Palogiannidi et al. (2016) are included as a reference. Sun et al. (2019) employ optimizations like layer-wise learning rate, producing slightly better performance than ours. Palogiannidi et al. (2016) is the best performing system on task SEM (Nakov et al., 2016). Again, masking yields results comparable to fine-tuning.

⁴ Similar observations were made: DistilBERT has a 10% accuracy drop on RTE compared to BERT-base (Sanh et al., 2019); Sajjad et al. (2020) report unstableness on MRPC and RTE when applying their model reduction strategies

⁵ Similar observations were made: <https://github.com/huggingface/transformers/issues/64>

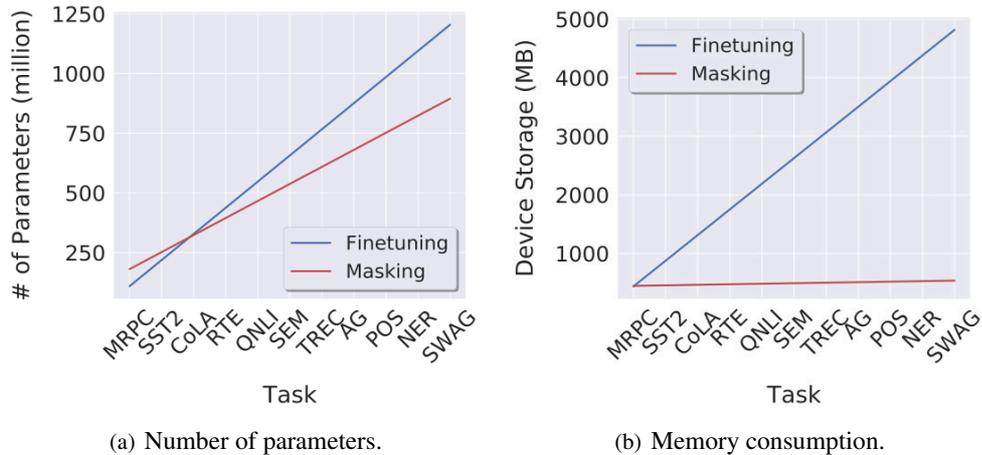


Figure 7.3 – The accumulated number of parameters and memory required by fine-tuning and masking to solve an increasing number of tasks.

Memory comparison. Having shown that task performance of masking and fine-tuning is comparable, we next demonstrate one key strength of masking: memory efficiency. We take BERT-base-uncased as our example. Figure 7.3 shows the accumulated number of parameters in million and memory in megabytes (MB) required when an increasing number of downstream tasks need to be solved using fine-tuning and masking. Masking requires a small overhead when solving a single task but is much more efficient than fine-tuning when several tasks need to be inferred. Masking saves a single copy of a pretrained language model containing 32-bit float parameters for all the eleven tasks and a set of 1-bit binary masks for each task. In contrast, fine-tuning saves every fine-tuned model so the memory consumption grows linearly.

Masking naturally allows light ensembles of models. Rows “Ensem.” in Table 7.2 compare ensembled results and model size. We consider the ensemble of predicted (i) labels; (ii) logits; (iii) probabilities. The best ensemble method is picked on dev and then evaluated on test. Masking only consumes 474MB of memory—much smaller than 1752MB required by fine-tuning—and achieves comparable performance. Thus, masking is also much more memory-efficient than fine-tuning in an ensemble setting.

7.7 Discussion

7.7.1 Intrinsic Evaluations

Section 7.6 demonstrates that masking is an efficient alternative to fine-tuning. Now we analyze properties of the representations computed by binary masked language models with intrinsic evaluation.

One intriguing property of fine-tuning, i.e. stacking a classifier layer on top of a pretrained language model then update all parameters, is that a linear classifier layer suffices to conduct

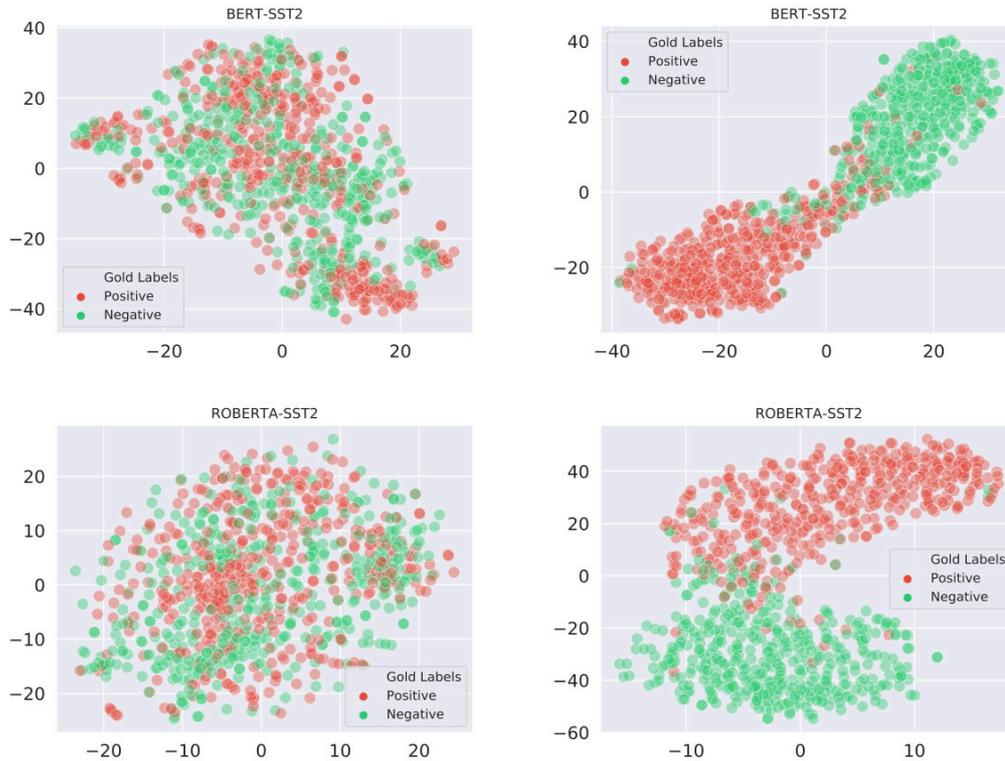


Figure 7.4 – t-SNE visualization of the representation of [CLS] computed by the topmost transformer block in pretrained (left), fine-tuned (top right), and masked (bottom right) BERT/RoBERTa. We use `scikit-learn` (Pedregosa et al., 2011) and default t-SNE parameters.

reasonably accurate classification. This observation implies that the configuration of data points, e.g. sentences with positive or negative sentiment in SST2, should be close to linearly separable in the hidden space. Like fine-tuning, masking also uses a linear classifier layer. Hence, we hypothesize that upper layers in binary masked language models, even without explicit weight updating, also create a hidden space in which data points are close to linearly separable.

Figure 7.4 uses t-SNE (Maaten and Hinton, 2008) to visualize the representation of [CLS] computed by the topmost transformer block in pretrained, fine-tuned, and masked BERT/RoBERTa, using the dev set examples of SST2. The pretrained models’ representations (left) are clearly not separable since the model needs to be adapted to downstream tasks. The sentence representations computed by the fine-tuned (top right) and the binary masked (bottom right) encoder are almost linearly separable and consistent with the gold labels. Thus, a linear classifier is expected to yield reasonably good classification accuracy. This intrinsic evaluation illustrates that binary masked models extract good representations from the data for the downstream NLP task.

Table 7.3 – Generalization on dev (%) of binary masked and fine-tuned BERT. Row: training dataset; Column: evaluating dataset. Numbers are improvements against the majority-vote baseline: 50.9 for SST2 and 74.4 for SEM. Results are averaged across four random seeds.

	SST2	SEM		SST2	SEM
SST2	41.8	-13.4	SST2	41.8	-10.1
SEM	20.0	11.5	SEM	18.9	12.2

(a) Masking (b) Fine-tuning

7.7.2 Properties of the Binary Masked Models

Do binary masked models generalize? Figure 7.4 shows that a binary masked language model produces proper representations for the classifier layer and hence performs as well as a fine-tuned model. Here, we are interested in verifying that the binary masked model does indeed solve downstream tasks by learning meaningful representations—instead of exploiting spurious correlations that generalize poorly (Niven and Kao, 2019; McCoy et al., 2019). To this end, we test if the binary masked mode is generalizable to other datasets of the same type of downstream task. We use the two sentiment classification datasets: SST2 and SEM. We simply evaluate the model masked or fine-tuned on SST2 against the dev set of SEM and vice versa. Table 7.3 reports the results against the majority-vote baseline. The fine-tuned and binary masked models of SEM generalize well on SST2, showing $\approx 20\%$ improvement against the majority-vote baseline.

On the other hand, we observe that the knowledge learned on SST2 does not generalize to SEM, for both fine-tuning and masking. We hypothesize that this is because the Twitter domain (SEM) is much more specific than movie reviews (SST2). For example, some Emojis or symbols like “:)” reflecting strong sentiment do not occur in SST2, resulting in unsuccessful generalization. To test our hypothesis, we take another movie review dataset IMDB (Maas et al., 2011), and directly apply the SST2-fine-tuned- and SST2-binary-masked- models on it. Masking and fine-tuning achieve accuracy 84.79% and 85.25%, which are comparable and both outperform the baseline 50%, demonstrating successful knowledge transfer.

Thus, fine-tuning and masking yield models with similar generalization ability. The binary masked models indeed create representations that contain valid information for downstream tasks.

Analyzing masks. We study the dissimilarity between masks learned by various BERT layers and downstream tasks. For the initial and trained binary masks $\mathbf{M}_{\text{bin}}^{t,init}$ and $\mathbf{M}_{\text{bin}}^{t,trained}$ of a layer trained on task $t \in \{t1, t2\}$. We compute:

$$S = \frac{\|\mathbf{M}_{\text{bin}}^{t1,trained} - \mathbf{M}_{\text{bin}}^{t2,trained}\|_1}{\|\mathbf{M}_{\text{bin}}^{t1,trained} - \mathbf{M}_{\text{bin}}^{t1,init}\|_1 + \|\mathbf{M}_{\text{bin}}^{t2,trained} - \mathbf{M}_{\text{bin}}^{t2,init}\|_1},$$

Chapter 7. Masking as an Efficient Alternative to Fine-tuning

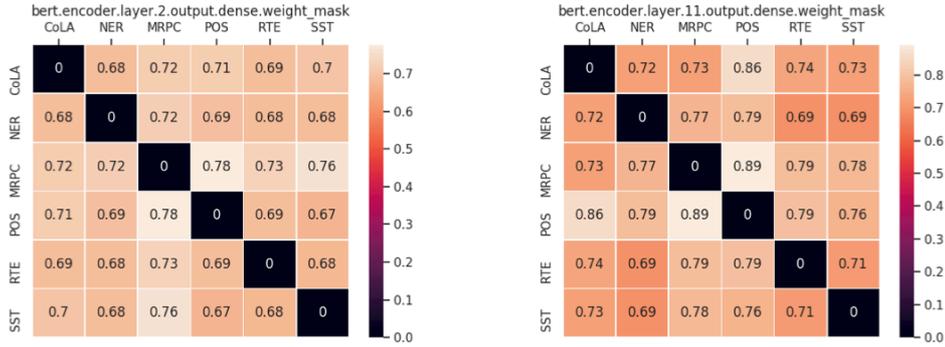


Figure 7.5 – Scores s of two sets of masks, trained with two different tasks, of layer \mathbf{W}_O in transformer blocks 2 (left) and 11 (right) in BERT. A large s means that the two masks are dissimilar.

where $\|\mathbf{W}\|_1 = \sum_{i=1}^m \sum_{j=1}^n |w_{i,j}|$. Note that for the same random seed, $\mathbf{M}_{\text{bin}}^{t1,init}$ and $\mathbf{M}_{\text{bin}}^{t2,init}$ are the same. The dissimilarity s measures the difference between two masks as a fraction of all changes brought about by training. Figure 7.5 shows that, after training, the dissimilarities of masks of higher BERT layers are larger than those of lower BERT layers. Similar observations are made for fine-tuning: top layer weights in fine-tuned BERT are more task-specific (Kovaleva et al., 2019). The figure also shows that the learned masks for downstream tasks tend to be dissimilar to each other, even for similar tasks. For a given task, there exist various sets of masks (initialized with various random seeds) yielding similar performance. This observation is similar to the results of evaluating the lottery ticket hypothesis on BERT (Prasanna et al., 2020; Chen et al., 2020): a number of subnetworks exist in BERT achieving similar task performance.

7.7.3 Loss Landscape

Training complex neural networks can be viewed as searching for good minima in the highly non-convex landscape defined by the loss function (Li et al., 2018). Good minima are typically depicted as points at the bottom of different locally convex valleys (Keskar et al., 2017; Draxler et al., 2018), achieving similar performance. In this section, we study the relationship between the two minima obtained by masking and fine-tuning.

Recent work analyzing the loss landscape suggests that the local minima in the loss landscape reached by standard training algorithms can be connected by a simple path (Garipov et al., 2018; Gotmare et al., 2019), e.g. a Bézier curve, with low task loss (or high task accuracy) along the path. We are interested in testing if the two minima found by fine-tuning and masking can be easily connected in the loss landscape. To start with, we verify the task performance of an interpolated model $\mathbf{W}(\gamma)$ on the line segment between a fine-tuned model \mathbf{W}_0 and a binary masked model \mathbf{W}_1 :

$$\mathbf{W}(\gamma) = \mathbf{W}_0 + \gamma(\mathbf{W}_1 - \mathbf{W}_0), 0 \leq \gamma \leq 1.$$

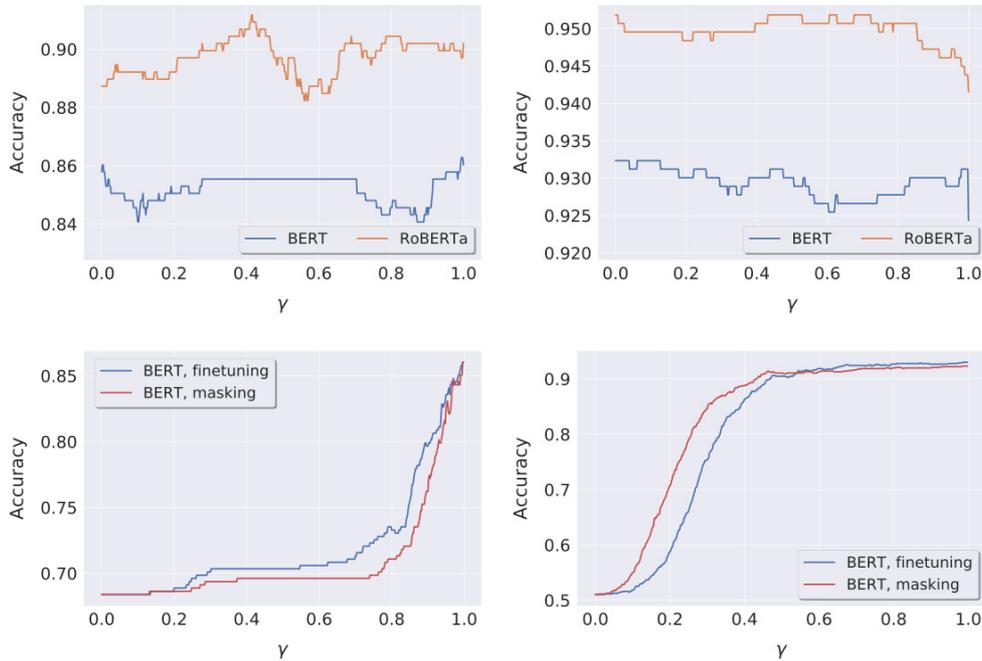


Figure 7.6 – Mode connectivity results on MRPC (left) and SST2 (right). Top images: dev set accuracy of an interpolated model between the two minima found by fine-tuning ($\gamma=0$) and masking ($\gamma=1$). Bottom images: accuracy of an interpolated model between pre-trained ($\gamma=0$) and fine-tuned/masked ($\gamma=1$) BERT.

We conduct experiments on MRPC and SST2 with the best-performing BERT and RoBERTa models obtained in Table 7.1 (same seed and training epochs); Figure 7.6 (top) shows the results of mode connectivity, i.e. the evolution of the task accuracy along a line connecting the two candidate minima.

Surprisingly, the interpolated models on the line segment connecting a fine-tuned and a binary masked model form a high accuracy path, indicating the extremely well-connected loss landscape. Thus, masking finds minima on the same connected low-loss manifold as fine-tuning, confirming the effectiveness of our method. Also, we show in Figure 7.6 (bottom) for the line segment between the pre-trained BERT and a fine-tuned/masked BERT, that mode connectivity is not solely due to an overparameterized pre-trained language model. Bézier curves experiments show similar results, c.f. Section 16.2.

7.8 Conclusion

We have presented masking, an efficient alternative to fine-tuning for utilizing pretrained language models like BERT/RoBERTa/DistilBERT. Instead of updating the pretrained parameters, we only train one set of binary masks per task to select critical parameters. Extensive experiments show that masking yields performance comparable to fine-tuning on a series of NLP tasks. Leaving the pretrained parameters unchanged, masking is much more memory efficient when several

Chapter 7. Masking as an Efficient Alternative to Fine-tuning

tasks need to be solved. Intrinsic evaluations show that binary masked models extract valid and generalizable representations for downstream tasks. Moreover, we demonstrate that the minima obtained by fine-tuning and masking can be easily connected by a line segment, confirming the effectiveness of applying masking to pretrained language models. Our code is available at: <https://github.com/ptlmasking/maskbert>.

Future work may explore the possibility of applying masking to the pretrained multilingual encoders like mBERT (Devlin et al., 2019) and XLM (Conneau and Lample, 2019). Also, the binary masks learned by our method have low sparsity such that inference speed is not improved. Developing methods improving both memory and inference efficiency without sacrificing task performance can open the possibility of widely deploying the powerful pretrained language models to more NLP applications.

Robustness to Heterogeneous **Part III** **Environments**

8 Ensemble Distillation for Robust Model Fusion in Federated Learning

8.1 Preface

Contribution and sources. This chapter builds on the work done in [Lin et al. \(2020b\)](#). The initial idea, development and implementation of the algorithm, and experiment design, were carried out mostly by the author. Lingjing Kong contributed to the implementation of algorithms, generalization bounds analysis, and writing. Detailed individual contributions:

Tao Lin (author): Conceptualization, Methodology, Software (70 %), Experiments (50 %), Analysis (50 %), Writing—original draft preparation (50 %).

Lingjing Kong: Software (30 %), Experiments (50 %), Analysis (50 %), Writing—original draft preparation (50 %).

Sebastian U. Stich: Writing—review and editing.

Martin Jaggi: Supervision, Administration, Writing—review and editing.

Summary. Federated Learning (FL) is a machine learning setting where many devices collaboratively train a machine learning model while keeping the training data decentralized. In most of the current training schemes the central model is refined by averaging the parameters of the server model and the updated parameters from the client side. However, directly averaging model parameters is only possible if all models have the same structure and size, which could be a restrictive constraint in many scenarios.

In this chapter, we investigate more powerful and more flexible aggregation schemes for FL. Specifically, we propose ensemble distillation for model fusion, i.e. training the central classifier through unlabeled data on the outputs of the models from the clients. This knowledge distillation technique mitigates privacy risk and cost to the same extent as the baseline FL algorithms, but allows flexible aggregation over heterogeneous client models that can vary e.g. in size, numerical precision or structure. We show in extensive empirical experiments on various CV/NLP datasets (CIFAR-10/100, ImageNet, AG News, SST2) and settings (heterogeneous models/data) that the server model can be trained much faster, requiring fewer communication rounds than any existing

FL technique so far.

8.2 Introduction

Federated Learning (FL) has emerged as an important machine learning paradigm in which a federation of clients participate in collaborative training of a centralized model (Shokri and Shmatikov, 2015; McMahan et al., 2017; Smith et al., 2017; Caldas et al., 2018; Bonawitz et al., 2019; Li et al., 2020a; Kairouz et al., 2021). The clients send their model parameters to the server but never their private training datasets, thereby ensuring a basic level of privacy. Among the key challenges in federated training are communication overheads and delays (one would like to train the central model with as few communication rounds as possible), and client heterogeneity: the training data (non-i.i.d.-ness), as well as hardware and computing resources, can change drastically among clients, for instance when training on commodity mobile devices.

Classic training algorithms in FL, such as federated averaging (FEDAVG) (McMahan et al., 2017) and its recent adaptations (Mohri et al., 2019; Li et al., 2020c; Hsu et al., 2019; Karimireddy et al., 2020; Hsu et al., 2020; Reddi et al., 2021), are all based on directly averaging of the participating client’s *parameters* and can hence only be applied if all client’s models have the same size and structure. In contrast, ensemble learning methods (You et al., 2017a; Furlanello et al., 2018; Anil et al., 2018; Dvornik et al., 2019; Park and Kwak, 2019; Liu et al., 2019a; Wu et al., 2019) allow to combine multiple heterogeneous weak classifiers by averaging the *predictions* of the individual models instead. However, applying ensemble learning techniques directly in FL is infeasible in practice due to the large number of participating clients, as it requires keeping weights of all received models on the server and performing naive ensembling (logits averaging) for inference.

To enable federated learning in more realistic settings, we propose to use ensemble distillation (Bucilua et al., 2006; Hinton et al., 2015) for robust model fusion (FedDF). Our scheme leverages unlabeled data or artificially generated examples (e.g. by a GAN’s generator (Goodfellow et al., 2014)) to aggregate knowledge from all received (heterogeneous) client models. We demonstrate with thorough empirical results that our ensemble distillation approach not only addresses the existing quality loss issue (Hsieh et al., 2020) of Batch Normalization (BN) (Ioffe and Szegedy, 2015) for networks in a homogeneous FL system, but can also break the knowledge barriers among heterogeneous client models. Our main contributions are:

- We propose a distillation framework for robust federated model fusion, which allows for heterogeneous client models and data, and is robust to the choices of neural architectures.
- We show in extensive numerical experiments on various CV/NLP datasets (CIFAR-10/100, ImageNet, AG News, SST2) and settings (heterogeneous models and/or data) that the server model can be trained much faster, requiring fewer communication rounds than any existing FL technique.

We further provide insights on when FedDF can outperform FEDAVG (see also Fig. 8.1 that

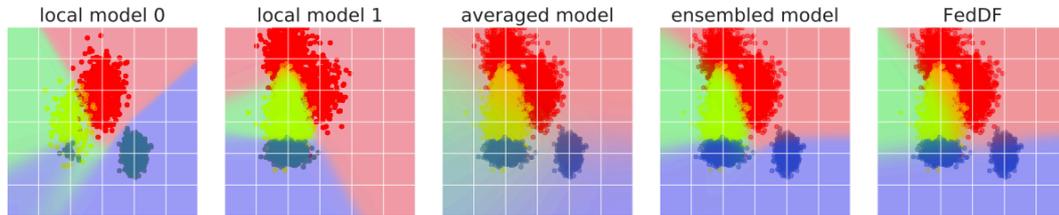


Figure 8.1 – **Limitations of FEDAVG.** We consider a toy example of a 3-class classification task with a 3-layer MLP, and display the decision boundaries (probabilities over RGB channels) on the input space. The left two figures show the individually trained local models. The right three figures evaluate aggregated models and the global data distribution; the averaged model results in much blurred decision boundaries. The used datasets are displayed in Figure 17.2 (Appendix 17.3.1).

highlights an intrinsic limitation of parameter averaging based approaches) and what factors influence FedDF.

8.3 Related Work

Federated learning. The classic algorithm in FL, FEDAVG (McMahan et al., 2017), or local SGD (Lin et al., 2020d) when all devices are participating, performs weighted parameter average over the client models after several local SGD updates with weights proportional to the size of each client’s local data. Weighting schemes based on client loss are investigated in Mohri et al. (2019); Li et al. (2020c). To address the difficulty of directly averaging model parameters, Singh and Jaggi (2020); Wang et al. (2020a) propose to use optimal transport and other alignment schemes to first align or match individual neurons of the neural nets layer-wise before averaging the parameters. However, these layer-based alignment schemes necessitate client models with the same number of layers and structure, which is restrictive in heterogeneous systems in practice. Another line of work aims to improve local client training, i.e. client-drift problem caused by the heterogeneity of local data (Li et al., 2020b; Karimireddy et al., 2020). For example, FEDPROX (Li et al., 2020b) incorporates a proximal term for the local training. Other techniques like acceleration, recently appear in Hsu et al. (2019, 2020); Reddi et al. (2021).

Knowledge distillation. Knowledge distillation for neural networks is first introduced in Buciluă et al. (2006); Hinton et al. (2015). By encouraging the student model to approximate the output logits of the teacher model, the student is able to imitate the teacher’s behavior with marginal quality loss (Romero et al., 2015; Komodakis and Zagoruyko, 2017; Kim et al., 2018; Tung and Mori, 2019; Koratana et al., 2019; Huang and Wang, 2017; Ahn et al., 2019; Tian et al., 2020). Some work study the ensemble distillation, i.e. distilling the knowledge of an ensemble of teacher models to a student model. To this end, existing approaches either average the logits from the ensemble of teacher models (You et al., 2017a; Furlanello et al., 2018; Anil et al., 2018; Dvornik et al., 2019), or extract knowledge from the feature level (Park and Kwak, 2019; Liu et al., 2019a; Wu et al., 2019).

Most of these schemes rely on using the original training data for the distillation process. In cases where real data is unavailable, some recent work (Nayak et al., 2019; Micaelli and Storkey, 2019) demonstrate that distillation can be accomplished by crafting pseudo data either from the weights of the teacher model or through a generator adversarially trained with the student. FedDF can be combined with all of these approaches. In this work, we consider unlabeled datasets for ensemble distillation, which could be either collected from other domains or directly generated from a pre-trained generator.

Comparison with close FL work. Guha *et al.* (Guha et al., 2019) propose “one-shot fusion” through unlabeled data for SVM loss objective, whereas we consider multiple-round scenarios on diverse neural architectures and tasks. FD (Jeong et al., 2018) utilizes distillation to reduce FL communication costs. To this end, FD synchronizes logits per label which are accumulated during the local training. The averaged logits per label (over local steps and clients) will then be used as a distillation regularizer for the next round’s local training. Compared to FEDAVG, FD experiences roughly 15% quality drop on MNIST. In contrast, FedDF shows superior learning performance over FEDAVG and can significantly reduce the number of communication rounds to reach target accuracy on diverse challenging tasks.

FedMD (Li and Wang, 2019) and the recently proposed Cronus (Chang et al., 2019) consider learning through averaged logits per sample on a public dataset. After the initial pre-training on the labeled public dataset, FedMD learns on the public and private dataset iteratively for personalization, whereas in Cronus, the public dataset (with soft labels) is used jointly with local private data for the local training. As FedMD trains client models simultaneously on both labeled public and private datasets, the model classifiers have to include all classes from both datasets. Cronus, in its collaborative training phase, mixes public and private data for local training. Thus for these methods, the public dataset construction requires careful deliberation and even prior knowledge on clients’ private data. Moreover, how these modifications impact local training quality remains unclear. FedDF faces no such issues: we show that FedDF is robust to distillation dataset selection and the distillation is performed on the server side, leaving local training unaffected. We include a detailed discussion with FedMD, Cronus in Appendix 17.1.

At the time of preparing the camera ready version of this chapter for NeurIPS 2020 conference (Lin et al., 2020b), we also noticed other contemporary work (Sun and Lyu, 2021; Chen and Chao, 2021; Zhou et al., 2020; He et al., 2020a) and we defer discussions to Appendix 17.1.

8.4 Ensemble Distillation for Robust Model Fusion

In this section, we first introduce the core idea of the proposed Federated Distillation Fusion (FedDF). We then comment on its favorable characteristics and discuss possible extensions.

Ensemble distillation. We first discuss the key features of FedDF for the special case of homogeneous models, i.e. when all clients share the same network architecture (Algorithm 5).

8.4. Ensemble Distillation for Robust Model Fusion

Algorithm 5 Illustration of FedDF on K homogeneous clients (indexed by k) for T rounds, n_k denotes the number of data points per client and C the fraction of clients participating in each round. The server model is initialized as \mathbf{x}_0 . While FEDAVG just uses the averaged models $\mathbf{x}_{t,0}$, we perform N iterations of server-side model fusion on top (line 7 – line 10).

```

1: procedure SERVER
2:   for each communication round  $t = 1, \dots, T$  do
3:      $S_t \leftarrow$  random subset ( $C$  fraction) of the  $K$  clients
4:     for each client  $k \in \mathcal{S}_t$  in parallel do
5:        $\hat{\mathbf{x}}_t^k \leftarrow$  Client-LocalUpdate( $k, \mathbf{x}_{t-1}$ ) ▷ detailed in Algorithm 17.
6:       initialize for model fusion  $\mathbf{x}_{t,0} \leftarrow \sum_{k \in \mathcal{S}_t} \frac{n_k}{\sum_{k \in \mathcal{S}_t} n_k} \hat{\mathbf{x}}_t^k$ 
7:       for  $j$  in  $\{1, \dots, N\}$  do
8:         sample a mini-batch of samples  $\mathbf{d}$ , from e.g. (1) an unlabeled dataset, (2) a
generator
9:         use ensemble of  $\{\hat{\mathbf{x}}_t^k\}_{k \in \mathcal{S}_t}$  to update server student  $\mathbf{x}_{t,j-1}$  through AVGLLOGITS
10:         $\mathbf{x}_t \leftarrow \mathbf{x}_{t,N}$ 
11:   return  $\mathbf{x}_T$ 

```

For model fusion, the server distills the ensemble of $|\mathcal{S}_t|$ client teacher models to one single server student model. For the distillation, the teacher models are evaluated on mini-batches of unlabeled data on the server (forward pass) and their logit outputs (denoted by $f(\hat{\mathbf{x}}_t^k, \mathbf{d})$ for mini-batch \mathbf{d}) are used to train the student model on the server:

$$\mathbf{x}_{t,j} := \mathbf{x}_{t,j-1} - \eta \frac{\partial \text{KL} \left(\sigma \left(\frac{1}{|\mathcal{S}_t|} \sum_{k \in \mathcal{S}_t} f(\hat{\mathbf{x}}_t^k, \mathbf{d}) \right), \sigma(f(\mathbf{x}_{t,j-1}, \mathbf{d})) \right)}{\partial \mathbf{x}_{t,j-1}}. \quad (\text{AVGLLOGITS})$$

Here KL stands for Kullback–Leibler divergence, σ is the softmax function, and η is the stepsize.

FedDF can easily be extended to heterogeneous FL systems (Algorithm 18 and Figure 17.1 in Appendix 17.2). We assume the system contains p distinct model prototype groups that potentially vary in neural architecture, structure and numerical precision. By ensemble distillation, each model architecture group acquires knowledge from logits averaged over *all* received models, thus mutual beneficial information can be shared across architectures; in the next round, each activated client receives the corresponding fused prototype model. Notably, as the fusion takes place on the server side, there is no additional burden and interference on clients.

Utilizing unlabeled/generated data for distillation. Unlike most existing ensemble distillation methods that rely on *labeled* data from the training domain, we demonstrate the feasibility of achieving model fusion by using *unlabeled* datasets from other domains for the sake of privacy-preserving FL. Our proposed method also allows the use of synthetic data from a pre-trained generator (e.g. GAN¹) as distillation data to alleviate potential limitations (e.g. acquisition,

¹ GAN training is not involved in all stages of FL and cannot steal clients’ data. Data generation is done by the (frozen) generator before the FL training by performing inference on random noise. Adversarially involving GAN’s training during the FL training may cause the privacy issue, but it is beyond the scope of this paper.

storage) of real unlabeled datasets.

Discussions on privacy-preserving extension. Our proposed model fusion framework in its simplest form—like most existing FL methods—requires to exchange models between the server and each client, resulting in potential privacy leakage due to e.g. memorization present in the models. Several existing protection mechanisms can be added to our framework to protect clients from adversaries. These include adding differential privacy (Geyer et al., 2017) for client models, or performing hierarchical and decentralized model fusion through synchronizing locally inferred logits e.g. on *random* public data², as in the recent work (Chang et al., 2019). We leave further explorations of this aspect for future work.

8.5 Experiments

8.5.1 Setup

Datasets and models. We evaluate the learning of various SOTA FL methods on both CV and NLP tasks, on architectures of ResNet (He et al., 2016a), VGG (Simonyan and Zisserman, 2015), ShuffleNetV2 (Ma et al., 2018a) and DistilBERT (Sanh et al., 2019). We consider federated learning CIFAR-10/100 (Krizhevsky and Hinton, 2009) and ImageNet (Krizhevsky et al., 2012) (down-sampled to image resolution 32 for computational feasibility (Chrabaszcz et al., 2017)) from scratch for CV tasks; while for NLP tasks, we perform federated fine-tuning on a 4-class news classification dataset (AG News (Zhang et al., 2015)) and a 2-class classification task (Stanford Sentiment Treebank, SST2 (Socher et al., 2013)). The validation dataset is created for CIFAR-10/100, ImageNet, and SST2, by holding out 10%, 1% and 1% of the original training samples respectively; the remaining training samples are used as the training dataset (before partitioning client data) and the whole procedure is controlled by random seeds. We use validation/test datasets on the server and report the test accuracy over three various random seeds.

Heterogeneous distribution of client data. We use the Dirichlet distribution as in Yurochkin et al. (2019); Hsu et al. (2019) to create disjoint non-i.i.d. client training data. The value of α controls the degree of non-i.i.d.-ness: $\alpha = 100$ mimics identical local data distributions, and the smaller α is, the more likely the clients hold examples from only one class (randomly chosen). Figure 8.2 visualizes how samples are distributed among 20 clients for CIFAR-10 on various α values; more visualizations are shown in Appendix 17.3.2.

Baselines. FedDF is designed for effective model fusion on the server, considering the accuracy of the global model on the test dataset. Thus we omit the comparisons to methods designed for personalization (e.g. FedMD (Li and Wang, 2019)), security/robustness (e.g. Cronus (Chang et al.,

² For instance, these data can be generated locally from identical generators with a controlled random state.

2019)), and communication efficiency (e.g. (Jeong et al., 2018), known for poorer performance than FEDAVG). We compare FedDF with SOTA FL methods, including 1) FEDAVG (McMahan et al., 2017), 2) FEDPROX (Li et al., 2020b) (for better local training under heterogeneous systems), 3) accelerated FEDAVG a.k.a. FEDAVGM³ (Hsu et al., 2019, 2020), and 4) FEDMA⁴ (Wang et al., 2020a) (for better model fusion). We elaborate on the reasons for omitted numerical comparisons in Appendix 17.1.

The local training procedure. The FL algorithm randomly samples a fraction (C) of clients per communication round for local training. For the sake of simplicity, the local training in our experiments uses a constant learning rate (no decay), no Nesterov momentum acceleration, and no weight decay. The hyperparameter tuning procedure is deferred to Appendix 17.3.2. Unless mentioned otherwise the learning rate is set to 0.1 for ResNet-like nets, 0.05 for VGG, and $1e-5$ for DistilBERT.

The model fusion procedure. We evaluate the performance of FedDF by utilizing either randomly sampled data from existing (unlabeled) datasets⁵ or BigGAN’s generator (Brock et al., 2019). Unless mentioned otherwise we use CIFAR-100 and downsampled ImageNet (image size 32) as the distillation datasets for FedDF on CIFAR-10 and CIFAR-100 respectively. Adam with learning rate $1e-3$ (w/ cosine annealing) is used to distill knowledge from the ensemble of received local models. We employ early-stopping to stop distillation after the validation performance plateaus for $1e3$ steps (total $1e4$ update steps). The hyperparameter used for model fusion is kept constant over all tasks.

8.5.2 Evaluation on the Common Federated Learning Settings

Performance overview for various FL scenarios. We can observe from Figure 8.2 that FedDF consistently outperforms FEDAVG for all client fractions and non-i.i.d. degrees when the local training is reasonably sufficient (e.g. over 40 epochs).

FedDF benefits from larger numbers of local training epochs. This is because the performance of the model ensemble is highly dependent on the diversity among its individual models (Kuncheva and Whitaker, 2003; Sollich and Krogh, 1996). Thus longer local training leads to greater diversity and quality of the ensemble and hence a better distillation result for the fused model. This characteristic is desirable in practice as it helps reduce the communication overhead in FL systems. In contrast, the performance of FEDAVG saturates and even degrades with the increased number of local epochs, which is consistent with observations in McMahan et al. (2017); Caldas

³ The performance of FEDAVGM is coupled with local learning rate, local training epochs, and the number of communication rounds. The preprints (Hsu et al., 2019, 2020) consider small learning rate for at least 10k communication rounds; while we use much fewer communication rounds, which sometimes result in various observations.

⁴ FEDMA does not support BN or residual connections, thus the comparison is only performed on VGG-9.

⁵ Note the actual computation expense for distillation is determined by the product of the number of distillation steps and distillation mini-batch size (128 in all experiments), rather than the distillation dataset size.

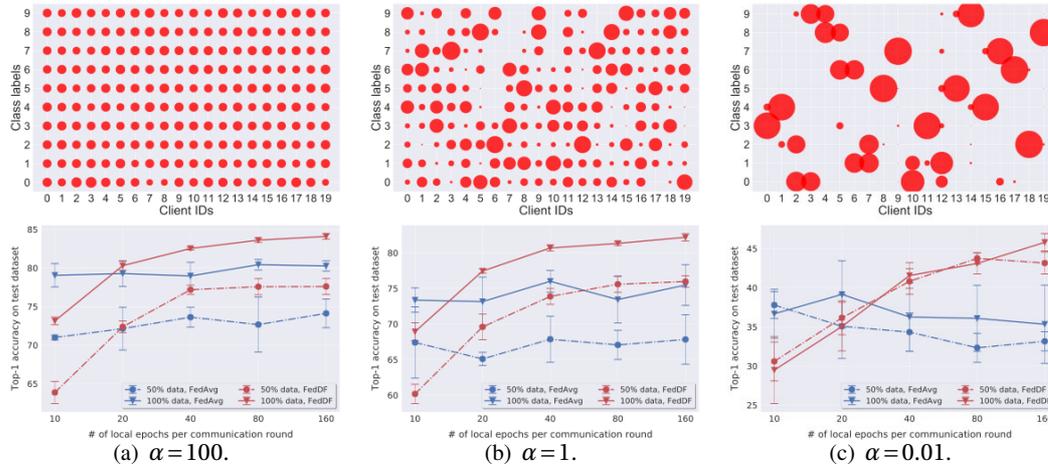


Figure 8.2 – **Top: Illustration of # of samples per class allocated to each client** (indicated by dot sizes), for various Dirichlet distribution α values. **Bottom: Test performance of FedDF and FEDAVG on CIFAR-10 with ResNet-8**, for various local training settings: non-i.i.d. degrees α , data fractions, and # of local epochs per communication round. We perform 100 communication rounds, and active clients are sampled with ratio $C=0.4$ from a total of 20 clients. Detailed learning curves in these scenarios can be found in Appendix 17.3.4.

et al. (2018); Wang et al. (2020a). As FedDF focuses on better model fusion on the server side, it is orthogonal to recent techniques (e.g. (Shoham et al., 2019; Karimireddy et al., 2020; Deng et al., 2020)) targeting the issue of non-i.i.d. local data. We believe combining FedDF with these techniques can lead to a more robust FL, which we leave as future work.⁶

Ablation study of FedDF. We provide detailed ablation study for FedDF in Appendix 17.3.4 to identify the source of the benefits. For example, Table 17.1 justifies the importance of using the uniformly averaged local models as a starting model (line 6 in Algorithm 5 and line 11 in Algorithm 18), for the quality of ensemble distillation in FedDF. We further investigate the effect of various optimizers (for on-server ensemble distillation) on the federated learning performance in Table 17.2 and Table 17.3.

Detailed comparison of FedDF with other SOTA federated learning methods for CV tasks. Table 8.1 summarizes the results for various degrees of non-i.i.d. data, local training epochs and client sampling fractions. In all scenarios, FedDF requires significantly fewer communication rounds than other SOTA methods to reach designated target accuracies. The benefits of FedDF can be further pronounced by taking more local training epochs as illustrated in Figure 8.2.

All competing methods have strong difficulties with increasing data heterogeneity (non-i.i.d. data, i.e. smaller α), while FedDF shows significantly improved robustness to data heterogeneity. In most scenarios in Table 8.1, the reduction of α from 1 to 0.1 almost triples the number

⁶ We include some preliminary results to illustrate the compatibility of FedDF in Table 17.4 (Appendix 17.3.4).

8.5. Experiments

Table 8.1 – **Evaluating various FL methods in diverse scenarios** (i.e. various client sampling fractions, # of local epochs and target accuracies), in terms of **the number of communication rounds to reach target top-1 test accuracy**. We evaluate on ResNet-8 with CIFAR-10. For each communication round, a fraction C of the total 20 clients are randomly selected. T denotes the specified target top-1 test accuracy. Hyperparameters are fine-tuned for each method (FEDAVG, FEDPROX, and FEDAVGM); FedDF uses the optimal learning rate from FEDAVG. The performance upper bound of (tuned) centralized training is 86% (trained on all local data).

Local epochs		The number of communication rounds to reach target performance T					
		$C=0.2$		$C=0.4$		$C=0.8$	
		$\alpha=1, T=80\%$	$\alpha=0.1, T=75\%$	$\alpha=1, T=80\%$	$\alpha=0.1, T=75\%$	$\alpha=1, T=80\%$	$\alpha=0.1, T=75\%$
FEDAVG	1	350 ± 31	546 ± 191	246 ± 41	445 ± 8	278 ± 83	361 ± 111
	20	144 ± 51	423 ± 105	97 ± 29	309 ± 88	103 ± 26	379 ± 151
	40	130 ± 13	312 ± 87	104 ± 52	325 ± 82	100 ± 76	312 ± 110
FEDPROX	20	99 ± 61	346 ± 12	91 ± 40	235 ± 41	92 ± 21	237 ± 93
	40	115 ± 17	270 ± 96	87 ± 49	229 ± 79	80 ± 44	284 ± 130
FEDAVGM	20	92 ± 15	299 ± 85	92 ± 46	221 ± 29	97 ± 37	235 ± 129
	40	135 ± 52	322 ± 99	78 ± 28	224 ± 38	83 ± 34	232 ± 11
FedDF (ours)	20	61 ± 24	102 ± 42	28 ± 10	51 ± 4	22 ± 1	33 ± 18
	40	28 ± 6	80 ± 25	20 ± 4	39 ± 10	14 ± 2	20 ± 4

of communication rounds for FEDAVG, FEDPROX and FEDAVGM to reach target accuracies, whereas less than twice the number of rounds are sufficient for FedDF.

Increasing the sampling ratio makes a more noticeable positive impact on FedDF compared to other methods. We attribute this to the fact that an ensemble tends to improve in robustness and quality, with a larger number of reasonable good participants, and hence results in better model fusion. Nevertheless, even in cases with a very low sampling fraction (i.e. $C=0.2$), FedDF still maintains a considerable leading margin over the closest competitor.

Comments on Batch Normalization. Batch Normalization (BN) (Ioffe and Szegedy, 2015) is the current workhorse in convolutional deep learning tasks and has been employed by default in most SOTA CNNs (He et al., 2016a; Huang et al., 2017b; Ma et al., 2018a; Tan and Le, 2019). However, it often fails on heterogeneous training data. Hsieh *et al.* (Hsieh et al., 2020) recently examined the non-i.i.d. data ‘quagmire’ for distributed learning and point out that replacing BN by Group Normalization (GN) (Wu and He, 2018) can alleviate some of the quality loss brought by BN due to the discrepancies between local data distributions.

As shown in Table 8.2, despite additional effort on architecture modification and hyperparameter tuning (i.e. the number of groups in GN), baseline methods with GN replacement still lag much behind FedDF. FedDF provides better model fusion which is robust to non-i.i.d. data, and is compatible with BN, thus avoids extra efforts for modifying the standard SOTA neural architectures. Figure 17.7 in Appendix 17.3.3 shows the complete learning curves.

We additionally evaluate architectures originally designed without BN (i.e. VGG), to demonstrate the broad applicability of FedDF. Due to the lack of normalization layers, VGG is vulnerable

Chapter 8. Ensemble Distillation for Robust Model Fusion in Federated Learning

Table 8.2 – **The impact of normalization techniques** (i.e. BN, GN) for ResNet-8 on CIFAR (20 clients with $C=0.4$, 100 communication rounds, and 40 local epochs per round). We use a constant learning rate and tune other hyperparameters. The distillation dataset of FedDF for CIFAR-100 is ImageNet (with image size of 32).

Top-1 test accuracy of various methods						
Datasets		FEDAVG, w/ BN	FEDAVG, w/ GN	FEDPROX, w/ GN	FEDAVGM, w/ GN	FedDF, w/ BN
CIFAR-10	$\alpha=1$	76.01 ± 1.53	78.57 ± 0.22	76.32 ± 1.98	77.79 ± 1.22	80.69 ± 0.43
	$\alpha=0.1$	62.22 ± 3.88	68.37 ± 0.50	68.65 ± 0.77	68.63 ± 0.79	71.36 ± 1.07
CIFAR-100	$\alpha=1$	35.56 ± 1.99	42.54 ± 0.51	42.94 ± 1.23	42.83 ± 0.36	47.43 ± 0.45
	$\alpha=0.1$	29.14 ± 1.91	36.72 ± 1.50	35.74 ± 1.00	36.29 ± 1.98	39.33 ± 0.03

Table 8.3 – **Top-1 test accuracy of federated learning CIFAR-10 on VGG-9 (w/o BN)**, for 20 clients with $C=0.4$, $\alpha=1$ and 100 communication rounds (40 epochs per round). We by default drop dummy predictors.

Methods	Top-1 test accuracy @ communication round				
	5	10	20	50	100
FEDAVG (w/o drop-worst)	45.72 ± 30.95	51.06 ± 35.56	53.22 ± 37.43	29.60 ± 40.66	7.52 ± 4.29
FEDMA (w/o drop-worst) ¹	23.41 ± 0.00	27.55 ± 0.10	41.56 ± 0.08	60.35 ± 0.03	65.0 ± 0.02
FEDAVG	64.77 ± 1.24	70.28 ± 1.02	75.80 ± 1.36	77.98 ± 1.81	78.34 ± 1.42
FEDPROX	63.86 ± 1.55	71.85 ± 0.75	75.57 ± 1.16	77.85 ± 1.96	78.60 ± 1.91
FedDF	66.08 ± 4.14	72.80 ± 1.59	75.82 ± 2.09	79.05 ± 0.54	80.36 ± 0.63

¹ FEDMA does not support drop-worst operation due to its layer-wise communication/fusion scheme. The number of local training epochs per layer is 5 (45 epochs per model) thus results in stabilized training. More details can be found in Appendix 17.3.2.

to non-i.i.d. local distributions. We observe that received models on the server might output random prediction results on the validation/test dataset and hence give rise to uninformative results overwhelmed by large variance (as shown in Table 8.3). We address this issue by a simple treatment⁷, “drop-worst”, i.e. dropping learners with random predictions on the server validation dataset (e.g. 10% accuracy for CIFAR-10), in each round before applying model averaging and/or ensemble distillation. Table 8.3 examines the FL methods (FEDAVG, FEDPROX, FEDMA and FedDF) on VGG-9; FedDF consistently outperforms other methods by a large margin for various communication rounds.

Extension to NLP tasks for federated fine-tuning of DistilBERT. Fine-tuning a pre-trained transformer language model like BERT (Devlin et al., 2019) yields SOTA results on various NLP benchmarks (Wang et al., 2018, 2019a). DistilBERT (Sanh et al., 2019) is a lighter version of BERT with only marginal quality loss on downstream tasks. As a proof of concept, in Figure 8.3 we consider federated fine-tuning of DistilBERT on non-i.i.d. local data ($\alpha=1$, depicted in Figure 17.5). For both AG News and SST2 datasets, FedDF achieves significantly faster convergence than FEDAVG and consistently outperforms the latter.

⁷ Techniques (e.g. Krum, Bulyan), can be adapted to further improve the robustness or defend against attacks.

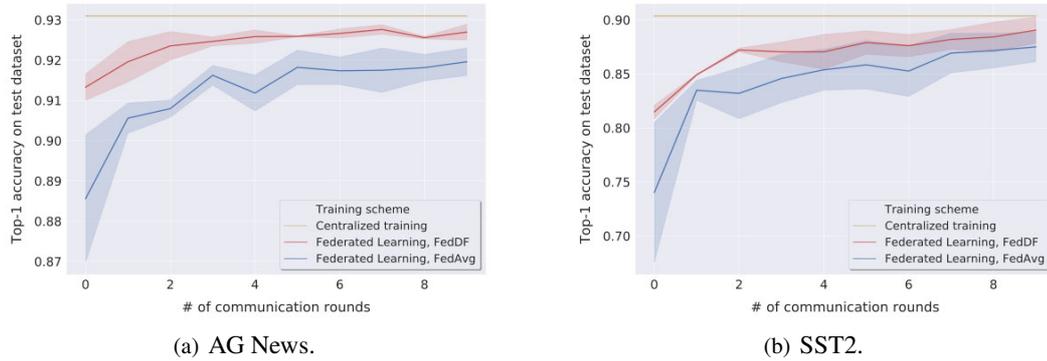


Figure 8.3 – **Federated fine-tuning DistilBERT** on (a) AG News and (b) SST-2. For simplicity, we consider 10 clients with $C=100\%$ participation ratio and $\alpha=1$; the number of local training epochs per communication round (10 rounds in total) is set to 10 and 1 respectively. The 50% of the original training dataset is used for the federated fine-tuning (for all methods) and the left 50% is used as the unlabeled distillation dataset for FedDF.

Table 8.4 – **Federated learning with low-precision models (1-bit binarized ResNet-8) on CIFAR-10**. For each communication round (100 in total), 40% of the total 20 clients ($\alpha=1$) are randomly selected.

Local Epochs	ResNet-8-BN (FEDAVG)	ResNet-8-GN (FEDAVG)	ResNet-8-BN (FedDF)
20	44.38 \pm 1.21	59.70 \pm 1.65	59.49 \pm 0.98
40	43.91 \pm 3.26	64.25 \pm 1.31	65.49 \pm 0.74
80	47.62 \pm 1.84	65.99 \pm 1.29	70.27 \pm 1.22

8.5.3 Case Studies

Federated learning for low-bit quantized models. FL for the Internet of Things (IoT) involves edge devices with diverse hardwares, e.g. from the perspective of computational capacities. Network quantization is hence of great interest to FL by representing the activations/weights in low precision, with benefits of significantly reduced local computational footprints and communication costs. Table 8.4 examines the model fusion performance for binarized ResNet-8 (Rastegari et al., 2016; Hubara et al., 2017). FedDF can be on par with or outperform FEDAVG by a noticeable margin, without introducing extra GN tuning overheads.

Federated learning on heterogeneous systems. Apart from non-i.i.d. local distributions, another major source of heterogeneity in FL systems manifests in neural architectures (Li and Wang, 2019). Figure 8.4 visualizes the training dynamics of FedDF and FEDAVG⁸ in a heterogeneous system with three distinct architectures, i.e. ResNet-20, ResNet-32, and ShuffleNetV2. On CIFAR-10/100 and ImageNet, FedDF dominates FEDAVG on test accuracy in each communication round with much less variance. Each fused model exhibits marginal quality loss compared to the ensemble performance, which suggests unlabeled datasets from other domains are sufficient for model fusion. Besides, the gap between the fused model and the ensemble one widens when

⁸ Model averaging is only performed among models with identical structures.

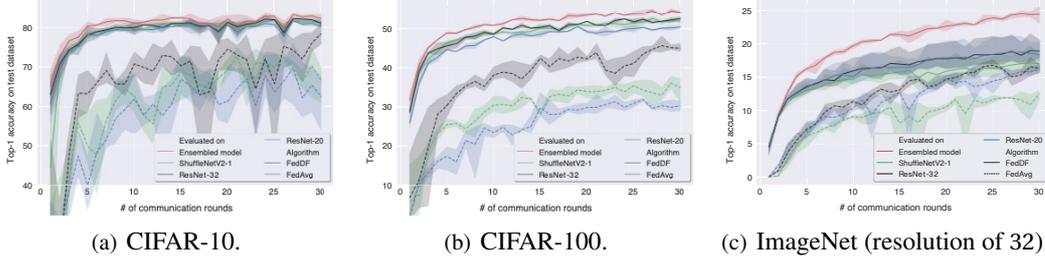


Figure 8.4 – **Federated learning on heterogeneous systems (model/data)**, with three neural architectures (ResNet-20, ResNet-32, ShuffleNetV2) and non-i.i.d. local data distribution ($\alpha = 1$). We consider 21 clients for CIFAR (client sampling ratio $C=0.4$) and 150 clients for ImageNet ($C=0.1$); various neural architectures are evenly distributed among clients. We train 80 local training epochs per communication round (total 30 rounds). CIFAR-100, STL-10, and STL-10 are used as the distillation datasets for CIFAR-10/100 and ImageNet training respectively. The *solid* lines show the results of FedDF for a given communication round, while *dashed* lines correspond to that of FEDAVG; *colors* indicate model architectures.

the training dataset contains a much larger number of classes⁹ than that of the distillation dataset. For instance, the performance gap is negligible on CIFAR-10, whereas on ImageNet, the gap increases to around 6%. In Section 8.6, we study this underlying interaction between training data and unlabeled distillation data in detail.

8.6 Understanding FedDF

FedDF consists of two chief components: ensembling and knowledge distillation via out-of-domain data. In this section, we first investigate what affects the ensemble performance on the global distribution (test domain) through a generalization bound. We then provide empirical understanding of how various attributes of the out-of-domain distillation dataset affect the student performance on the global distribution.

Generalization bound. Theorem 8.6.1 provides insights into ensemble performance on the global distribution. Detailed description and derivations are deferred to Appendix 17.4.

Theorem 8.6.1 (informal). *We denote the global distribution as \mathcal{D} , the k -th local distribution and its empirical distribution as \mathcal{D}_k and $\hat{\mathcal{D}}_k$ respectively. The hypothesis $h \in \mathcal{H}$ learned on $\hat{\mathcal{D}}_k$ is denoted by $h_{\hat{\mathcal{D}}_k}$. The upper bound on the risk of the ensemble of K local models on \mathcal{D} mainly consists of 1) the empirical risk of a model trained on the global empirical distribution $\hat{\mathcal{D}} = \frac{1}{K} \sum_k \hat{\mathcal{D}}_k$, and 2) terms dependent on the distribution discrepancy between \mathcal{D}_k and \mathcal{D} , with the probability $1 - \delta$:*

$$L_{\mathcal{D}}\left(\frac{1}{K} \sum_k h_{\hat{\mathcal{D}}_k}\right) \leq L_{\hat{\mathcal{D}}}\left(h_{\hat{\mathcal{D}}}\right) + \frac{1}{K} \sum_k \left(\frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_k, \mathcal{D}) + \lambda_k \right) + \frac{4 + \sqrt{\log(\tau_{\mathcal{H}}(2m))}}{\delta / K \sqrt{2m}},$$

where $d_{\mathcal{H}\Delta\mathcal{H}}$ measures the distribution discrepancy between two distributions (Ben-David et al.,

⁹ # of classes is a proxy measurement for distribution shift; labels are not used in our distillation procedure.

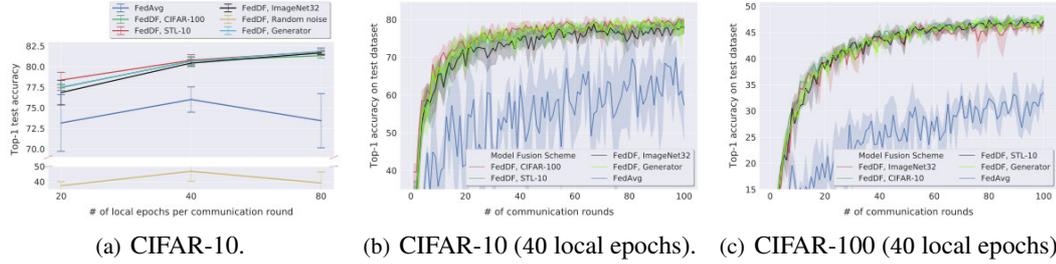


Figure 8.5 – **The performance of FedDF on various distillation datasets:** random uniformly sampled noises, randomly generated images (from the generator), CIFAR, downsampled ImageNet32, and downsampled STL-10. We evaluate ResNet-8 on CIFAR for 20 clients, with $C = 0.4$, $\alpha = 1$ and 100 communication rounds.

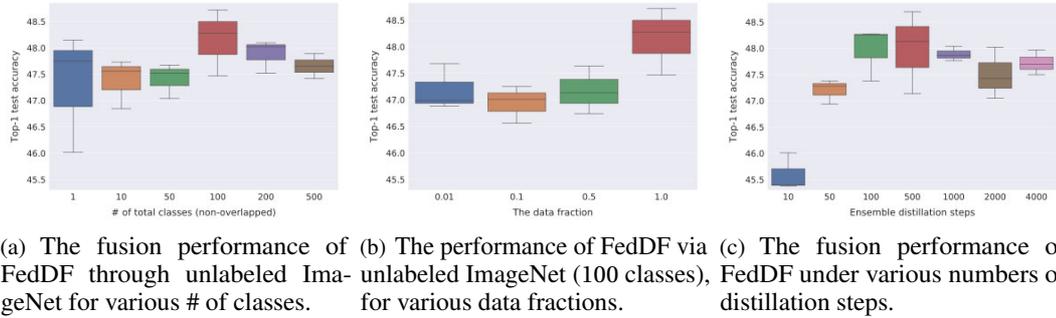


Figure 8.6 – **Understanding knowledge distillation behaviors of FedDF on # of classes (8.6(a)), sizes of the distillation dataset (8.6(b)), and # of distillation steps (8.6(c)),** for federated learning ResNet-8 on CIFAR-100, with $C=0.4$, $\alpha = 1$ and 100 communication rounds (40 local epochs per round). ImageNet with image resolution 32 is considered as our base unlabeled dataset. For simplicity, only classes without overlap with CIFAR-100 classes are considered, in terms of the synonyms, hyponyms, or hypernyms of the class name.

2010), m is the number of samples per local distribution, λ_k is the minimum of the combined loss $\mathcal{L}_{\mathcal{D}}(h) + \mathcal{L}_{\mathcal{D}_k}(h), \forall h \in \mathcal{H}$, and $\tau_{\mathcal{H}}$ is the growth function bounded by a polynomial of the VCdim of \mathcal{H} .

The ensemble of the local models sets the performance upper bound for the later distilled model on the global distribution as shown in Figure 8.4. Theorem 8.6.1 shows that compared to a model trained on the global empirical distribution (ideal centralized case), the performance of the ensemble on the global distribution is associated with the discrepancy between local distributions \mathcal{D}_k 's and the global distribution \mathcal{D} . Besides, the shift between the distillation and the global distribution determines the knowledge transfer quality between these two distributions and hence the test performance of the fused model. In the following, we empirically examine how the choice of distillation data distributions and the number of distillation steps influence the quality of ensemble knowledge distillation.

Source, diversity and size of the distillation dataset. The fusion in FedDF demonstrates remarkable consistency across a wide range of realistic data sources as shown in Figure 8.5, although an abrupt performance declination is encountered when the distillation data are sampled from a dramatically different manifold (e.g. random noise). Notably, synthetic data from the generator of a pre-trained GAN does not incur noticeable quality loss, opening up numerous possibilities for effective and efficient model fusion. Figure 8.6(a) illustrates that in general the diversity of the distillation data does not significantly impact the performance of ensemble distillation, though the optimal performance is achieved when two domains have a similar number of classes. Figure 8.6(b) shows the FedDF is not demanding on the distillation dataset size: even 1% of data (~ 48% of the local training dataset) can result in a reasonably good fusion performance.

Distillation steps. Figure 8.6(c) depicts the impact of distillation steps on fusion performance, where FedDF with a moderate number of the distillation steps is able to approach the optimal performance. For example, 100 distillation steps in Figure 8.6(c), which corresponds to 5 local epochs of CIFAR-100 (partitioned by 20 clients), suffice to yield satisfactory performance. Thus FedDF introduces minor time-wise expense.

Broader Impact

We believe that collaborative learning schemes such as federated learning are an important element towards enabling privacy-preserving training of ML models, as well as a better alignment of each individual’s data ownership with the resulting utility from jointly trained machine learning models, especially in applications where data is user-provided and privacy sensitive ([Kairouz et al., 2021](#); [Nedic, 2020](#)).

In addition to privacy, efficiency gains and lower resource requirements in distributed training reduce the environmental impact of training large machine learning models. The introduction of a practical and reliable distillation technique for heterogeneous models and for low-resource clients is a step towards more broadly enabling collaborative privacy-preserving and efficient decentralized learning.

9 Accelerating Decentralized Deep Learning on Heterogeneous Data

9.1 Preface

Contribution and sources. This chapter reproduces [Lin et al. \(2021\)](#). The initial idea, development and implementation of the algorithm, analysis, experiments, and most of the writing, were carried out by the author. The convergence analysis of the proposed algorithm was mostly provided by Sai Praneeth Karimireddy.

Summary. Decentralized training of deep learning models is a key element for enabling data privacy and on-device learning over networks. In realistic learning scenarios, the presence of heterogeneity across diverse clients’ local datasets poses an optimization challenge and may severely deteriorate the generalization performance.

In this chapter, we investigate and identify the limitation of several decentralized optimization algorithms for various degrees of data heterogeneity. We propose a novel momentum-based method—quasi-global momentum—to mitigate this decentralized training difficulty. We show in extensive empirical experiments on various CV/NLP datasets (CIFAR-10, ImageNet, and AG News) and several network topologies (Ring and Social Network) that our method is much more robust to the heterogeneity of clients’ data than other existing methods, by a significant improvement in test performance (1%–20%). Our code is publicly available.¹

Our method can be seen as a first practical deep learning algorithm in the decentralized setting, mitigates the optimization difficulties, and the resultant generalization performance outperforms all other available methods while remaining both communication and computation efficient. Together with other colleagues, we further developed relaySGD ([Vogels et al., 2021](#)), a follow-up work that investigated the limitation of gossip averaging—the backbone of existing decentralized learning algorithms—and addressed the data heterogeneity from an entirely new perspective.

¹Code: github.com/epfml/quasi-global-momentum

9.2 Introduction

Decentralized machine learning methods—allowing communications in a peer-to-peer fashion on an underlying communication network topology (without a central coordinator)—have emerged as an important paradigm in large-scale machine learning (Lian et al., 2017, 2018; Koloskova et al., 2019, 2020b). Decentralized Stochastic Gradient Descent (DSGD) methods offer (1) scalability to large datasets and systems in large data-centers (Lian et al., 2017; Assran et al., 2019; Koloskova et al., 2020a), as well as (2) privacy-preserving learning for the emerging EdgeAI applications (Kairouz et al., 2021; Koloskova et al., 2020a), where the training data remains distributed over a large number of clients (e.g. mobile phones, sensors, or hospitals) and is kept locally (never transmitted during training).

A key challenge—in particular in the second scenario—is the large heterogeneity (non-i.i.d.-ness) in the data present on the various clients (Zhao et al., 2018; Kairouz et al., 2021; Hsieh et al., 2020). Heterogeneous data (e.g. as illustrated in Figure 9.1) causes very diverse optimization objectives on each client, which results in slow and unstable global convergence, as well as poor generalization performance (shown in the inline table of Figure 9.1). Addressing these optimization difficulties is essential to realize reliable decentralized deep learning applications. Although such challenges have been theoretically pointed out in Shi et al. (2015); Lee et al. (2015); Tang et al. (2018b); Koloskova et al. (2020b), the empirical performance of various DSGD methods remains poorly understood. To the best of our knowledge, there currently exists no efficient, effective, and robust optimization algorithm yet for decentralized deep learning on heterogeneous data.

In the meantime, SGD with momentum acceleration (SGDm) remains the current workhorse for the state-of-the-art (SOTA) centralized deep learning training (He et al., 2016a; Goyal et al., 2017; He et al., 2019a). For decentralized deep learning, the currently used training recipes (i.e. DSGDm) maintain a local momentum buffer on each worker (Assran et al., 2019; Koloskova et al., 2020a; Nadiradze et al., 2021; Singh et al., 2021; Kong et al., 2021a) while only communicating the model parameters to the neighbors. However, these attempts in prior work mainly consider homogeneous decentralized data—and there is no evidence that local momentum enhances generalization performance of decentralized deep learning on heterogeneous data.

As our first contribution, we investigate how DSGD and DSGDm are impacted by the degree of data heterogeneity and the choice of the network topology. We find that heterogeneous data hinders the local momentum acceleration in DSGDm. We further show that using a high-quality shared momentum buffer (e.g. synchronizing the momentum buffer globally) improves the optimization and generalization performance of DSGDm. However, such a global communication significantly increases the communication cost and violates the decentralized learning setup.

We instead propose Quasi-Global (QG) momentum, a simple, yet effective, method that mitigates the difficulties for decentralized learning on heterogeneous data. Our approach is based on locally approximating the global optimization direction without introducing extra communication overhead. We demonstrate in extensive empirical results that QG momentum can stabilize the

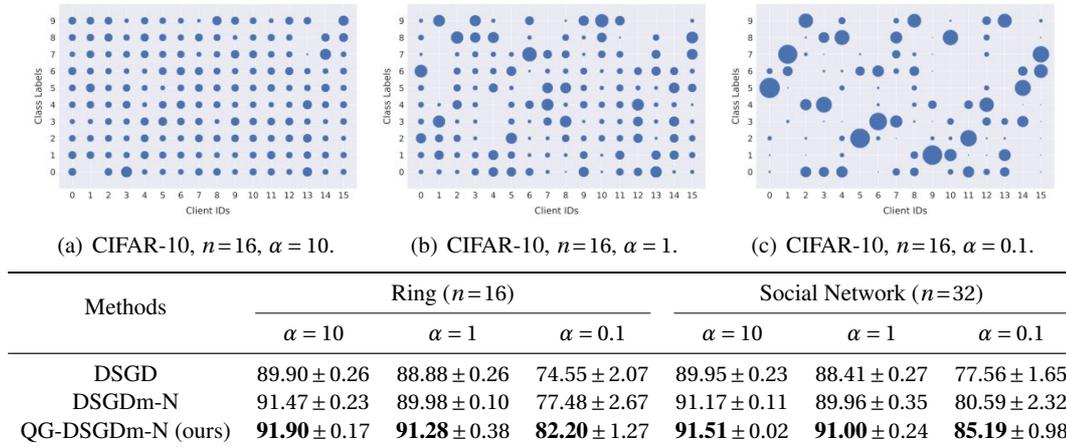


Figure 9.1 – **Illustrating the challenge of heterogeneous data in decentralized deep learning**, for training ResNet-EvoNorm-20 on CIFAR-10. The Dirichlet distribution α values control various non-i.i.d. degrees (Yurochkin et al., 2019; Hsu et al., 2019; He et al., 2020b); the smaller α is, the more likely the clients hold examples from only one class. **The inline figures illustrate the # of samples per class allocated to each client (indicated by dot sizes)**, for the case of $\alpha = 10, 1, 0.1$. The test top-1 accuracy results in the table are averaged over three random seeds, with learning rate tuning for each setting. The performance upper bound (i.e. centralized training without local data re-shuffling) for $n = 16$ and 32 nodes are 92.95 ± 0.13 and 92.88 ± 0.07 respectively. Following prior work, the evaluated DSGD methods maintain local momentum buffer (without synchronization) for each worker; other experimental setup refers to Section 9.6.1.

optimization trajectory, and that it can accelerate decentralized learning achieving much better generalization performance under high data heterogeneity than previous methods.

- We systematically examine the behavior of decentralized optimization algorithms on standard deep learning benchmarks for various degrees of data heterogeneity.
- We propose a novel momentum-based decentralized optimization method—QG-DSGDm and QG-DSGDm-N—to stabilize the local optimization. We validate the effectiveness of our method on a spectrum of non-i.i.d. degrees and network topologies— it is much more robust to the data heterogeneity than all other existing methods.
- We rigorously prove the convergence of our scheme.
- We additionally investigate normalization methods alternative to Batch Normalization (BN) (Ioffe and Szegedy, 2015) in CNNs, due to its particular vulnerable to non-i.i.d. local data and the caused severe quality loss.

9.3 Related Work

Decentralized Deep Learning. The study of decentralized optimization algorithms dates back to Tsitsiklis (1984), relating to use gossip algorithms (Kempe et al., 2003; Xiao and Boyd, 2004; Boyd et al., 2006) to compute aggregates (find consensus) among clients. In the context of machine learning/deep learning, combining SGD with gossip averaging (Lian et al., 2017, 2018; Assran et al., 2019; Koloskova et al., 2020b) has gained a lot of attention recently for the benefits

of *computational scalability*, *communication efficiency*, *data locality*, as well as the favorable leading term in the convergence rate $\mathcal{O}\left(\frac{1}{n\epsilon^2}\right)$ (Lian et al., 2017; Scaman et al., 2017, 2018; Tang et al., 2018b; Koloskova et al., 2019, 2020a,b) which is the same as in centralized mini-batch SGD (Dekel et al., 2012). A weak version of decentralized learning also covers the recent emerging federated learning (FL) setting (Konečný et al., 2016; McMahan et al., 2017; Kairouz et al., 2021; Karimireddy et al., 2020; Lin et al., 2020b) by using (centralized) star-shaped network topology and local updates. Note that specializing our results to the FL setting is beyond the scope of our work. It is also non-trivial to adapt certain very recent techniques developed in FL for heterogeneous data (Karimireddy et al., 2020, 2021; Lin et al., 2020b; Wang et al., 2020b; Das et al., 2020; Haddadpour et al., 2021) to the gossip-based decentralized deep learning.

A line of recent works on decentralized stochastic optimization, like D²/Exact-diffusion (Tang et al., 2018b; Yuan et al., 2020a; Yuan and Alghunaim, 2021), and gradient tracking (Pu and Nedić, 2020; Pan et al., 2020; Lu et al., 2019), proposes various techniques to theoretically eliminate the influence of data heterogeneity between nodes. However, it remains unclear if these theoretically sound methods still endow with superior convergence and generalization properties in deep learning.

Other works focus on improving communication efficiency, from the aspect of communication compression (Tang et al., 2018a; Koloskova et al., 2019, 2020a; Lu and De Sa, 2020; Taheri et al., 2020; Singh et al., 2021; Vogels et al., 2020; Taheri et al., 2020; Nadiradze et al., 2021), less frequent communication through multiple local updates (Hendrikx et al., 2019; Koloskova et al., 2020b; Nadiradze et al., 2021), or better communication topology design (Nedić et al., 2018; Assran et al., 2019; Wang et al., 2019b, 2020c; Neglia et al., 2020; Nadiradze et al., 2021; Kong et al., 2021a).

Mini-batch SGD with Momentum Acceleration. Momentum is a critical component for training the SOTA deep neural networks (Sutskever et al., 2013; Lucas et al., 2019). Despite various empirical successes, the current theoretical understanding of momentum-based SGD methods remains limited (Bottou et al., 2018). A line of work on the serial (centralized) setting has aimed to develop a convergence analysis for various momentum methods as a special case (Yan et al., 2018; Gitman et al., 2019). However, SGD is known to be optimal in the worst case for stochastic non-convex optimization (Arjevani et al., 2019).

In distributed deep learning, most prior works focus on homogeneous data (especially for numerical evaluations) and incorporate momentum with a locally maintained buffer (which has no synchronization) (Lian et al., 2017; Assran et al., 2019; Lin et al., 2020d,a; Koloskova et al., 2020a; Singh et al., 2021). Yu et al. (2019a) propose synchronizing the local momentum buffer periodically for better performance at the cost of doubling the communication. SlowMo (Wang et al., 2020d) instead proposes to periodically perform a slow momentum update on the globally synchronized model parameters (with additional All-Reduce communication cost), for centralized or decentralized methods. Parallel work (Balu et al., 2021) introduces DMSGD for decentralized learning²—it constructs the acceleration momentum from the mixture of the local momentum

² We detail the DMSGD algorithm and clarify the difference in Appendix 18.2.2; we empirically compare with

and consensus momentum. Our proposed method has no extra communication overhead and significantly outperforms all existing methods (in Section 9.6.2).

Batch Normalization in Distributed Learning. Batch Normalization (BN) (Ioffe and Szegedy, 2015) is an indispensable component in deep learning (Santurkar et al., 2018; Luo et al., 2019a) and has been employed by default in most SOTA CNNs (He et al., 2016a; Huang et al., 2017b; Tan and Le, 2019). However, it often fails on distributed deep learning with heterogeneous local data due to the discrepancies between local activation statistics (see recent empirical examination for federated learning in Hsieh et al., 2020; Andreux et al., 2020; Li et al., 2021; Diao et al., 2021). As a remedy, Hsieh et al. (2020) propose to replace BN with Group Normalization (GN) (Wu and He, 2018) to address the issue of local BN statistics, while Andreux et al. (2020); Li et al. (2021); Diao et al. (2021) modify the way of synchronizing the local BN weight/statistics for better generalization performance. In the scope of decentralized learning, the effect of batch normalization has not been investigated yet.

9.4 Method

9.4.1 Notation and Setting

We consider sum-structured distributed optimization problems $f: \mathbb{R}^d \rightarrow \mathbb{R}$ of the form

$$f^* := \min_{\mathbf{x} \in \mathbb{R}^d} [f(\mathbf{x}) := \frac{1}{n} \sum_{i=1}^n f_i(\mathbf{x})], \quad (9.1)$$

where the components $f_i: \mathbb{R}^d \rightarrow \mathbb{R}$ are distributed among the n nodes and are given in stochastic form: $f_i(\mathbf{x}) := \mathbb{E}_{\xi \sim \mathcal{D}_i} [F_i(\mathbf{x}, \xi)]$, where \mathcal{D}_i denotes the local data distribution on node $i \in [n]$. In D(ecentralized)SGD, each node i maintains local parameters $\mathbf{x}_i^{(t)} \in \mathbb{R}^d$, and updates them as:

$$\mathbf{x}_i^{(t+1)} = \sum_{j=1}^n w_{ij} \left(\mathbf{x}_j^{(t)} - \eta \nabla F_j(\mathbf{x}_j^{(t)}, \xi_j^{(t)}) \right), \quad (\text{DSGD})$$

that is, by a stochastic gradient step based on a sample $\xi_i^{(t)} \sim \mathcal{D}_i$, followed by gossip averaging with neighboring nodes in the network topology encoded by the mixing weights w_{ij} .

In this paper, we denote DSGD with local HeavyBall momentum by DSGDm, and DSGD with local Nesterov momentum by DSGDm-N; the naming rule also applies to our method. For the sake of simplicity, we use HeavyBall momentum variants in Section 9.4 and 9.5 for analysis purposes.

9.4.2 QG-DSGDm Algorithm

To motivate the algorithm design, we first illustrate the impact of using different momentum buffers (local v.s. global) on distributed training on heterogeneous data.

DMSGD in Table 9.6.

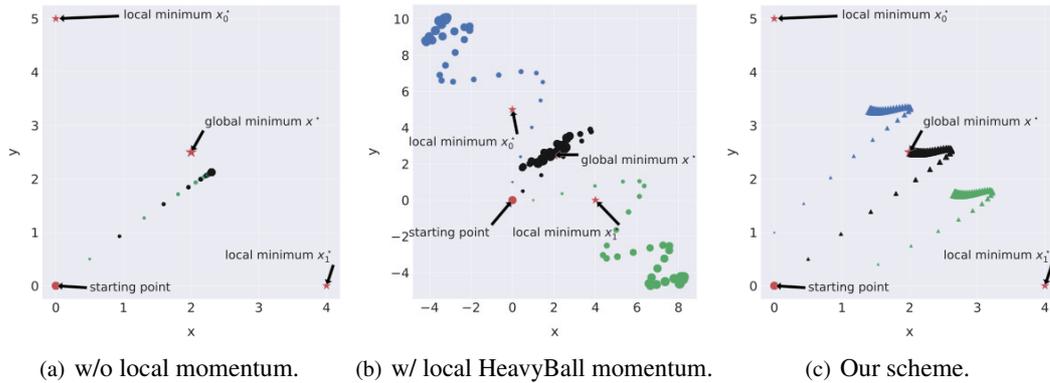


Figure 9.2 – **The ineffectiveness of local momentum acceleration under heterogeneous data setup:** the local momentum buffer accumulates “biased” gradients, causing unstable and oscillation behaviors. The size of marker increase proportional to the number of update steps; colors blue and green indicate the local models of two workers (after performing local update), while black color indicates the synchronized global model. Uniform weight averaging is performed after each update step, and the new gradients are computed on the averaged model. We use the common $\beta = 0.9$ in this illustration. For additional results with various β values refer to Appendix 18.4.2.

Heterogeneous data hinders local momentum acceleration—an example 2D optimization illustration. Figure 9.2 shows a toy 2D optimization example that simulates the biased local gradients caused by heterogeneous data. It depicts the optimization trajectories of two agents ($n = 2$) that start the optimization from the position $(0, 0)$ and receive in every iteration a gradient that points to the local minimum $(0, 5)$ and $(4, 0)$ respectively. The gradient is given by the direction from the current model (position) to the local minimum, and scaled to a constant update magnitude. Model synchronization (i.e. uniform averaging) is performed for every local model update step.

Heterogeneous data strongly influences the effectiveness of the local momentum acceleration. Though local momentum in Figure 9.2(b) assists the models to converge to the neighborhood of the global minimum (better convergence than when excluding local momentum in Figure 9.2(a)), it also causes an unstable and oscillation optimization trajectory. The problem gets even worse in decentralized deep learning, where the learning relies on stochastic gradients from non-convex function and only has limited communication.

Synchronizing the local momentum buffers boosts decentralized learning. We here consider a hypothetical method, which synchronizes the local momentum buffer as in Yu et al. (2019a), to use the global momentum buffer locally (avoid using ill-conditioned local momentum buffer caused by heterogeneous data, as shown by the poor performance in Figure 9.1). We can witness from Table 9.6 that synchronizing the buffer per update step by global averaging to some extent mitigates the issue caused by heterogeneity (1%–5% improvement comparing row 3 with row 7 in Table 9.6). Despite its effectiveness, the global synchronization fundamentally violates the realistic decentralized learning setup and introduces extra communication overhead.

Algorithm 6 Decentralized learning algorithms: QG-DSGDm v.s. DSGDm; Colors indicate the two alternative algorithm variants. At initialization $\mathbf{m}_i^{(0)} = \hat{\mathbf{m}}_i^{(0)} := \mathbf{0}$.

```

1: procedure WORKER- $i$ 
2:   for  $t \in \{1, \dots, T\}$  do
3:     sample  $\xi_i^{(t)}$  and compute  $\mathbf{g}_i^{(t)} = \nabla F_i(\mathbf{x}_i^{(t)}, \xi_i^{(t)})$ 
4:      $\mathbf{m}_i^{(t)} = \beta \mathbf{m}_i^{(t-1)} + \mathbf{g}_i^{(t)}$ 
5:      $\mathbf{m}_i^{(t)} = \beta \hat{\mathbf{m}}_i^{(t-1)} + \mathbf{g}_i^{(t)}$ 
6:      $\mathbf{x}_i^{(t+\frac{1}{2})} = \mathbf{x}_i^{(t)} - \eta \mathbf{m}_i^{(t)}$ 
7:      $\mathbf{x}_i^{(t+1)} = \sum_{j \in \mathcal{N}_i^{(t)}} w_{ij} \mathbf{x}_j^{(t+\frac{1}{2})}$ 
8:      $\mathbf{d}_i^{(t)} = \frac{\mathbf{x}_i^{(t)} - \mathbf{x}_i^{(t+1)}}{\eta}$ 
9:      $\hat{\mathbf{m}}_i^{(t)} = \mu \hat{\mathbf{m}}_i^{(t-1)} + (1 - \mu) \mathbf{d}_i^{(t)}$ 
10:  return  $\mathbf{x}_i^{(T)}$ 

```

Our proposal—QG-DSGDm. Motivated by the performance gain brought by employing a global momentum buffer, we propose a **Quasi-Global (QG)** momentum buffer—a communication-free approach to mimic the global optimization direction—to mitigate the difficulties for decentralized learning on heterogeneous data. Integrating quasi-global momentum with local stochastic gradients alleviates the drift in the local optimization direction, and thus results in a stabilized training and high robustness to heterogeneous data.

Algorithm 6 highlights the difference between DSGDm and QG-DSGDm. Instead of using local gradients from heterogeneous data to form the local momentum (line 4 for DSGDm), which may significantly deflect from the global optimization direction, for QG-DSGDm, we use the difference of two consecutive synchronized models (line 8)

$$\mathbf{d}_i^{(t)} = \frac{1}{\eta} (\mathbf{x}_i^{(t)} - \mathbf{x}_i^{(t+1)}), \quad (9.2)$$

to update the momentum buffer (line 9) by $\hat{\mathbf{m}}_i^{(t)} = \mu \hat{\mathbf{m}}_i^{(t-1)} + (1 - \mu) \mathbf{d}_i^{(t)}$. We set $\mu = \beta$ for all our numerical experiments, without needing hyperparameter tuning.

The update scheme of QG-DSGDm can be re-formulated in matrix form ($\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_n] \in \mathbb{R}^{d \times n}$, etc.) as follows

$$\begin{aligned} \mathbf{X}^{(t+1)} &= \mathbf{W} (\mathbf{X}^{(t)} - \eta (\beta \mathbf{M}^{(t-1)} + \mathbf{G}^{(t)})) \\ \mathbf{M}^{(t)} &= \mu \mathbf{M}^{(t-1)} + (1 - \mu) \frac{\mathbf{X}^{(t)} - \mathbf{X}^{(t+1)}}{\eta}. \end{aligned} \quad (9.3)$$

9.4.3 Convergence Analysis

We provide a convergence analysis for our novel QG-DSGDm for non-convex functions. The proof details can be found in Appendix 18.3.

Assumption 9. We assume that the following hold:

- The function $f(\mathbf{x})$ we are minimizing is lower bounded from below by f^* , and each node's loss f_i is smooth satisfying $\|\nabla f_i(\mathbf{y}) - \nabla f_i(\mathbf{x})\| \leq L\|\mathbf{y} - \mathbf{x}\|$.
- The stochastic gradients within each node satisfy $\mathbb{E}[g_i(\mathbf{x})] = \nabla f_i(\mathbf{x})$ and $\mathbb{E}\|g_i(\mathbf{x}) - \nabla f_i(\mathbf{x})\|^2 \leq \sigma^2$. The variance across the workers is also bounded as $\frac{1}{n} \sum_{i=1}^n \|\nabla f_i(\mathbf{x}) - \nabla f(\mathbf{x})\|^2 \leq \zeta^2$.
- The mixing matrix is doubly stochastic: for the all ones vector $\mathbf{1}$, we have $\mathbf{W}\mathbf{1} = \mathbf{1}$ and $\mathbf{W}^\top \mathbf{1} = \mathbf{1}$.
- Define $\bar{\mathbf{Z}} = \mathbf{Z}_n \frac{1}{n} \mathbf{1}\mathbf{1}^\top$ for any matrix $\mathbf{Z} \in \mathbb{R}^{d \times n}$, then the mixing matrix satisfies $\mathbb{E}_{\mathbf{W}} \|\mathbf{Z}\mathbf{W} - \bar{\mathbf{Z}}\|_F^2 \leq (1 - \rho) \|\mathbf{Z} - \bar{\mathbf{Z}}\|_F^2$.

Theorem 9.4.1 (Convergence of QG-DSGDm for non-convex functions). Given Assumption 9, the sequence of iterates generated by (9.3) for step size $\eta = \mathcal{O}\left(\sqrt{\frac{n}{\sigma^2 T}}\right)$ and momentum parameter $\frac{\beta}{1-\beta} \leq \frac{\rho}{21}$ satisfies $\frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E} \|\nabla f(\bar{\mathbf{x}}^t)\|^2 \leq \epsilon$ in iterations

$$T = \mathcal{O}\left(\frac{L\sigma^2}{n\epsilon^2} + \frac{L\tilde{\zeta}}{\rho\epsilon^{3/2}} + \frac{L}{\epsilon}\left(\frac{1}{\rho} + \frac{1}{(1-\mu)(1-\beta)^2}\right)\right),$$

where $\tilde{\zeta}^2 := \zeta^2 + (1 + \frac{1-\beta}{1-\mu})\sigma^2$.

Remark 9.4.2. The asymptotic number of iterations required, $\mathcal{O}\left(\frac{\sigma^2}{n\epsilon^2}\right)$ shows perfect linear speedup in the number of workers n , independent of the communication topology. This upper bound matches the convergence bounds of DSGD (Lian et al., 2017) and centralized mini-batch SGD (Dekel et al., 2012), and is optimal (Arjevani et al., 2019). This significantly improves over previous analyses of distributed momentum methods which need $\frac{L\sigma^2}{n(1-\beta)\epsilon^2}$ iterations, slowing down for larger values of β (Yu et al., 2019a; Balu et al., 2021). The second drift term $\frac{1}{\rho\epsilon^{3/2}}$ arises due to the non-iid data distribution, and matches the tightest analysis of DSGD without momentum (Koloskova et al., 2020b). Finally, our theorem imposes some constraint on the momentum parameter β (but not on μ). In practice however, QG-DSGDm performs well even when this constraint is violated.

9.4.4 Connection with Other Methods

We bridge quasi-global momentum with two recent works below. The corresponding algorithm details are included in Appendix 18.2.1 for clarity.

Connection with MimeLite. MimeLite (Karimireddy et al., 2021) was recently introduced for FL on heterogeneous data. It shares a similar ingredient as ours: a “global” movement direction \mathbf{d} is used locally to alleviate the issue caused by heterogeneity. The difference falls into the way

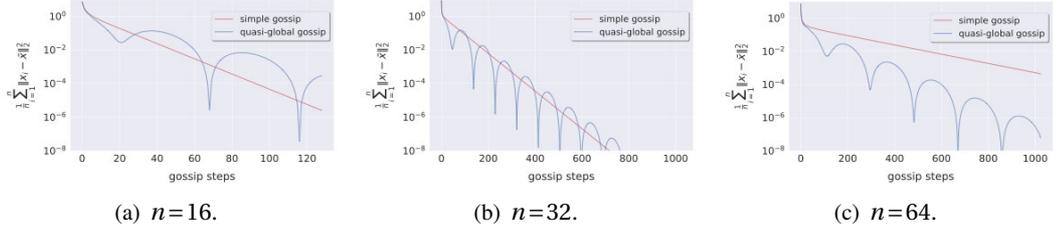


Figure 9.3 – Understanding QG-DSGDm through the distributed average consensus problem on a fixed ring topology. QG-DSGDm without gradient update step (9.4) still presents faster convergence (to a relative high precision) than the standard gossip averaging. Appendix 18.4.1 illustrates the results on other communication topologies and topology scales.

of forming \mathbf{d} (c.f. line 8 in Algorithm 6): in MimeLite, \mathbf{d} is the full batch gradients computed on the previously synchronized model, while the \mathbf{d} in our QG-DSGDm is the difference on two consecutive synchronized models.

MimeLite only addresses the FL setting, which results in a computation and communication overhead (to form \mathbf{d}), and is non-trivial to extend to decentralized learning.

Connection with SlowMo. SlowMo and its “noaverage” variant (Wang et al., 2020d) aim to improve generalization performance in the homogeneous data-center training scenario, while QG-DSGDm is targeting learning with data heterogeneity. In terms of update scheme, SlowMo variants update the slow momentum buffer through the model difference \mathbf{d} of $\tau \gg 1$ local update (and synchronization) steps, while QG-DSGDm only considers consecutive models (analogously $\tau = 1$).³ Besides, in contrast to QG-DSGDm, the slow momentum buffer in SlowMo will never interact with the local update—setting τ to 1 in SlowMo variants cannot recover QG-DSGDm. SlowMo variants are orthogonal to QG-DSGDm; combining these two algorithms may lead to a better generalization performance, and we leave it for future work.

9.5 Understanding QG-DSGDm

9.5.1 Faster Convergence in Average Consensus

We now consider the simpler averaging consensus problem (isolated from the learning part of QG-DSGDm): we simplify (9.3) by removing gradients and step-size:

$$\begin{aligned} \mathbf{X}^{(t+1)} &= \mathbf{W}(\mathbf{X}^{(t)} - \beta \mathbf{M}^{(t-1)}) \\ \mathbf{M}^{(t)} &= \mu \mathbf{M}^{(t-1)} + (1 - \mu)(\mathbf{X}^{(t)} - \mathbf{X}^{(t+1)}), \end{aligned} \quad (9.4)$$

and compare it with gossip averaging $\mathbf{X}^{(t+1)} = \mathbf{W}\mathbf{X}^{(t)}$.

Figure 9.3 depicts the advantages of (9.4) over standard gossip averaging, where QG-DSGDm can quickly converge to a critical consensus distance (e.g. 10^{-2}). It partially explains the performance

³ We also study the variant of QG-DSGDm with $\tau > 1$ in Appendix 18.4.6—we stick to $\tau = 1$ in the main paper for its superior performance and hyperparameter (τ) tuning free.

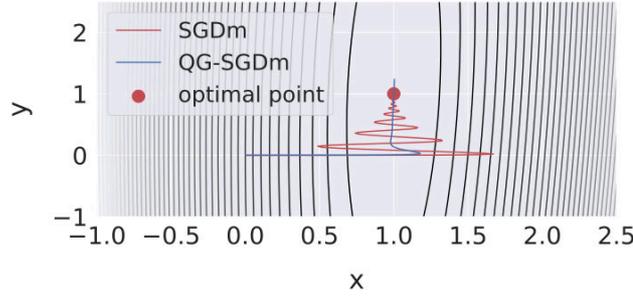


Figure 9.4 – **Understanding the optimization trajectory of QG-SGDm and SGDm** (i.e. single worker case) via a 2D toy function $f(x, y) = (y - x^2)^2 + 100(x - 1)^2$. This function has a global minimum at $(x, y) = (1, 1)$. SGDm and QG-SGDm use $\beta = 0.9, \eta = 0.001$, with initial point $(0, 0)$. For additional trajectories with various initial points and/or β values refer to Appendix 18.4.8.

gain of QG-DSGDm from the aspect of improved decentralized communication (which leads to better optimization)—decentralized training can converge as fast as its centralized counterpart once the consensus distance is lower than the critical one, as stated in Kong et al. (2021a) (see Chapter 5).

9.5.2 QG-DSGDm (Single Worker Case) Recovers QHM

Considering the single worker case, QG-DSGDm can be further simplified to (derivations in Appendix 18.2.3):

$$\begin{aligned} \hat{\mathbf{m}}^{(t)} &= \hat{\beta} \hat{\mathbf{m}}^{(t-1)} + \mathbf{g}^{(t)} \\ \mathbf{x}^{(t+1)} &= \mathbf{x}^{(t)} - \eta \left(\left(1 - \frac{\mu}{\hat{\beta}}\right) \hat{\mathbf{m}}^{(t)} + \frac{\mu}{\hat{\beta}} \mathbf{g}^{(t)} \right), \end{aligned}$$

where $\hat{\beta} := \mu + (1 - \mu)\beta$. Thus, the single worker case of QG-DSGDm (i.e. QG-SGDm) recovers Quasi-Hyperbolic Momentum (QHM) (Ma and Yarats, 2019; Gitman et al., 2019). We illustrate its acceleration benefits as well as the performance gain in Figure 18.8 and Figure 18.9 of Appendix 18.4.7. We elaborate in Appendix 18.2.3 that SGDm is only a special case of QG-SGDm/QHM (by setting $\mu = 0$). Besides, it is non-trivial to adapt (centralized) QHM to (decentralized) QG-DSGDm due to discrepant motivation.

Stabilized optimization trajectory. We study the optimization trajectory of Rosenbrock function (Rosenbrock, 1960) $f(x, y) = (y - x^2)^2 + 100(x - 1)^2$ as in Lucas et al. (2019) to better understand the performance gain of QG-SGDm (with zero stochastic noise).⁴ Figure 18.12 illustrates the effects of stabilization in QG-SGDm (much less oscillation than SGDm).

⁴ We further study the optimization trajectory for more complicated non-convex function in Appendix 18.4.8.

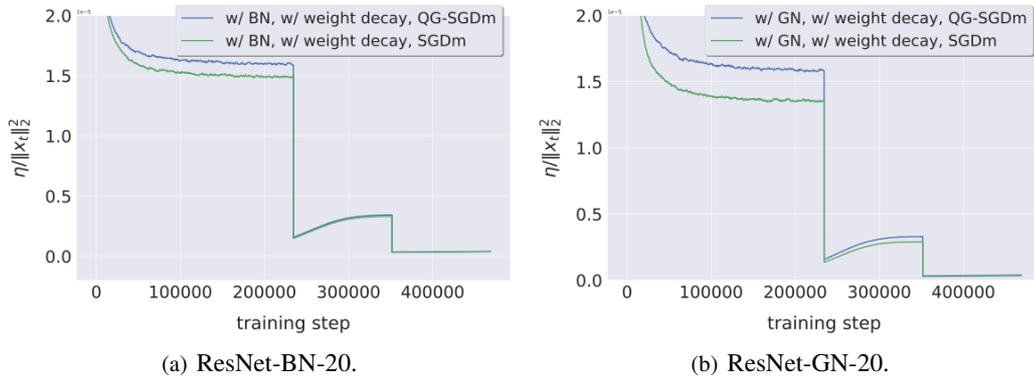


Figure 9.5 – **The effective step-size $\eta/\|\mathbf{x}_t\|_2^2$ of QG-SGDm and SGDm** (single worker case) on CIFAR-10. For the weight norm curves refer to Figure 18.10 in Appendix 18.4.7.

Larger effective step-size. Recent works (Hoffer et al., 2018; Zhang et al., 2019b) point out the larger effective step-size (i.e. $\eta/\|\mathbf{x}_t\|_2^2$) brought by weight decay provides the primary regularization effect for deep learning training. Figure 9.5 examines the effective step-size during the optimization procedure: QG-SGDm illustrates a larger effective step-size than SGDm, explaining the performance gain e.g. in Figure 18.8 and Figure 18.9 of Appendix 18.4.7.

9.6 Experiments

9.6.1 Setup

Datasets and models. We empirically study the decentralized training behavior on both CV and NLP benchmarks, on the architecture of ResNet (He et al., 2016a), VGG (Simonyan and Zisserman, 2015) and DistilBERT (Sanh et al., 2019). • Image classification (CV) benchmark: we consider training CIFAR-10 (Krizhevsky and Hinton, 2009), ImageNet-32 (i.e. image resolution of 32) (Chrabaszcz et al., 2017), and ImageNet (Deng et al., 2009) from scratch, with standard data augmentation and preprocessing scheme (He et al., 2016a). We use VGG-11 (with width factor 1/2 and without BN) and ResNet-20 for CIFAR-10, ResNet-20 with width factor 2 (noted as ResNet-20-x2) for ImageNet-32, and ResNet-18 for ImageNet. The width factor indicates the proportional scaling of the network width corresponding to the original neural network. Weight initialization schemes follow (He et al., 2015; Goyal et al., 2017). • Text classification (NLP) benchmark: we perform fine-tuning on a 4-class classification dataset (AG News (Zhang et al., 2015)). Unless mentioned otherwise, all our experiments are repeated over three random seeds. We report the averaged performance of local models on the full test dataset.

Heterogeneous distribution of client data. We use the Dirichlet distribution to create disjoint non-i.i.d. client training data (Yurochkin et al., 2019; Hsu et al., 2019; He et al., 2020b)—the created client data is fixed and never shuffled across clients during the training. The degree of

non-i.i.d.-ness is controlled by the value of α ; the smaller α is, the more likely the clients hold examples from only one class. An illustration regarding how samples are distributed among 16 clients on CIFAR-10 can be found in Figure 9.1; more visualizations on other datasets/scales are shown in Appendix 18.1.2. Besides, Figure 18.1 in Appendix 18.1.1 visualizes the Social Network topology.

Training schemes. Following the SOTA deep learning training scheme, we use mini-batch SGD as the base optimizer for CV benchmark (He et al., 2016a; Goyal et al., 2017), and similarly, Adam for NLP benchmark (Zhang et al., 2020; Mosbach et al., 2021). In Section 9.6.2, we adapt these base optimizers to various distributed variants.⁵

For the CV benchmark, the models are trained for 300 and 90 epochs for CIFAR-10 and ImageNet(-32) respectively; the local mini-batch size are set to 32 and 64. All experiments use the SOTA learning rate scheme in distributed deep learning training (Goyal et al., 2017; He et al., 2019a) with learning rate scaling and warm-up. The learning rate is always gradually warmed up from a relatively small value (i.e. 0.1) for the first 5 epochs. Besides, the learning rate will be divided by 10 when the model has accessed specified fractions of the total number of training samples— $\{\frac{1}{2}, \frac{3}{4}\}$ for CIFAR and $\{\frac{1}{3}, \frac{2}{3}, \frac{8}{9}\}$ for ImageNet.

For the NLP benchmark, we fine-tune the *distilbert-base-uncased* from HuggingFace (Wolf et al., 2019) with constant learning rate and mini-batch of size 32 for 10 epochs.

We fine-tune the learning rate for both CV⁶ and NLP tasks; we use constant weight decay (1e-4). The tuning procedure ensures that the best hyperparameter lies in the middle of our search grids; otherwise we extend our search grid.

Regarding momentum related hyperparameters, we follow the common practice in the community ($\beta=0.9$ and without dampening for Nesterov/HeavyBall momentum variants, and $\beta_1=0.9, \beta_2=0.99$ for Adam variants).

BN and its alternatives for distributed deep learning. The existence of BN layer is challenging for the SOTA distributed training, especially for heterogeneous data setting. To better understand the impact of various normalization schemes in distributed deep learning, we investigate:

- Distributed BN implementation. Our default implementation⁷ follows (Goyal et al., 2017; Andreux et al., 2020) that computes the BN statistics independently for each client while only synchronizing the BN weights.

⁵ We by default use local momentum variants without buffer synchronization. We consider DSGDm-N as our primary competitor for CNNs, as Nesterov momentum is the SOTA training scheme. We also investigate the performance of DSGDm in Table 9.6.

⁶ We tune the initial learning rate and warm it up from 0.1 (if the tuned one is above 0.1).

⁷ We also try the BN variant (Li et al., 2021) proposed for FL, but we exclude it in our comparison due to its poor performance.

9.6. Experiments

Table 9.1 – **The test top-1 accuracy of decentralized optimization algorithms evaluated on various degrees of non-i.i.d. local CIFAR-10 data, for various neural architectures and network topologies.** The results are averaged over three random seeds, with learning rate tuning for each setting. We also include the results of centralized baseline for reference purposes, following the decentralized experiment configuration, except that the centralized baseline uses randomly partitioned local training data (i.e. independent of α).

Datasets	Neural Architectures	Methods	Ring ($n=16$)			Social Network ($n=32$)		
			$\alpha = 10$	$\alpha = 1$	$\alpha = 0.1$	$\alpha = 10$	$\alpha = 1$	$\alpha = 0.1$
CIFAR-10	ResNet-BN-20	SGDm-N (centralized)		92.95 ± 0.13			92.88 ± 0.07	
		DSGD	90.94 ± 0.15	88.95 ± 0.59	54.66 ± 3.58	90.52 ± 0.24	89.22 ± 0.35	58.32 ± 3.27
		DSGDm-N	92.53 ± 0.27	89.13 ± 0.81	57.19 ± 2.65	92.20 ± 0.24	90.19 ± 0.54	63.00 ± 2.50
		QG-DSGDm-N	92.65 ± 0.17	91.21 ± 0.28	58.16 ± 3.32	92.52 ± 0.09	91.20 ± 0.16	64.32 ± 2.43
	ResNet-GN-20	SGDm-N (centralized)		88.06 ± 1.12			86.19 ± 1.08	
		DSGD	86.86 ± 0.37	85.93 ± 0.14	73.14 ± 3.92	84.00 ± 0.67	82.98 ± 0.47	67.84 ± 3.94
		DSGDm-N	89.86 ± 0.15	88.30 ± 0.49	71.86 ± 2.22	88.54 ± 0.22	86.36 ± 0.55	72.02 ± 1.79
		QG-DSGDm-N	90.18 ± 0.44	89.68 ± 0.41	82.78 ± 2.05	88.58 ± 0.09	88.19 ± 0.30	83.60 ± 1.83
	ResNet-EvoNorm-20	SGDm-N (centralized)		92.18 ± 0.19			91.92 ± 0.33	
		DSGD	89.90 ± 0.26	88.88 ± 0.26	74.55 ± 2.07	89.95 ± 0.23	88.41 ± 0.27	77.56 ± 1.65
		DSGDm-N	91.47 ± 0.23	89.98 ± 0.10	77.48 ± 2.67	91.17 ± 0.11	89.96 ± 0.35	80.59 ± 2.32
		QG-DSGDm-N	91.90 ± 0.17	91.28 ± 0.38	82.20 ± 1.27	91.51 ± 0.02	91.00 ± 0.24	85.19 ± 0.98
	VGG-11 (w/o normalization layer)	SGDm-N (centralized)		88.87 ± 0.29			87.38 ± 0.39	
		DSGDm-N	88.68 ± 0.30	88.52 ± 0.24	77.45 ± 3.15	86.39 ± 0.06	85.85 ± 0.22	77.02 ± 2.66
		QG-DSGDm-N	89.01 ± 0.04	89.00 ± 0.22	83.41 ± 2.20	86.87 ± 0.60	86.09 ± 0.30	84.86 ± 0.58

- Using other normalization layers: for instance on ResNet with BN layers (denoted by ResNet-BN-20), we can instead use ResNet-GN by replacing all BN with GN with group number of 2, as suggested in [Hsieh et al. \(2020\)](#). We also examine the recently proposed S0 variant of EvoNorm ([Liu et al., 2020b](#)) (which does not use runtime mini-batches statistics), noted as ResNet-EvoNorm.

9.6.2 Results

Comments on BN and its alternatives. Table 9.1 and Table 9.2 examine the effects of BN and its alternatives on the training quality of decentralized deep learning on CIFAR-10 and ImageNet dataset. *ResNet with EvoNorm replacement outperforms its GN counterpart on a spectrum of optimization algorithms, non-i.i.d. degrees, and network topologies, illustrating its efficacy to be a new alternative to BN in CNNs for distributed learning on heterogeneous data.*

Superior performance of quasi-global momentum. We evaluate QG-DSGDm-N and compare it with several DSGD variants in Table 9.1, for training various neural networks on CIFAR-10 in terms of various non-i.i.d. degrees on Ring ($n=16$) and Social Network ($n=32$). *QG-DSGDm-N accelerates the training by stabilizing the oscillating optimization trajectory caused by heterogeneity and leads to a significant performance gain over all other strong competitors on all levels of data heterogeneity. The benefits of our method are further pronounced when considering a higher degree of non-i.i.d.-ness.* These observations are consistent with the results on the challenging ImageNet(-32) dataset in Table 9.2 (and the learning curves in Figure 18.6 in Appendix 18.4.3).

Chapter 9. Accelerating Decentralized Deep Learning on Heterogeneous Data

Table 9.2 – **Test top-1 accuracy of decentralized optimization algorithms evaluated on various degrees of non-i.i.d. local ImageNet data.** The results are over three random seeds. We perform sufficient learning rate tuning on ImageNet-32 for each setup while we use the same one for ImageNet due to the computational feasibility. “★” indicates non-convergence.

Datasets	Neural Architectures	Methods	Ring ($n=16$)	
			$\alpha = 1$	$\alpha = 0.1$
ImageNet-32 (resolution 32)	ResNet-20-x2 (EvoNorm)	SGDm-N (centralized)	44.43 ± 0.20	
		DSGDm-N	30.35 ± 0.05	16.71 ± 0.17
		QG-DSGDm-N	31.24 ± 0.27	19.53 ± 0.91
	ResNet-20-x2 (GN)	SGDm-N (centralized)	37.89 ± 0.67	
		DSGDm-N	34.16 ± 1.37	★
		QG-DSGDm-N	38.57 ± 0.45	21.42 ± 0.81
ImageNet	ResNet-18 (EvoNorm)	SGDm-N (centralized)	69.55 ± 0.25	
		DSGDm-N	68.77 ± 0.05	53.15 ± 0.14
		QG-DSGDm-N	69.20 ± 0.08	56.50 ± 0.01
	ResNet-18 (GN)	SGDm-N (centralized)	62.59 ± 0.01	
		DSGDm-N	60.76 ± 0.48	39.57 ± 1.22
		QG-DSGDm-N	64.92 ± 0.27	47.86 ± 1.05

Table 9.3 – **Test top-1 accuracy of decentralized SGD algorithms evaluated on various degrees of non-i.i.d.-ness and communication topologies,** for training ResNet-EvoNorm-18 on ImageNet. The results are over three random seeds. We use the same learning rate for various experiments due to the computational feasibility. Centralized SGDm-N reaches 69.55 ± 0.25.

Communication Topology	Methods	Test Top-1 Accuracy	
		$\alpha = 1$	$\alpha = 0.1$
Ring ($n=16$)	DSGDm-N	68.77 ± 0.05	53.15 ± 0.14
	QG-DSGDm-N	69.20 ± 0.08	56.50 ± 0.01
1-peer directed exponential graph ($n=16$) (Assran et al., 2019)	DSGDm-N	69.00 ± 0.11	58.52 ± 0.27
	QG-DSGDm-N	69.34 ± 0.17	61.44 ± 0.20

Decentralized Adam. We further extend the idea of quasi-global momentum to the Adam optimizer for decentralized learning, noted as QG-DAdam (the algorithm details are deferred to Algorithm 19 in Appendix 18.2.1). We validate the effectiveness of QG-DAdam over D(decentralized)Adam in Table 9.4, on fine-tuning DistilBERT on AG News and training ResNet-EvoNorm-20 on CIFAR-10 from scratch: *QG-DAdam is still preferable over DAdam.* We leave a better adaptation and theoretical proof for future work.

Generalizing quasi-global momentum to time-varying topologies. The benefits of quasi-global momentum are not limited to the fixed and undirected communication topologies, e.g. Ring and Social network in Table 9.1—it also generalizes to other topologies, like the time-varying directed topology (Assran et al., 2019), as shown in Table 9.3 for training ResNet-EvoNorm-18 on ImageNet. These results are aligned with the findings on the influence of the consensus distance on the generalization performance of decentralized deep learning (Kong et al., 2021a) (as in Chapter 5), supporting the fact that *quasi-global momentum can be served as a simple*

9.6. Experiments

Table 9.4 – **Test accuracy of decentralized optimization algorithms (with Adam), evaluated on various degrees of non-i.i.d. local data.** The results are over three random seeds, with tuned learning rate.

Models & Datasets	Methods	$\alpha = 0.1$
Fine-tuning DistilBERT-base (AG News)	DAdam	87.29 \pm 0.60
	QG-DAdam	88.33 \pm 0.67
Training ResNet-EvoNorm-20 from scratch (CIFAR-10)	DAdam	65.52 \pm 3.32
	QG-DAdam	66.86 \pm 2.81

Table 9.5 – **Comparison with Gradient Tracking (GT) methods** for training CIFAR-10. D^2 and D_+^2 do not include the momentum acceleration. We carefully tune the learning rate for each case, and results are averaged over three seeds where std is indicated.

	ResNet-EvoNorm-20 on Ring ($n=16$)					ResNet-EvoNorm-20 on Ring ($n=32$)			
	DSGD (w/ GT)	DSGDm-N	DSGDm-N (w/ GT)	D^2	D_+^2	QG-DSGDm-N	DSGDm-N	DSGDm-N (w/ GT)	QG-DSGDm-N
$\alpha = 1$	87.36 \pm 0.40	89.98 \pm 0.10	90.38 \pm 0.41	74.89	85.70 \pm 0.29	91.28 \pm 0.38	88.46 \pm 0.29	89.44 \pm 0.60	90.27 \pm 0.27
$\alpha = 0.1$	66.16 \pm 1.05	77.48 \pm 2.67	78.64 \pm 1.84	49.80	69.18 \pm 3.30	82.20 \pm 1.27	78.17 \pm 1.63	79.25 \pm 2.17	83.18 \pm 1.11

Table 9.6 – **An extensive investigation for a wide spectrum of DSGD variants**, for training ResNet-EvoNorm-20 on CIFAR-10. The results are averaged over three seeds, each with learning rate tuning. We use “communication topology” to synchronize the model parameters, while some methods involve “extra communication”, with specified objective to be communicated on the given network topology.

Methods	Communication Topology	Extra Communication	Momentum Type	Test Top-1 Accuracy ($n=16$)	
				$\alpha = 1$	$\alpha = 0.1$
SGDm-N	complete	-	global	92.18 \pm 0.19	
DSGDm-N	complete	-	local	91.47 \pm 0.10	71.24 \pm 3.08
DSGDm-N	ring	momentum buffer (complete)	local	90.96 \pm 0.33	81.22 \pm 1.78
SlowMo	ring	model parameters (complete)	local & global	91.06 \pm 0.26	79.20 \pm 1.16
DSGD	ring	-	-	88.88 \pm 0.26	74.55 \pm 2.07
DSGDm	ring	-	local	89.67 \pm 0.33	77.66 \pm 0.95
DSGDm-N	ring	-	local	89.98 \pm 0.10	77.48 \pm 2.67
DSGDm	ring	momentum buffer (ring)	local	90.42 \pm 0.32	78.69 \pm 2.39
DSGDm-N	ring	momentum buffer (ring)	local	90.48 \pm 0.67	79.83 \pm 2.29
DSGDm-N	ring	local gradients (ring)	local	90.10 \pm 0.61	78.58 \pm 4.12
DMSGD	ring	-	local	90.06 \pm 0.04	79.89 \pm 0.97
QG-DSGDm	ring	-	local	91.22 \pm 0.41	82.24 \pm 1.05
QG-DSGDm-N	ring	-	local	91.28 \pm 0.38	82.20 \pm 1.27

plugin to further improve the performance of decentralized deep learning.

Comparison with D^2 and Gradient Tracking (GT). As shown in Table 9.5, D^2 (Tang et al., 2018b) and GT methods (Pu and Nedić, 2020; Pan et al., 2020; Lu et al., 2019) cannot achieve comparable test performance on the standard deep learning benchmark, while QG-DSGDm-N outperforms them significantly. Additional detailed comparisons are deferred to Appendix 18.2.4. It is non-trivial to integrate D^2 with momentum. Besides, D^2 requires constant learning rate,

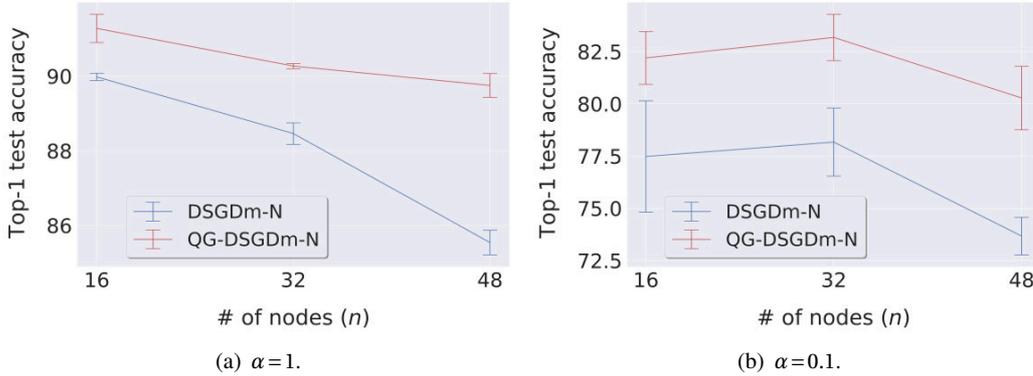


Figure 9.6 – Test top-1 accuracy of decentralized algorithms evaluated on various topology scales and non-i.i.d. degrees, for training ResNet-EvoNorm-20 on CIFAR-10. The results are over three random seeds, each with sufficient learning rate tuning. Colors blue and red indicate DSGDm-N and QG-DSGDm-N respectively. Numerical results refer to Table 18.1 in Appendix 18.4.5.

which does not fit the SOTA learning rate schedules (e.g. stage-wise) in deep learning.⁸ We include an improved D^2 variant⁹ (denoted as D^2_+) to address this learning rate decay issue in D^2 , but the performance of D^2_+ still remains far behind our scheme.

Ablation study. Table 9.6 empirically investigates a wide range of DSGD variants, in terms of the generalization performance on various degrees of data heterogeneity. We can witness that (1) DSGD variants with quasi-global momentum always significantly surpass all other methods (excluding the centralized upper bound), without introducing extra communication cost; (2) local momentum accelerates the decentralized optimization (c.f. the results of DSGD v.s. DSGDm and DSGDm-N), while our quasi-global momentum further improves the performance gain; (3) synchronizing local momentum buffer or local gradients only marginally improves the generalization performance, but the gains fall behind our quasi-global momentum (as we accelerate the consensus and stabilize trajectories, as illustrated in Section 9.5); (4) the parallel work DMSGD¹⁰ (Balu et al., 2021) does show some improvements, but its performance gain is much less significant than ours. Table 18.7 in the Appendix 18.4.4 further shows that tuning momentum factor for DSGDm-N cannot alleviate the training difficulty caused by data heterogeneity.

Besides, Figure 9.6 showcases the generality of quasi-global momentum for achieving remarkable performance gain on various topology scales and non-i.i.d. degrees.

⁸ D^2 can be rewritten as $\mathbf{w}(\mathbf{x}^{(t)} - \eta((\mathbf{x}^{(t-1)} - \mathbf{x}^{(t)})/\eta + \nabla f(\mathbf{x}^{(t)}) - \nabla f(\mathbf{x}^{(t-1)})))$, and the update would break if the magnitude of $\mathbf{x}^{(t-1)} - \mathbf{x}^{(t)}$ is a factor of 10η (i.e. performing learning rate decay at step t).

⁹The update scheme of D^2_+ follows $\mathbf{w}(\mathbf{x}^{(t)} - \eta^{(t)}((\mathbf{x}^{(t-1)} - \mathbf{x}^{(t)})/\eta^{(t-1)} + \nabla f(\mathbf{x}^{(t)}) - \nabla f(\mathbf{x}^{(t-1)})))$.

¹⁰We tune both learning rate η and weighting factor μ (using the grid suggested in Balu et al. (2021)) for DMSGD (option D).

Conclusion

We demonstrated that heterogeneity has an out sized impact on the performance of deep learning models, leading to unstable convergence and poor performance. We proposed a novel momentum-based algorithm to stabilize the training and established its efficacy through thorough empirical evaluations. Our method, especially for mildly heterogeneous settings, leads to a 10–20% increase in accuracy. However, a gap still remains between the centralized training. Closing this gap, we believe, is critical for wider adoption of decentralized learning.

10 Conclusion and Future Work

This thesis offers some encouraging results from my research to address the fundamental challenges of efficiency and robustness faced in distributed deep learning, covering i) understanding and benchmarking communication efficient local SGD (Chapter 2), ii) offering communication efficient and effective strategies to alleviate the distributed large-batch generalization difficulties in data centers (Chapter 2 and Chapter 3), iii) a seminal decentralized deep learning algorithm that allows for arbitrary communication compression (Chapter 4), iv) a theoretical and empirical understanding on bridging the performance gap between centralized and decentralized deep learning (Chapter 5), v) a dynamic model pruning technique (during training) for computationally efficient learning and deployment (Chapter 6), vi) a computationally and memory efficient way to fine-tune large pre-trained language models (Chapter 7), vii) a robust knowledge aggregation scheme for federated learning on edge devices with heterogeneous hardware/data (Chapter 8), and viii) a communication efficient solution to address the challenging decentralized learning scenario on heterogeneous data (Chapter 9).

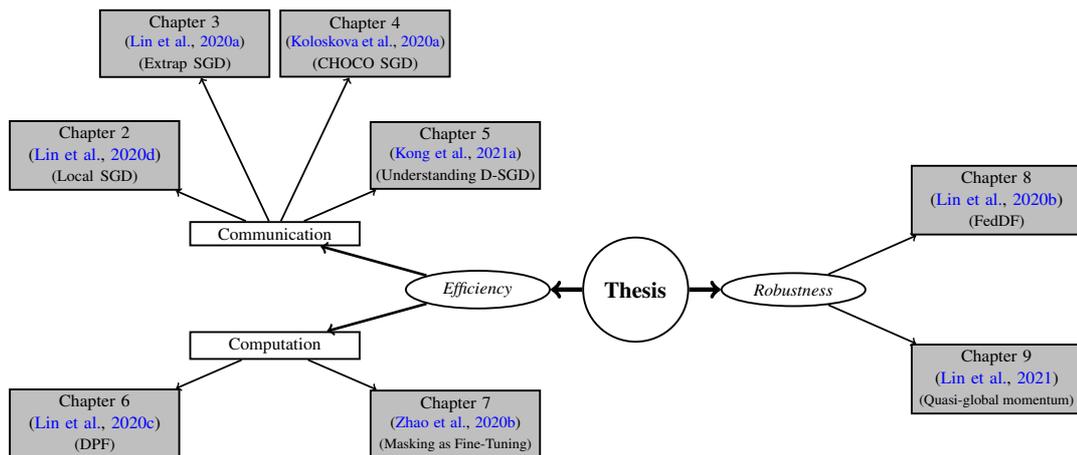


Figure 10.1 – Thesis outline.

However, the challenges for efficient and robust distributed deep learning are far from being

solved: we still need to work on developing distributed deep learning algorithms for all data modalities, which can be run in data centers and on edge devices, that are both efficient and resistant to real-world challenges such as heterogeneous computing systems, malicious actors, and data distribution shifts. We outline a few of such potential research directions below:

Efficiency

Efficient learning and deployment are crucial for both data centers and edge devices. While there has been extensive research by many groups on the efficiency of (distributed) deep learning, these progresses are still far from fulfilling the deployment demands in practice. The gaps might result from the discrepancy between empirical deep learning observations and optimization/generalization theory, the mismatch between tasks considered in academics and the real-world issues, etc. The future research here will focus specifically on approaches that can close these gaps, for example,

- Despite our theoretically-sound error feedback framework ([Stich and Karimireddy, 2020](#); [Lin et al., 2020c](#)) now being the workhorse strategy for bandwidth-efficient distributed learning, the successes observed in data centers remain non-trivial to apply to cross-device¹ federated learning, due to the very low client involvement ratio in practice and the violated stateful requirement of the framework: one of the future research steps could be to overcome this restriction e.g. by generalizing our momentum-based techniques.
- Decentralized learning holds huge promise for improved communication efficiency as well as for enabling privacy-preserving training. This thesis and my current research group have contributed some crucial building blocks ([Koloskova et al., 2020a](#); [Lin et al., 2021, 2020b](#); [Vogels et al., 2021](#); [Guo et al., 2021](#)) to push this domain forward. A key research task left is to standardize decentralized learning towards broader applicability, while requiring less hyperparameter tuning, as to become a drop-in replacement for the corresponding centralized training algorithms. We will work on this open challenge and contribute a family of easy-to-use decentralized algorithms, as part of steps to enable hybrid learning in data centers and on edge devices.
- Resource-constrained distributed on-device learning in practice could strongly benefit from a memory-efficient collaborative transfer learning approach to continually adapt pre-trained models to the newly collected client data. This pursuit is in contrast to the current research focus of the community, which either collaboratively trains the model from scratch (but without resource constraints), or solely designs efficient on-device inference techniques (thus only considering on-device deployment and without learning), or efficient on-device learning (but without distributed collaborative learning). This gap also stimulates the co-design of distributed collaborative optimization algorithms and learning systems. One promising direction is the collaborative subspace learning—a unified model compression approach ([Lin et al.,](#)

¹ The cross-device setting indicates that (1) the clients are a very large number of mobile or IoT devices, (2) only a fraction of clients are available at any one time, and (3) clients may be stateless.

2020c)—in which optimizing a neural network in a small subspace of parameters, rather than its full dense space, solves the computing and memory issues that full dense space learning raises while maintaining similar generalization performance.

To the best of our knowledge, none of the currently existing methods are sufficient to address the above-mentioned challenges, motivated by different real-world constraints.

Robustness

Robustness to various kinds of heterogeneity raised in practice such as data heterogeneity and system heterogeneity is required for the development of trustworthy decentralized or federated deep learning applications. Though decades of study on obstacles linked to heterogeneity has resulted from the enormous literature on computer systems, the issues faced by these traditional *non-learnable* systems are fundamentally different from those encountered by the *learnable* scenarios. Divergent key objectives can explain this: prior studies in computer systems directly optimize the system performance itself, whereas we will focus on improving learning algorithms (rather than the system) to alleviate these heterogeneity constraints and aim to achieve a similar generalization performance as the centralized counterpart.

The heterogeneity challenges that decentralized/federated learning raises are diverse in multiple aspects, including, but not limited to,

- *Training data heterogeneity*: the heterogeneous local data presented on edge devices causes client drift in the updates of each device, resulting in slow and unstable learning. A line of research has emerged in the recent one to two years in federated learning to address the time-invariant data heterogeneity. However, most theoretically sound approaches are far from sufficient for dealing with complicated deep learning tasks in reality, and most practical deep learning methods may lack appropriate assessment, due to the missing benchmarks in the field. The concerns also apply to more difficult decentralized learning scenarios, such as involving local SGD (with multiple local update steps) and data heterogeneity in decentralized deep learning, for which there is currently no solution. Furthermore, the time-varying nature of local client datasets has not been investigated by the community yet. Despite our very recent attempt to model such a scenario (Guo et al., 2021), the current theoretical analyses are still far from tight enough to address the real-world challenges. Our future research aims to advance heterogeneity-independent distributed on-device learning, by addressing the above-mentioned challenges.
- *The distribution shifts (heterogeneity) between training and deployment*. Learning on time-varying heterogeneous client data is inherently correlated with the distribution shifts when deployed in the wild. Despite there exists active research on distribution robust methods, these methods are mostly designed for *non-distributed* and *non-collaborative* deep learning tasks, and without taking into account practical restrictions. We will bridge the gaps between existing

Chapter 10. Conclusion and Future Work

approaches and real-world challenges on edge devices—all of which have not been addressed in previous research—by considering factors such as constraints from communication and computation, limited access to the multiple domains simultaneously, progressive interaction between (continual) distributed learning and deployment, etc.

- *Training system heterogeneity*: the variation in processing power and energy, device memory and storage, and network bandwidth and latency. Besides, a decentralized/federated learning system deployed on the client side naturally is dynamic and vulnerable, e.g. in the case of unreliable client devices and communication networks, unpredictable client participation patterns. These types of system heterogeneity, not only burden the system efficiency, but also hinder optimization and generalization for distributed on-device learning. Despite its prevalence and importance, research in this area is limited, making it difficult to achieve distributed on-device learning that is resistant to many forms of heterogeneity mentioned above. We will develop learning algorithms that are resilient to the (1) network topology, and (2) intermittent devices with behaviors of disappearing, straggling, or re-appearing, as well as efficient and effective model aggregation schemes over different neural architectures.

Distributed collaborative learning algorithms have a rich history and there exist many research groups working on improving the efficiency and robustness of deep learning training on distributed systems. However, the above-mentioned directions are far from being fully explored, and the present solutions are not yet suitable for use in practice.

Appendices **Part IV**

11 Appendix for Local SGD Variants

11.1 Details on Deep Learning Experimental Setup

11.1.1 Dataset

We consider the following tasks.

- Image classification for CIFAR-10/100 (Krizhevsky and Hinton, 2009). Each consists of a training set of 50K and a test set of 10K color images of 32×32 pixels, as well as 10 and 100 target classes respectively. We adopt the standard data augmentation scheme and preprocessing scheme (He et al., 2016a; Huang et al., 2016). For preprocessing, we normalize the data using the channel means and standard deviations.
- Image classification for ImageNet (Russakovsky et al., 2015). The ILSVRC 2012 classification dataset consists of 1.28 million images for training, and 50K for validation, with 1K target classes. We use ImageNet-1k (Deng et al., 2009) and adopt the same data preprocessing and augmentation scheme as in He et al. (2016a,b); Simonyan and Zisserman (2015). The network input image is a 224×224 pixel random crop from augmented images, with per-pixel mean subtracted.
- Language Modeling for WikiText-2 (Merity et al., 2017). WikiText-2 is sourced from curated Wikipedia articles. It is frequently used for machine translation and language modeling, and features a vocabulary of over 30,000 words. Compared to the preprocessed version of Penn Treebank (PTB), WikiText-2 is over 2 times larger.

11.1.2 Models and Model Initialization

We use ResNet-20 (He et al., 2016a) with CIFAR-10 as a base configuration to understand the properties of (post-)local SGD. We then empirically evaluate the large-batch training performance of post-local SGD, for ResNet-20, DensetNet-40-12 (Huang et al., 2017b) and WideResNet-28-10 (Zagoruyko and Komodakis, 2016) on CIFAR-10/100, and for LSTM on WikiText-2 (Merity

Table 11.1 – Scaling ratio for various models.

Model	Communication # parameters	Computation # flops per image	Computation/Communication scaling ratio
ResNet-20 (CIFAR-10)	0.27 million	0.041 billion	151.85
ResNet-20 (CIFAR-100)	0.27 million	0.041 billion	151.85
ResNet-50 (ImageNet-1k)	25.00 million	7.7 billion	308.00
DenseNet-40-12 (CIFAR-10)	1.06 million	0.28 billion	264.15
DenseNet-40-12 (CIFAR-100)	1.10 million	0.28 billion	254.55
WideResNet-28-10 (CIFAR-10)	36.48 million	5.24 billion	143.64
WideResNet-28-10 (CIFAR-100)	36.54 million	5.24 billion	143.40

et al., 2017). Finally, we train ResNet-50 (He et al., 2016a) on ImageNet to investigate the accuracy and scalability of (post-)local SGD training.

For the weight initialization we follow Goyal et al. (2017), where we adopt the initialization introduced by He et al. (2015) for convolutional layers and initialize fully-connected layers by a zero-mean Gaussian distribution with the standard deviation of 0.01.

Table 11.1 demonstrates the scaling ratio of our mainly used Neural Network architectures. The scaling ratio (You et al., 2017c) identifies the ratio between computation and communication, wherein DNN models, the computation is proportional to the number of floating point operations required for processing an input while the communication is proportional to model size (or the number of parameters). Our local SGD training scheme will show more advantages over models with small “computation and communication scaling ratio”.

11.1.3 Large Batch Learning Schemes

The work of Goyal et al. (2017) proposes common configurations to tackle large-batch training for the ImageNet dataset. We specifically refer to their crucial techniques w.r.t. learning rate as “large batch learning schemes” in our main text. For a precise definition, this is formalized by the following two configurations:

- **Scaling the learning rate:** When the mini-batch size is multiplied by k , multiply the learning rate by k .
- **Learning rate gradual warm-up:** We gradually ramp up the learning rate from a small to a large value. In (our) experiments, with a large mini-batch of size kn , we start from a learning rate of η and increment it by a constant amount at each iteration such that it reaches $\hat{\eta} = k\eta$ after 5 epochs. More precisely, the incremental step size for each iteration is calculated from $\frac{\hat{\eta} - \eta}{5N/(kn)}$, where N is the number of total training samples, k is the number of computing units and n is the local mini-batch size.

11.1.4 Hyperparameter Choices and Training Procedure

CIFAR-10/CIFAR-100

The experiments follow the common mini-batch SGD training scheme for CIFAR (He et al., 2016a,b; Huang et al., 2017b) and all competing methods access the same total amount of data samples regardless of the number of local steps. The training procedure is terminated when the distributed algorithms have accessed the same number of samples as a standalone worker would access. For example, ResNet-20, DensetNet-40-12 and WideResNet-28-10 would access 300, 300 and 250 epochs respectively. The data is partitioned among the GPUs and reshuffled globally every epoch. The local mini-batches are then sampled among the local data available on each GPU, and its size is fixed to $B_{\text{loc}} = 128$.

The learning rate scheme follows works (He et al., 2016a; Huang et al., 2017b), where we drop the initial learning rate by 10 when the model has accessed 50% and 75% of the total number of training samples. The initial learning rates of ResNet-20, DensetNet-40-12 and WideResNet-28-10 are fine-tuned on single GPU (which are 0.2, 0.2 and 0.1 respectively), and can be scaled by the global mini-batch size when using large-batch learning schemes.

In addition to this, we use a Nesterov momentum of 0.9 without dampening, which is applied independently to each local model. For all architectures, following He et al. (2016a), we do not apply weight decay on the learnable Batch Normalization (BN) coefficients. The weight decay of ResNet-20, DensetNet-40-12 and WideResNet-28-10 are $1e-4$, $1e-4$ and $5e-4$ respectively. For the BN for distributed training we again follow Goyal et al. (2017) and compute the BN statistics independently for each worker.

Unless mentioned specifically, local SGD uses the exact same optimization scheme as mini-batch SGD.

The procedure of fine-tuning. There is no optimal learning rate scaling rule for large-batch SGD across various mini-batch sizes, tasks and architectures, as revealed in the Figure 8 of Shallue et al. (2019). Our tuning procedure is built on the insight of their Figure 8, where we grid-search the optimal learning rate for each mini-batch size, starting from the linearly scaled learning rate. For example, compared to mini-batch size 128, mini-batch size 2048 with default large-batch learning schemes need to linearly scale the learning rate by the factor of 16. In order to find out its optimal learning rate, we will evaluate a linear-spaced grid of five factors (i.e. $\{15, 15.5, 16, 16.5, 17\}$). If the best performance was ever at one of the extremes of the grid, we would try new grid points so that the best performance was contained in the middle of the parameters. Please note that this unbounded grid search—even though initialized at the linearly scaled learning rate—does also cover other suggested scaling heuristics, for instance square root scaling, and finds the best scaling for each scenario.

Note that in our experiments of large-batch SGD, either with the default large-batch learning

Chapter 11. Appendix for Local SGD Variants

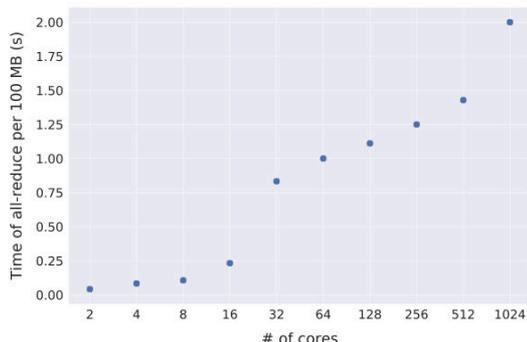


Figure 11.1 – The data transmission cost (in seconds) of an all-reduce operation for 100 MB, over a varied number of cores, using PyTorch’s built-in MPI all-reduce operation. Each evaluation is the average result of 100 data transmissions on a Kubernetes cluster. The network bandwidth is 10 Gbps, and we use 48 cores per physical machine.

Table 11.2 – The system performance of running forward and backward pass on a single GPU, for training ResNet20 on CIFAR-10. The “Ratio” indicates the $\frac{\text{Time of evaluating 4096 samples with specified mini-batch size}}{\text{Time of evaluating 4096 samples with mini-batch size 4096}}$. The “Time” is in seconds.

Mini-Batch Size	Titan XP		Tesla V100	
	Time per iteration (over 100 iterations)	Ratio	Time per iteration (over 100 iterations)	Ratio
32	0.058	1.490	0.028	9.028
64	0.100	1.284	0.030	4.836
128	0.175	1.124	0.034	2.741
256	0.323	1.037	0.043	1.733
512	0.737	1.183	0.073	1.471
1024	1.469	1.179	0.124	1.249
2048	2.698	1.083	0.212	1.068
4096	4.983	1	0.397	1

schemes, or tuning/using the optimal learning rate, we always warm-up the learning rate for the first 5 epochs.

ImageNet

ResNet-50 training is limited to 90 passes over the data in total, and the data is disjointly partitioned and is re-shuffled globally every epoch. All competing methods access the same total number of data samples (i.e. gradients) regardless of the number of local steps. We adopt the large-batch learning schemes as in [Goyal et al. \(2017\)](#) below. We linearly scale the learning rate based on $(\text{Number of GPUs} \times \frac{0.1}{256} \times B_{\text{glob}})$ where 0.1 and 256 is the base learning rate and mini-batch size respectively for standard single GPU training. The local mini-batch size is set to 128. For learning rate scaling, we perform gradual warmup for the first 5 epochs, and decay the scaled learning rate by the factor of 10 when local models have access 30, 60, 80 epochs of training samples respectively.

11.1.5 System Performance Evaluation

Figure 11.1 investigates the increased latency of transmitting data among CPU cores.

Table 11.2 evaluates the time of running forward and backward with various sizes of mini-batch, for training ResNet20 on CIFAR-10. We can witness that a larger mini-batch size we use, the better parallelism can a GPU have.

11.2 Local SGD Training

11.2.1 Formal Definition of the Local SGD Algorithm

The formal definition of local SGD algorithm can be found in Algorithm 7.

Algorithm 7 Local SGD.

Requires: the initial model $\mathbf{x}_{(0)}$; training data with labels \mathcal{S} ; mini-batch of size B_{loc} per local model; step size η , and momentum m (optional); number of synchronization steps T ; number of local steps H ; number of nodes K .

```

1: procedure WORKER- $k$ 
2:   for  $t \in \{1, \dots, T\}$  do
3:     for  $h := 1, \dots, H$  do
4:       sample a mini-batch from  $\mathcal{S}_{(t)+h-1}^k$ .
5:       compute the gradient  $\mathbf{g}_{(t)+h-1}^k := \frac{1}{B_{\text{loc}}} \sum_{i \in \mathcal{S}_{(t)+h-1}^k} \nabla f_i(\mathbf{x}_{(t)+h-1}^k)$ .
6:       update the local model to  $\mathbf{x}_{(t)+h}^k := \mathbf{x}_{(t)+h-1} - \eta_{(t)} \mathbf{g}_{(t)+h-1}^k$ .
7:       All-Reduce aggregation of the gradients from all nodes:  $\Delta_{(t)}^k := \mathbf{x}_{(t)}^k - \mathbf{x}_{(t)+H}^k$ .
8:       get new global model  $\mathbf{x}_{(t+1)}^k$  for all nodes:  $\mathbf{x}_{(t+1)}^k := \mathbf{x}_{(t)}^k - \eta_{(t)} \frac{1}{K} \sum_{i=1}^K \Delta_{(t)}^k$ 
9:   return  $\mathbf{x}_i^{(T)}$ 

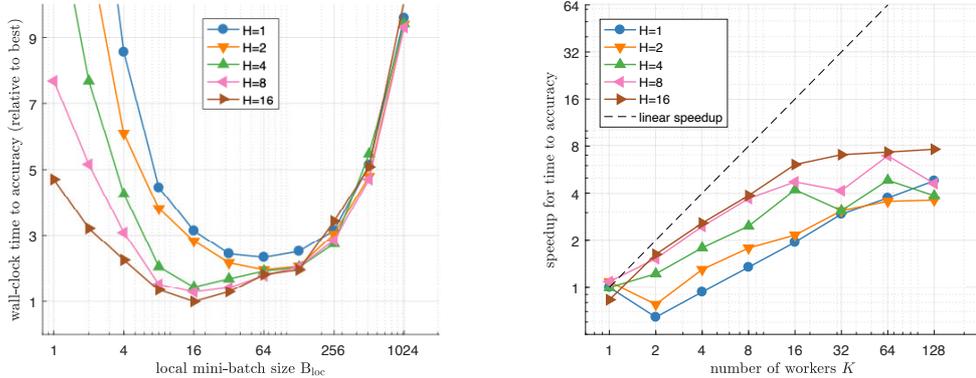
```

11.2.2 Numerical Illustration of Local SGD on a Convex Problem

In addition to our deep learning experiments, we first illustrate the convergence properties of local SGD on a small scale convex problem. For this, we consider logistic regression on the w8a dataset¹ ($d = 300, n = 49749$). We measure the number of iterations to reach the target accuracy $\epsilon = 0.005$. For each combination of H, B_{loc} and K we determine the best learning rate by extensive grid search (c.f. paragraph below for the detailed experimental setup). In order to mitigate extraneous effects on the measured results, we here measure time in discrete units, that is we count the number of stochastic gradient computations and communication rounds, and assume that communication of the weights is $25 \times$ more expensive than a gradient computation, for ease of illustration.

Figure 11.2(a) shows that various combinations of the parameters (B_{loc}, H) can impact the

¹www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html



(a) **Time** (relative to best method) to solve a regularized logistic regression problem to target accuracy $\epsilon = 0.005$ for $K = 16$ workers for $H \in 1, 2, 4, 8, 16$ and local mini-batch size B_{loc} . We simulate the network traffic under the assumption that communication is $25\times$ slower than a stochastic gradient computation. (b) **Speedup** over the number of workers K to solve a regularized logistic regression problem to target accuracy $\epsilon = 0.005$, for $B_{loc} = 16$ and $H \in 1, 2, 4, 8, 16$. We simulate the network traffic under the assumption that communication is $25\times$ slower than a stochastic gradient computation.

Figure 11.2 – Numerical illustration of local SGD on a convex problem.

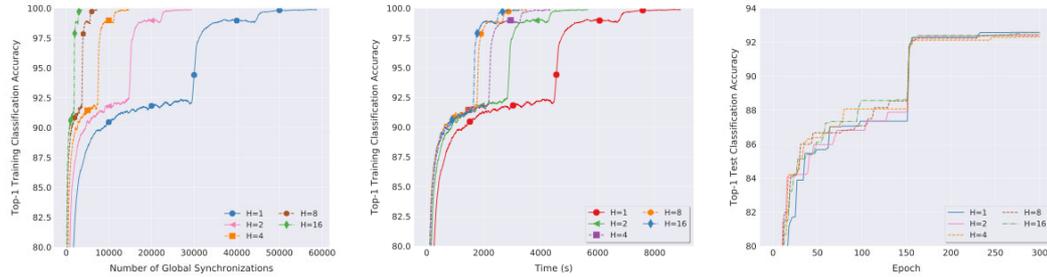
convergence time for $K = 16$. Here, local SGD with $(16, 16)$ converges more than $2\times$ faster than for $(64, 1)$ and $3\times$ faster than for $(256, 1)$.

Figure 11.2(b) depicts the speedup when increasing the number of workers K . Local SGD shows the best speedup for $H = 16$ on a small number of workers, while the advantage gradually diminishes for very large K .

Experimental Setup for Convex Experiments For the illustrative experiments here we study the convergence of local SGD on the logistic regression problem,

$$f(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n \log(1 + \exp(-b_i \mathbf{a}_i^\top \mathbf{x})) + \frac{\lambda}{2} \|\mathbf{x}\|^2,$$

where $\mathbf{a}_i \in \mathbb{R}^d$ and $b_i \in \{-1, +1\}$ are the data samples, and regularization parameter $\lambda = 1/n$. For each run, we initialize $\mathbf{x}_0 = \mathbf{0}_d$ and measure the number of stochastic gradient evaluations (and communication rounds) until the best of last iterate and weighted average of the iterates reaches the target accuracy $f(\mathbf{x}_t) - f^* \leq \epsilon := 0.005$, with $f^* := 0.126433176216545$. For each configuration (K, H, B_{loc}) , we report the best result found with any of the following two stepsizes: $\eta_t := \min(32, \frac{cn}{t+1})$ and $\eta_t = 32c$. Here c is a parameter that can take the values $c = 2^i$ for $i \in \mathbb{Z}$. For each stepsize we determine the best parameter c by a grid search, and consider parameter c optimal, if parameters $\{2^{-2}c, 2^{-1}c, 2c, 2^2c\}$ yield worse results (i.e. more iterations to reach the target accuracy).



(a) Training accuracy v.s. number (b) Training accuracy v.s. training (c) Test accuracy v.s. number of global synchronization rounds. time. epochs.

Figure 11.3 – Training **CIFAR-10** with **ResNet-20** via **local SGD** (2×1 -GPU). The local batch size B_{loc} is fixed to 128, and the number of local steps H is varied from 1 to 16. All the experiments are under the same training configurations.

11.2.3 More Results on Local SGD Training

Training CIFAR-10 via local SGD

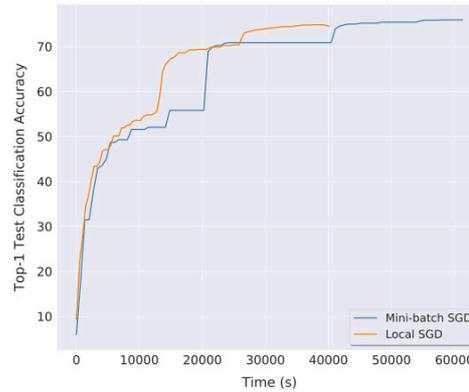
Better communication efficiency, with guaranteed test accuracy. Figure 11.3 shows that *local SGD is significantly more communication efficient while guaranteeing the same accuracy and enjoys faster convergence speed.* In Figure 11.3, the local models use a fixed local mini-batch size $B_{\text{loc}} = 128$ for all updates. All methods run for the same number of total gradient computations. Mini-batch SGD—the baseline method for comparison—is a special case of local SGD with $H = 1$, with full global model synchronization for each local update. We see that local SGD with $H > 1$, as illustrated in Figure 11.3(a), by design does H times less global model synchronizations, alleviating the communication bottleneck while accessing the same number of samples (see section 2.2). The impact of local SGD training upon the total training time is more significant for larger number of local steps H (i.e. Figure 11.3(b)), resulting in an at least $3\times$ speed-up when comparing mini-batch $H = 1$ to local SGD with $H = 16$. The final training accuracy remains stable across various H values, and there is no difference or negligible difference in test accuracy (Figure 11.3(c)).

Training ImageNet via Local SGD

Figure 11.4 shows the effectiveness of scaling local SGD to the challenging ImageNet dataset. We limit ResNet-50 training to 90 passes over the data in total, and use the standard training configurations as mentioned in Section 11.1.4.

Moreover, in our ImageNet experiment, the initial phase of local SGD training follows the theoretical assumption mentioned in section 2.3, and thus we gradually warm up the number of local steps from 1 to the desired value H during the first few epochs of the training. We found that exponentially increasing the number of local steps from 1 by the factor of 2 (until reaching the expected number of local steps) performs well. For example, our ImageNet training uses

Figure 11.4 – The performance of **local SGD** trained on **ImageNet-1k** with **ResNet-50**. We evaluate the model performance on test dataset after each complete accessing of the whole training samples. We apply the large-batch learning schemes (Goyal et al., 2017) to the ImageNet for these two methods. For local SGD, the number of local steps is set to $H = 8$.

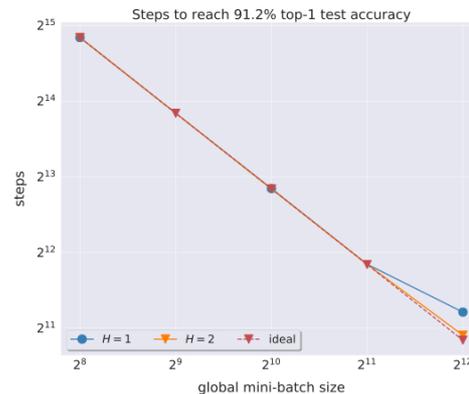


$H = 8$, so the number of local steps for the first three epochs is 1, 2, 4 respectively.

Local SGD Scales to Larger Batch Sizes than Mini-batch SGD

The empirical studies (Shallue et al., 2019; Golmant et al., 2018) reveal the regime of maximal data parallelism across various tasks and models, where the large-batch training would reach the limit and additional parallelism provides no benefit whatsoever.

Figure 11.5 – The relationship between steps to top-1 test accuracy and batch size, of training **ResNet-20** on **CIFAR-10**. The “step” is equivalent to the number of applying gradients. The global mini-batch size is increased by adding more workers K with fixed $B_{loc} = 128$. Results are averaged over three runs, each with fine-tuned learning rate.



On contrary to standard large-batch training, *local SGD scales to larger batch size and provides additional parallelism upon the limitation of current large batch training*. Figure 11.5 shows the example of training ResNet-20 on CIFAR-10 with $H = 2$, which trains and generalizes better in terms of update steps while with reduced communication cost.

11.2.4 Practical Improvement Potentials for Standard Local SGD Training

We investigate various aspects of the training to address the quality when scaling local SGD to the extreme case, e.g. hybrid momentum scheme, warming up the local SGD or fine-tuning the learning rate. In this section, we briefly present how these strategies are, and how they work in practice where we train ResNet-20 on CIFAR-10 on 16 GPUs.

Local SGD with Momentum

Momentum mini-batch SGD is widely used in place of vanilla SGD. The distributed mini-batch SGD with vanilla momentum on K training nodes follows

$$\mathbf{u}_{(t)} = m\mathbf{u}_{(t-1)} + \frac{1}{K} \sum_{k=1}^K \nabla^k_{(t)}, \quad \mathbf{x}_{(t+1)} = \mathbf{x}_{(t)} - \eta \mathbf{u}_{(t)},$$

where $\nabla^k_{(t)} = \frac{1}{|\mathcal{S}_{(t)}^k|} \sum_{i \in \mathcal{S}_{(t)}^k} \nabla f_i(\mathbf{x}_{(t)})$.

After H updates of mini-batch SGD, we have the following updated $\mathbf{x}_{(t+H)}$:

$$\mathbf{x}_{(t+H)} = \mathbf{x}_{(t)} - \eta \left(\sum_{\tau=1}^H m^\tau \mathbf{u}_{(t-1)} + \sum_{\tau=0}^{H-1} \frac{m^\tau}{K} \sum_{k=1}^K \nabla^k_{(t)} + \dots + \sum_{\tau=0}^0 \frac{m^\tau}{K} \sum_{k=1}^K \nabla^k_{(t+H-1)} \right).$$

Coming back to the setting of local SGD, we can apply momentum acceleration on each local model, or on a global level (Chen and Huo, 2016). In the remaining part of this section, we analyze the case of applying local momentum and global momentum. For ease of understanding, we assume the learning rate η is the same throughout the H update steps.

Local SGD with Local Momentum. When applying local momentum on the local SGD, i.e. using independent identical momentum acceleration for each local model and only globally aggregating the gradients at the time $(t) + H$, we have the following local update scheme

$$\mathbf{u}_{(t)}^k = m\mathbf{u}_{(t-1)}^k + \nabla^k_{(t)}, \quad \mathbf{x}_{(t+1)}^k = \mathbf{x}_{(t)}^k - \eta \mathbf{u}_{(t)}^k,$$

where $\nabla^k_{(t)} = \frac{1}{|\mathcal{S}_{(t)}^k|} \sum_{i \in \mathcal{S}_{(t)}^k} \nabla f_i(\mathbf{x}_{(t)})$. Consequently, after H local steps,

$$\mathbf{x}_{(t+H)}^k = \mathbf{x}_{(t)}^k - \eta \left(\sum_{\tau=1}^H m^\tau \mathbf{u}_{(t-1)}^k + \sum_{\tau=0}^{H-1} m^\tau \nabla^k_{(t)} + \dots + \sum_{\tau=0}^0 m^\tau \nabla^k_{(t+H-1)} \right).$$

Substituting the above equation into eq. (2.2), we have the update

$$\begin{aligned} \mathbf{x}_{(t+1)} &= \mathbf{x}_{(t)} - \frac{1}{K} \sum_{k=1}^K \eta \left(\sum_{\tau=1}^H m^\tau \mathbf{u}_{(t-1)}^k + \sum_{\tau=0}^{H-1} m^\tau \nabla^k_{(t)} + \dots + \sum_{\tau=0}^0 m^\tau \nabla^k_{(t+H-1)} \right) \\ &= w_{(t)} - \eta \left(\sum_{\tau=1}^H \frac{m^\tau}{K} \sum_{k=1}^K \mathbf{u}_{(t-1)}^k + \sum_{\tau=0}^{H-1} m^\tau \left(\frac{1}{K} \sum_{k=1}^K \nabla^k_{(t)} \right) + \dots + \sum_{\tau=0}^0 \frac{m^\tau}{K} \sum_{k=1}^K \nabla^k_{(t+H-1)} \right). \end{aligned}$$

Comparing the mini-batch SGD with local momentum local SGD after H update steps (H global update steps v.s. H local update steps and 1 global update step), we witness that the main difference of these two update schemes is the difference between $\sum_{\tau=1}^H m^\tau \mathbf{u}_{(t-1)}$ and

Chapter 11. Appendix for Local SGD Variants

$\sum_{\tau=1}^H \frac{m^\tau}{K} \sum_{k=1}^K \mathbf{u}^k_{(t-1)}$, where mini-batch SGD holds a global $\mathbf{u}_{(t-1)}$ while each local model of the local SGD has their own $\mathbf{u}^k_{(t-1)}$. We will soon see the difference between the global momentum of mini-batch SGD and the local momentum of local SGD.

Local SGD with Global Momentum For global momentum local SGD, i.e. a more general variant of block momentum (Chen and Huo, 2016), we would like to apply the momentum factor only to the accumulated/synchronized gradients:

$$\begin{aligned} \mathbf{u}_{(t)} &= m\mathbf{u}_{(t-1)} + \frac{1}{\eta} \sum_{k=1}^K \frac{1}{K} (\mathbf{x}_{(t)}^k - \mathbf{x}_{(t+H)}^k) = m\mathbf{u}_{(t-1)} + \frac{1}{\eta} \sum_{k=1}^K \frac{1}{K} \sum_{l=0}^{H-1} \eta \nabla_{(t+l)}^k, \\ \mathbf{x}_{(t+1)} &= \mathbf{x}_{(t)} - \eta \mathbf{u}_{(t)} = \mathbf{x}_{(t)} - \eta \left(m\mathbf{u}_{(t-1)} + \sum_{l=0}^{H-1} \sum_{k=1}^K \frac{1}{K} \nabla_{(t+l)}^k \right), \end{aligned}$$

where $\mathbf{x}_{(t+H)}^k = \mathbf{x}_{(t)}^k - \eta \sum_{l=0}^{H-1} \nabla_{(t+l)}^k = \mathbf{x}_{(t)} - \eta \sum_{l=0}^{H-1} \nabla_{(t+l)}^k$. Note that for local SGD, we consider summing up the gradients from each local update, i.e. the model difference before and after one global synchronization, and then apply the global momentum to the gradients over workers over previous local update steps.

Obviously, there exists a significant difference between mini-batch momentum SGD and global momentum local SGD, at least the term $\sum_{\tau=0}^H m^\tau$ is cancelled.

Local SGD with Hybrid Momentum. The following equation tries to combine local momentum with global momentum, showing a naive implementation.

First of all, based on the local momentum scheme, after H local update steps,

$$\mathbf{x}_{(t+H)}^k = \mathbf{x}_{(t)}^k - \eta \left(\sum_{\tau=1}^H m^\tau \mathbf{u}_{(t-1)}^k + \sum_{\tau=0}^{H-1} m^\tau \nabla_{(t)}^k + \dots + \sum_{\tau=0}^0 m^\tau \nabla_{(t+H-1)}^k \right).$$

Together with the result from local momentum with the global momentum, we have

$$\begin{aligned} \mathbf{u}_{(t)} &= m\mathbf{u}_{(t-1)} + \frac{1}{\eta} \sum_{k=1}^K \frac{1}{K} (\mathbf{x}_{(t)}^k - \mathbf{x}_{(t+H)}^k) \\ \mathbf{x}_{(t+1)} &= \mathbf{x}_{(t)} - \eta \mathbf{u}_{(t)} = \mathbf{x}_{(t)} - \eta \left[m\mathbf{u}_{(t-1)} + \frac{1}{\eta} \sum_{k=1}^K \frac{1}{K} (\mathbf{x}_{(t)}^k - \mathbf{x}_{(t+H)}^k) \right] \\ &= \mathbf{x}_{(t)} - \eta \left[m\mathbf{u}_{(t-1)} + \sum_{\tau=1}^H \frac{m^\tau}{K} \sum_{k=1}^K \mathbf{u}_{(t-1)}^k + \sum_{\tau=0}^{H-1} \frac{m^\tau}{K} \sum_{k=1}^K \nabla_{(t)}^k + \dots + \sum_{\tau=0}^0 \frac{m^\tau}{K} \sum_{k=1}^K \nabla_{(t+H-1)}^k \right], \end{aligned}$$

where $\mathbf{u}_{(t-1)}$ is the global momentum memory and $\mathbf{u}_{(t-1)}$ is the local momentum memory for each node k .

11.2. Local SGD Training

Table 11.3 – Evaluating local momentum and global momentum for **ResNet-20** on **CIFAR-10** data via local SGD training ($H = 1$ case) on 5×2 -GPU Kubernetes cluster. The local mini-batch size is 128 and base batch size is 64 (used for learning rate linear scale). Each local model will access to a disjoint data partition, using the standard learning rate scheme as [He et al. \(2016a\)](#).

local momentum	global momentum	test top-1
0.0	0.0	90.57
0.9	0.0	92.41
0.9	0.1	92.22
0.9	0.2	92.09
0.9	0.3	92.54
0.9	0.4	92.45
0.9	0.5	92.19
0.9	0.6	91.32
0.9	0.7	18.76
0.9	0.8	14.35
0.9	0.9	12.21
0.9	0.95	10.11

Local SGD with Momentum in Practice. In practice, it is possible to combine the local momentum with global momentum to further improve the model performance. For example, a toy example in Table 11.3 investigates the impact of various momentum schemes on CIFAR-10 trained with ResNet-20 on a 5×2 -GPU cluster, where some factors of global momentum could further slightly improve the final test accuracy.

However, the theoretical understanding of how local momentum and global momentum contribute to the optimization still remains unclear, which further increase the difficulty of tuning local SGD over H, K . An efficient way of using local and global momentum remains a future work and in this work, we only consider the local momentum.

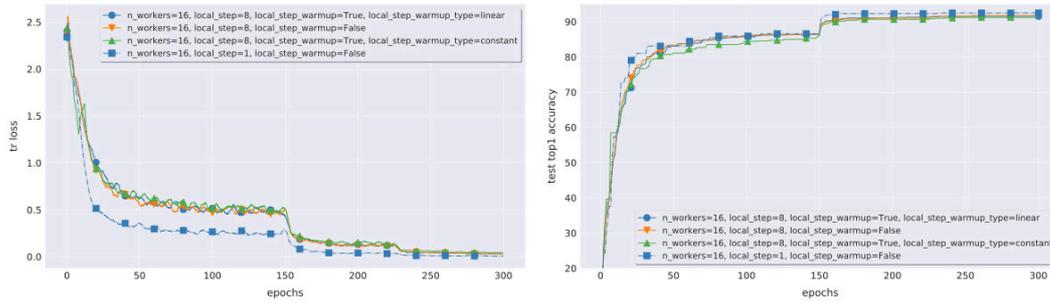
Warm-up of the Number of Local SGD Steps

We use the term “local step warm-up strategy”, to refer to a specific variant of post-local SGD. More precisely, instead of the two-phase regime which we presented here the used number of local steps H will be gradually increased from 1 to the expected number of local steps H . The warm-up strategies investigated here are “linear”, “exponential” and “constant”.

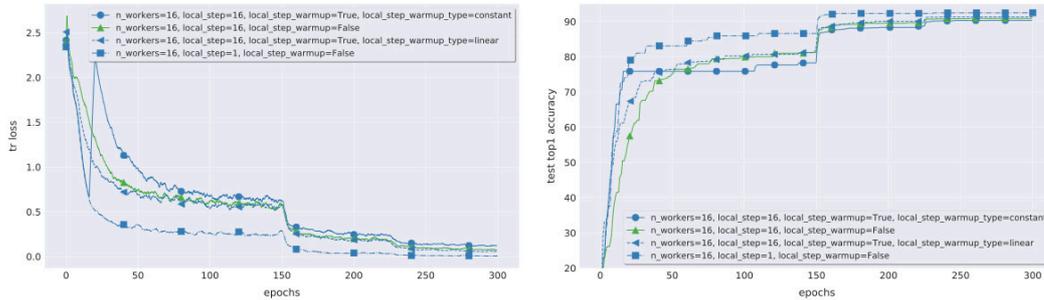
Please note that the implemented post-local SGD over the whole text only refers to the training scheme that uses frequent communication (i.e. $H = 1$) before the first learning rate decay and then reduces the communication frequency (i.e. $H > 1$) after the decay.

This section then investigates the trade-off between stochastic noise and the training stability. Also note that the Figure 2.3(a) in the main text has already presented one aspect of the trade-off. So the exploration below mainly focuses on the other aspects and tries to understand how will the scale of the added stochastic noise impact the training stability. Infact, even the model has been

Chapter 11. Appendix for Local SGD Variants



(a) Applying local step warm-up strategy to $H = 8$.



(b) Applying local step warm-up strategy to $H = 16$.

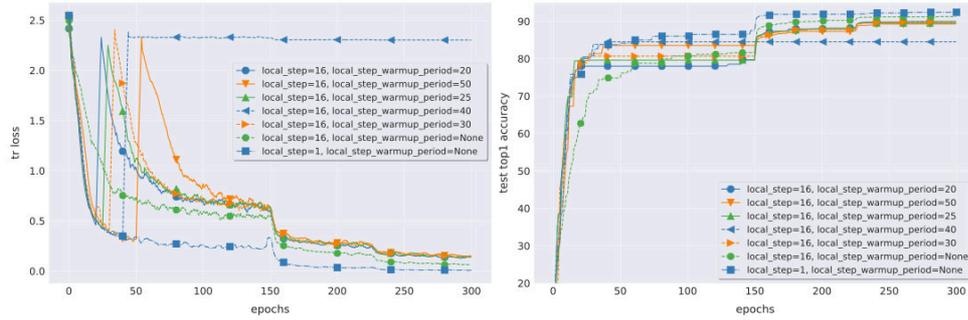
Figure 11.6 – Investigate how local step warm-up strategy impacts the performance of training **CIFAR-10** with **ResNet-20** via **local SGD** (8×2 -GPU). The local batch size B_{loc} is fixed to 128. The warmup strategies are “linear” and “constant”, and the warm-up period used here is equivalent to the number of local steps H .

stabilized to a region with good quality.

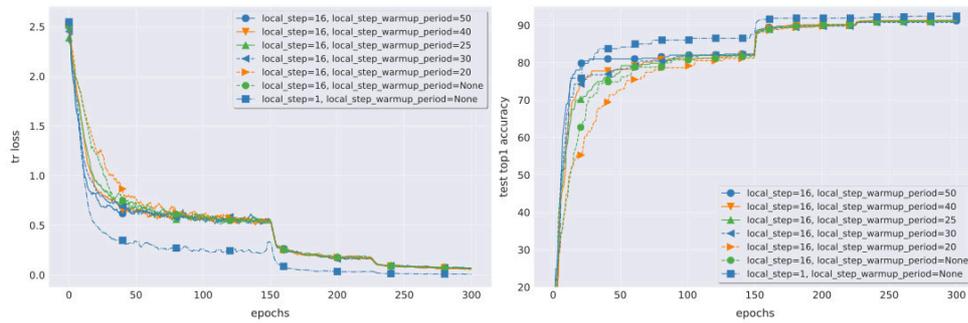
Figure 11.6 and Figure 11.7 investigate the potential improvement through using the local step warm-up strategy for the case of training ResNet-20 on CIFAR-10. Figure 11.6 evaluates the warm-up strategies of “linear” and “constant” for various H , while the evaluation of “exponential” warm-up strategy is omitted due to its showing similar performance as “linear” warm-up.

However, none of the investigated strategies show convincing performance. Figure 11.7 further studies how the period of warm-up impacts the training performance. We can witness that even if we increase the warm-up phase to 50 epochs where the training curve of mini-batch SGD becomes stabilized, the large noise introduced by the local SGD will soon degrade the status of training and lead to potential quality loss, as in Figure 11.7(a).

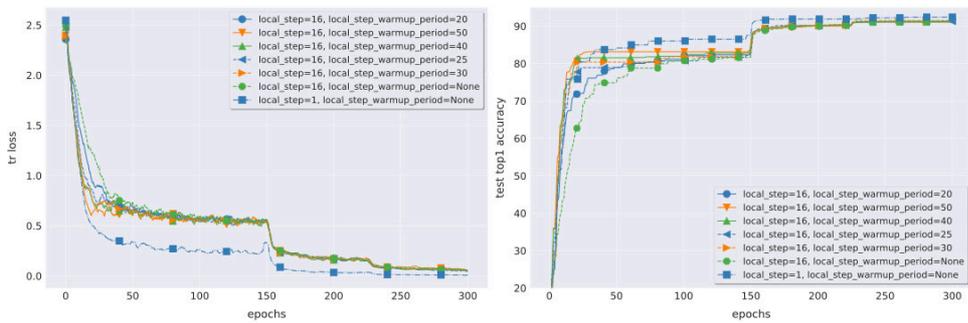
11.2. Local SGD Training



(a) Evaluating the impact of “constant” local step warm-up for various periods of warm-up phase.



(b) Evaluating the impact of “linear” local step warm-up for various periods of warm-up phase.



(c) Evaluating the impact of “exponential” local step warm-up for various periods of warm-up phase.

Figure 11.7 – Investigate how the warm-up period of the local step warm-up impacts the performance of training **CIFAR-10** with **ResNet-20** via **local SGD** (8×2 -GPU). The local batch size B_{loc} is fixed to 128, and the strategies to warm-up the number of local steps H are “linear”, “exponential” and “constant”.

11.3 Post-local SGD Training

11.3.1 The Algorithm of Post-local SGD

The formal definition of post-local SGD algorithm can be found in Algorithm 8.

Algorithm 8 Post-local SGD.

Requires: the initial model $\mathbf{x}_{(0)}$; training data with labels \mathcal{S} ; mini-batch of size B_{loc} per local model; step size η , and momentum m (optional); number of synchronization steps T , and the first learning rate decay is performed at T' ; number of eventual local steps H ; number of nodes K .

```

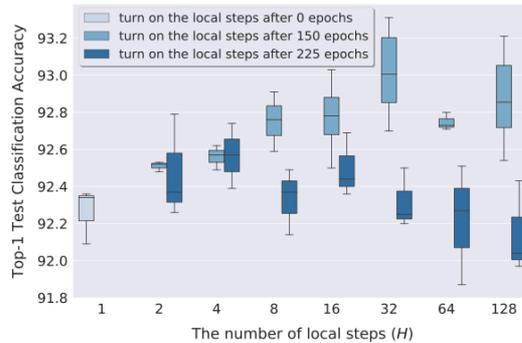
1: procedure WORKER- $k$ 
2:   for  $t \in \{1, \dots, T\}$  do
3:     if  $t < T'$  then
4:        $H_{(t)} = 1$ 
5:     else
6:        $H_{(t)} = H$ 
7:     for  $h := 1, \dots, H_{(t)}$  do
8:       sample a mini-batch from  $\mathcal{S}_{(t)+h-1}^k$ .
9:       compute the gradient  $\mathbf{g}_{(t)+h-1}^k := \frac{1}{B_{\text{loc}}} \sum_{i \in \mathcal{S}_{(t)+h-1}^k} \nabla f_i(\mathbf{x}_{(t)+h-1}^k)$ .
10:      update the local model to  $\mathbf{x}_{(t)+h}^k := \mathbf{x}_{(t)+h-1}^k - \eta_{(t)} \mathbf{g}_{(t)+h-1}^k$ .
11:      All-Reduce aggregation of the gradients from all nodes:  $\Delta_{(t)}^k := \mathbf{x}_{(t)}^k - \mathbf{x}_{(t)+H}^k$ .
12:      get new global model  $\mathbf{x}_{(t+1)}^k$  for all nodes:  $\mathbf{x}_{(t+1)}^k := \mathbf{x}_{(t)}^k - \eta_{(t)} \frac{1}{K} \sum_{i=1}^K \Delta_{(t)}^k$ 
13:   return  $\mathbf{x}_i^{(T)}$ 

```

11.3.2 The Effectiveness of Turning on Post-local SGD after the First Learning Rate Decay

In Figure 11.8, we study the sufficiency as well as the necessity of “injecting” more stochastic noise (i.e. using post-local SGD) into the optimization procedure after performing the first learning rate decay. Otherwise, the delayed noise injection (i.e. starting the post-local SGD only from the second learning rate decay) not only introduces more communication cost but also meets the increased risk of converging to sharper minima.

Figure 11.8 – The effectiveness and necessary of turning on the post-local SGD after the first learning rate decay. The example here trains **ResNet-20** on **CIFAR-10** on $K = 16$ GPUs with $B_{\text{loc}}K = 2048$.



11.3. Post-local SGD Training

Table 11.4 – The **Speedup** of mini-batch SGD and post-local SGD (only consider the phase of performing post-local SGD) over various CNN models and datasets. The speedup is evaluated by $\frac{T_a}{T_a^K}$, where T_a is the training time of the algorithm a (corresponding to the second phase) on 1 GPU and T_a^K corresponds to the training on K GPUs. We use 16 GPUs in total (with $B_{\text{loc}}=128$) on an 8×2 -GPU cluster with 10 Gbps network bandwidth. The experimental setup is the same as Table 2.2.

	CIFAR-10			CIFAR-100		
	H=1	H=16	H=32	H=1	H=16	H=32
ResNet-20	9.45	17.33	20.80	8.75	15.00	17.50
DenseNet-40-12	8.31	15.43	18.00	8.04	15.50	16.69
WideResNet-28-12	5.33	15.52	17.66	5.29	15.09	17.69

Table 11.5 – The **Speedup** of mini-batch SGD and post-local SGD, over various CNN models and datasets. The speedup is evaluated by $\frac{T_a}{T_a^K}$, where T_a is the training time of the algorithm a on 1 GPU and T_a^K corresponds to the training on K GPUs. We use 16 GPUs in total (with $B_{\text{loc}}=128$) on an 8×2 -GPU cluster with 10 Gbps network bandwidth. The experimental setup is the same as Table 2.2.

	CIFAR-10			CIFAR-100		
	H=1	H=16	H=32	H=1	H=16	H=32
ResNet-20	9.45	12.24	13.00	8.75	11.05	11.67
DenseNet-40-12	8.31	10.80	11.37	8.04	10.59	10.85
WideResNet-28-12	5.33	7.94	8.19	5.29	7.83	8.14

11.3.3 The Speedup of Post-local SGD Training on CIFAR

Table 2.2 and Table 11.4 evaluate the speedup of mini-batch SGD and post-local SGD, over various CNN models, datasets, and training phases.

11.3.4 Understanding the Generalization of Post-local SGD

The Sharpness Visualization Figure 11.9 visualizes the sharpness of the minima for training ResNet-20 on CIFAR-10. Ten random direction vectors are used for the filter normalization (Li et al., 2018), to ensure the correctness and consistence of the sharp visualization.

The spectrum of the Hessian for mini-batch SGD and post-local SGD Figure 11.10 evaluates the spectrum of the Hessian for the model trained from mini-batch SGD and post-local SGD with various numbers of local update H , which again demonstrates the fact that large-batch SGD tends to stop at points with high Hessian spectrum while post-local SGD could easily generalize to a low curvature solution and with better generalization.

1-d linear interpolation between models The 1-d linear interpolation was first used by Goodfellow et al. (2015) and then widely used to study the “sharpness” and “flatness” of minima

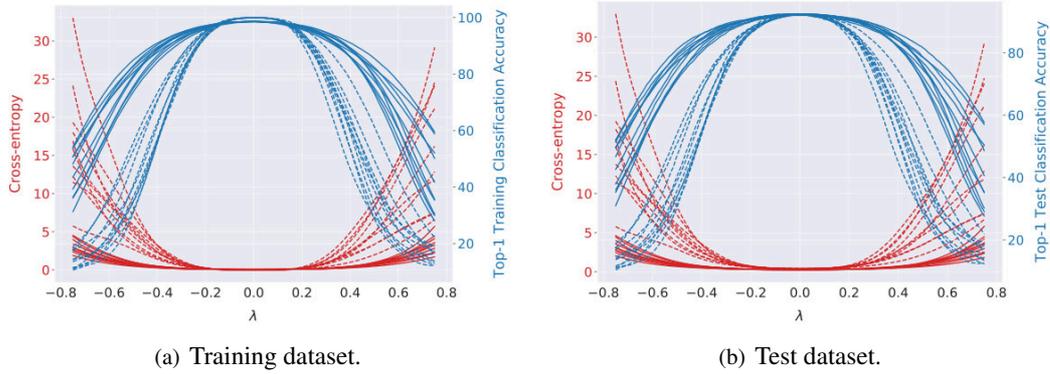
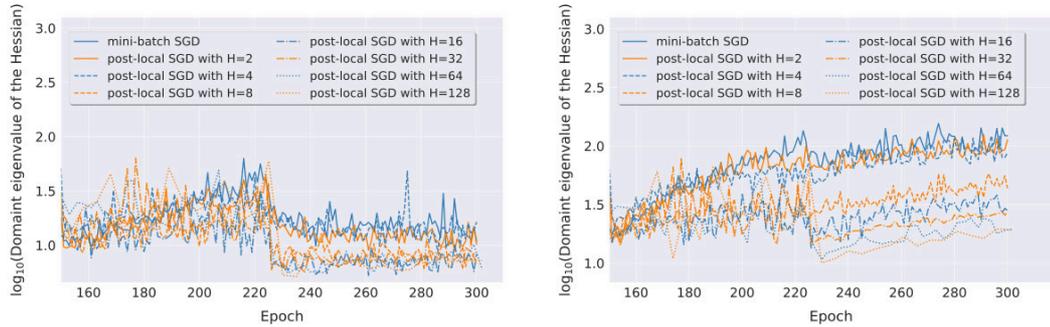


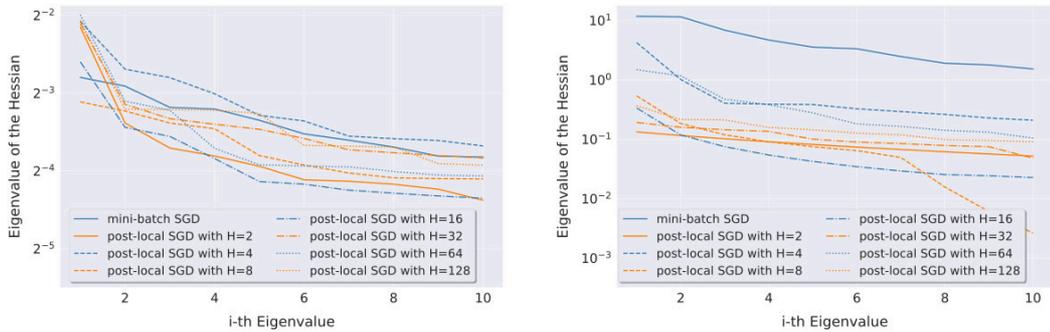
Figure 11.9 – Sharpness visualization of the minima for **ResNet-20** trained on **CIFAR-10**. The training is on top of $K = 16$ GPUs and the local batch size is fixed to $B_{\text{loc}} = 128$. The dashed lines are standard mini-batch SGD and the solid lines are post-local SGD with $H = 16$. The sharpness visualization of minima is performed via filter normalization (Li et al., 2018). The model is perturbed as $\mathbf{x} + \lambda \mathbf{d}$ by a shared random direction \mathbf{d} , and is evaluated by the whole dataset (training or test respectively). The top-1 test accuracy of mini-batch SGD is 92.25, while that of post-local SGD is 92.61. The sharpness of these two minima is consistent over 10 random directions.

in several works (Keskar et al., 2017; Dinh et al., 2017; Li et al., 2018). In Figure 11.11, the model trained by post-local SGD ($\mathbf{x}_{\text{post-local SGD}}$) can generalize to flatter minima than that of mini-batch SGD ($\mathbf{x}_{\text{mini-batch SGD}}$), either $\mathbf{x}_{\text{post-local SGD}}$ is trained from scratch, or resumed from the checkpoint of $\mathbf{x}_{\text{mini-batch SGD}}$ (i.e. the checkpoint is one-epoch ahead of the first learning rate decay so as to share the common weight structure with $\mathbf{x}_{\text{mini-batch SGD}}$).

11.3. Post-local SGD Training



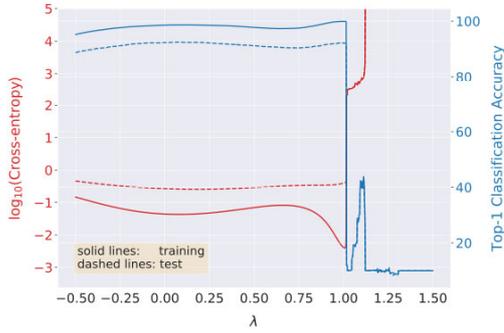
- (a) The dominant eigenvalue of the Hessian, which is evaluated on the training dataset per epoch. Only the phase related to the strategy of post-local SGD is visualized. Mini-batch SGD or post-local SGD with very much H (e.g. $H = 2, 4$) have noticeably larger dominant eigenvalue.
- (b) The dominant eigenvalue of the Hessian, which is evaluated on the test dataset per epoch. Only the phase related to the strategy of post-local SGD is visualized. Mini-batch SGD or post-local SGD with very much H (e.g. $H = 2, 4$) have noticeably larger dominant eigenvalue.



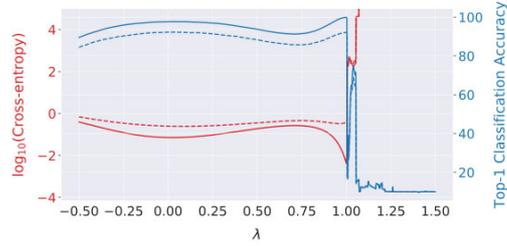
- (c) Top 10 eigenvalues of the Hessian, which is evaluated on the training dataset for the best model of each training scheme. The top eigenvalues of the Hessian of the mini-batch SGD clearly present a significant different pattern than the post-local SGD counterpart.
- (d) Top 10 eigenvalues of the Hessian, which is evaluated on the test dataset for the best model of each training scheme. The top eigenvalues of the Hessian of the mini-batch SGD clearly present a significant different pattern than the post-local SGD counterpart.

Figure 11.10 – The spectrum of the Hessian for **ResNet-20** trained on **CIFAR-10**. The training is on top of $K = 16$ GPUs with $KB_{\text{loc}} = 2048$. The spectrum is computed using power iteration (Martens and Sutskever, 2012; Yao et al., 2018) with the relative error of $1e-4$. The top-1 test accuracy of mini-batch SGD is 92.57, while that of post-local SGD ranges from 92.33 to 93.07. Current large-batch SGD tends to stop at points with considerably “larger” Hessian spectrum, while large-batch trained with post-local SGD generalizes to solution with low curvature and with better generalization.

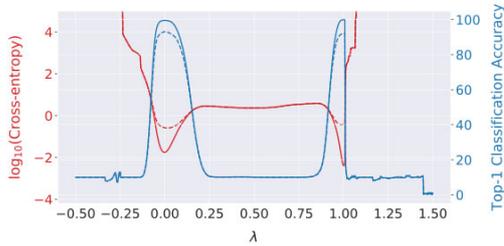
Chapter 11. Appendix for Local SGD Variants



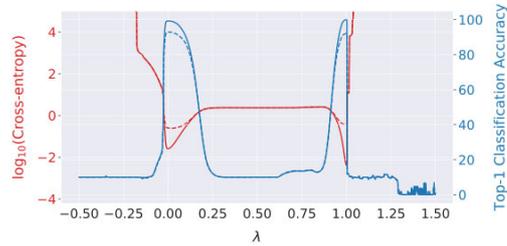
(a) $\mathbf{x}_{\text{post-local SGD}}$ is trained with $H = 16$, where the training resumes from the checkpoint of $\mathbf{x}_{\text{mini-batch SGD}}$ which is one-epoch ahead of the first learning rate decay.



(b) $\mathbf{x}_{\text{post-local SGD}}$ is trained with $H = 32$, where the training resumes from the checkpoint of $\mathbf{x}_{\text{mini-batch SGD}}$ which is one-epoch ahead of the first learning rate decay.



(c) $\mathbf{x}_{\text{post-local SGD}}$ is trained from scratch with $H = 16$.



(d) $\mathbf{x}_{\text{post-local SGD}}$ is trained from scratch with $H = 32$.

Figure 11.11 – 1-d linear interpolation between models $\mathbf{x}_{\text{post-local SGD}}$ and $\mathbf{x}_{\text{mini-batch SGD}}$, i.e. $\hat{\mathbf{x}} = \lambda \mathbf{x}_{\text{mini-batch SGD}} + (1 - \lambda) \mathbf{x}_{\text{post-local SGD}}$, for different minima of **ResNet-20** trained on **CIFAR-10**. The training is on top of $K = 16$ GPUs and the local batch size is fixed to $B_{\text{loc}} = 128$. The solid lines correspond to evaluate $\hat{\mathbf{x}}$ on the whole training dataset while the dashed lines are on the test dataset. The post-local SGD in Figure 11.11(a) and Figure 11.11(d) is trained from the checkpoint of $\mathbf{x}_{\text{mini-batch SGD}}$ before performing the first learning rate decay, while that of Figure 11.11(c) and Figure 11.11(d) is trained from scratch. The top-1 test accuracy of mini-batch SGD is 92.25, while that of post-local SGD in Figure 11.11(a), Figure 11.11(b), Figure 11.11(c) and Figure 11.11(d), are 92.61, 92.35, 93.13 and 93.04 respectively.

11.3. Post-local SGD Training

Table 11.6 – Top-1 **test accuracy** of training various CNN models via **post-local SGD** on $K=32$ GPUs with a large batch size ($B_{\text{loc}}K=4096$). The reported results are the average of three runs. We include the small and large batch baseline, where the models are trained by mini-batch SGD with mini-batch size 256 and 4096 respectively. The \star indicates the fine-tuned learning rate.

	CIFAR-100			
	small batch baseline \star	large batch baseline \star	post-local SGD (H=8)	post-local SGD (H=16)
ResNet-20	68.84 \pm 0.06	67.34 \pm 0.34	68.38 \pm 0.48	68.30 \pm 0.30
DenseNet-40-12	74.85 \pm 0.14	73.00 \pm 0.04	74.50 \pm 0.34	74.96 \pm 0.30
WideResNet-28-10	79.78 \pm 0.16	77.82 \pm 0.65	79.53 \pm 0.45	79.80 \pm 0.39

Table 11.7 – Top-1 **test accuracy** of large-batch SGD and post-local SGD, for training ResNet-20 on CIFAR-100 with $B_{\text{loc}}K=4096$. The reported results are the average of three runs. We include the small and large batch baseline, where the models are trained by mini-batch SGD with 256 and 4096 respectively. The \star indicates the fine-tuned learning rate. The learning rate will be decayed by 10 when the distributed algorithm has accesses 50% and 75% of the total number of training samples.

# of epochs	small batch baseline \star	large batch baseline \star	post-local SGD (H=8)	post-local SGD (H=16)
300	68.84 \pm 0.06	67.34 \pm 0.34	68.38 \pm 0.48	68.30 \pm 0.30
400	69.07 \pm 0.27	67.55 \pm 0.21	69.06 \pm 0.15	69.05 \pm 0.26
500	69.03 \pm 0.10	67.42 \pm 0.63	69.02 \pm 0.38	68.87 \pm 0.27

11.3.5 Post-local SGD Training on Diverse Tasks

Post-local SGD Training on CIFAR-100 for Global Mini-batch Size $KB_{\text{loc}}=4096$

Table 11.6 presents the severe quality loss (at around 2%) of the fine-tuned large-batch SGD for training three CNNs on CIFAR-100. Our post-local SGD with default hyperparameters (i.e. the hyperparameters from small mini-batch size and via large-batch training schemes) can perfectly close the generalization gap or even better the fine-tuned small mini-batch baselines.

We further justify the argument of works (Hoffer et al., 2017; Shallue et al., 2019) in Table 11.7, where we increase the number of training epochs and train it longer (from 300 to 400 and 500) for ResNet-20 on CIFAR-100. The results below illustrate that increasing the number of training epochs alleviates the optimization difficulty of large-batch training.

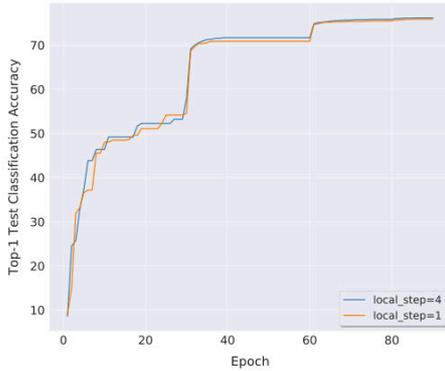
Post-local SGD Training on Language Modeling

We evaluate the effectiveness of post-local SGD for training the language modeling task on WikiText-2 through LSTM. We borrowed and adapted the general experimental setup of Merity et al. (2018), where we use a three-layer LSTM with hidden dimension of size 650. The loss will be averaged over all examples and timesteps. The BPTT length is set to 30. We fine-tune the value of gradient clipping (0.4) and the dropout (0.4) is only applied on the output of LSTM. The local mini-batch size B_{loc} is 64 and we train the model for 120 epochs. The learning rate is again decayed at the phase when the training algorithm has accessed 50% and 75% of the total training samples.

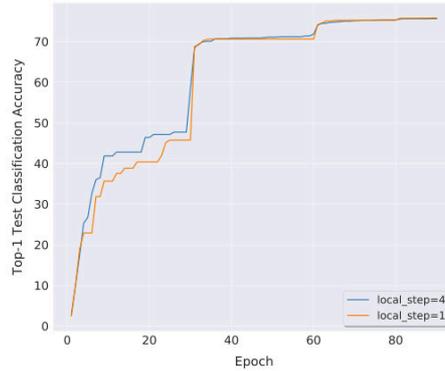
Chapter 11. Appendix for Local SGD Variants

Table 11.8 – The perplexity (lower is better) of language modeling task on WikiText-2. We use $K = 16$ and $KB = KB_{\text{loc}} = 1024$. The reported results are evaluated on the validation dataset (average of three runs). We fine-tune the learning rate for mini-batch SGD baselines.

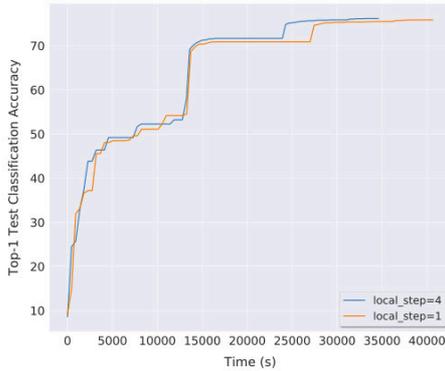
small batch baseline \star	large batch baseline \star	large-batch (H=8)	large-batch (H=16)
86.50 ± 0.35	86.90 ± 0.49	86.61 ± 0.30	86.85 ± 0.13



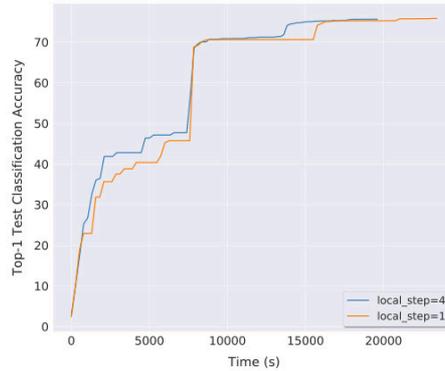
(a) The performance of post-local SGD training for **ImageNet-1k** with $KB_{\text{loc}} = 4096$, in terms of epoch-to-accuracy.



(b) The performance of post-local SGD training for **ImageNet-1k** with $KB_{\text{loc}} = 8192$, in terms of epoch-to-accuracy.



(c) The performance of post-local SGD training for **ImageNet-1k** with $KB_{\text{loc}} = 4096$, in terms of time-to-accuracy.



(d) The performance of post-local SGD training for **ImageNet-1k** with $KB_{\text{loc}} = 8192$, in terms of time-to-accuracy.

Figure 11.12 – The performance of **post-local SGD** training for **ImageNet-1k**. We evaluate the model performance on test dataset after each complete accessing of the whole training samples. Note that due to the resource limitation of the main experimental platform in the paper, these experiments are on top of a 8×4 -GPU (V100) Kubernetes cluster with 10 Gbps network bandwidth.

Table 11.8 demonstrates the effectiveness of post-local SGD for large-batch training on language modeling task (without extra fine-tuning except our baselines). Note that most of the existing work focuses on improving the large-batch training issue for computer vision tasks; and it is non-trivial to scale the training of LSTM for language modeling task due to the presence of various hyperparameters. Here we provide a proof of concept result in Table 11.8 to show that our

post-local SGD is able to improve upon standard large-batch baseline. The benefits can be further pronounced if we scale to larger batches (as the case of image classification e.g. in Table 11.6 and Table 11.7).

Post-local SGD Training on ImageNet

We evaluate the performance of post-local SGD on the challenging ImageNet training. Again we limit ResNet-50 training to 90 passes over the data in total, and use the standard training configurations as mentioned in Section 11.1.4. The post-local SGD begins when performing the first learning rate decay.

We can witness from Figure 11.12 that post-local SGD outperforms mini-batch SGD baseline for both of mini-batch size 4096 (76.18 and 75.87 respectively) and 8192 (75.65 and 75.64 respectively).

Post-local SGD v.s. Other Noise Injection Methods

The role of “noise” has been actively studied in SGD for non-convex deep learning, from optimization and generalization aspects. [Neelakantan et al. \(2015\)](#) propose to inject isotropic white noise for better optimization. [Zhu et al. \(2019\)](#); [Xing et al. \(2018\)](#) study the “structured” anisotropic noise and find the importance of anisotropic noise in SGD for escaping from minima (over isotropic noise) in terms of generalization. [Wen et al. \(2019\)](#) on top of these work and try to inject noise (sampled from the expensive empirical Fisher matrix) to large-batch SGD.

However, to our best knowledge, none of the prior work can provide a computation efficient way to inject noise to achieve as good generalization performance as small-batch SGD. Either it is practically unknown if injecting the isotropic noise ([Neelakantan et al., 2015](#)) can alleviate the issue of large-batch training, or it has been empirically justified ([Wen et al., 2019](#)) that injecting anisotropic noise from expensive (diagonal) empirical Fisher matrix fails to recover the same performance as the small mini-batch baselines. In this section, we only evaluate the impact of injecting isotropic noise ([Neelakantan et al., 2015](#)) for large-batch training, and omit the comparison to [Wen et al. \(2019\)](#).²

The idea of [Neelakantan et al. \(2015\)](#) considers to add time-dependent Gaussian noise to the gradient at every training step t : $\nabla f_i(\mathbf{x}_{(t)}) \leftarrow \nabla f_i(\mathbf{x}_{(t)}) + \mathcal{N}(0, \sigma_t^2)$, where σ_t^2 follows $\sigma_t^2 := \frac{\eta}{(1+t)^{\eta'}}$. We follow the general hyperparameter search scheme (as mentioned in Section 11.1.4) and fine-tune the η and η' in the range of $\{1e^{-6}, 5e^{-6}, 1e^{-5}, 5e^{-5}\}$ and $\{0.5, 1, 1.5, 2, 2.5, 3.0\}$ respectively.

Table 11.9 below compares the post-local SGD to [Neelakantan et al. \(2015\)](#), where the noise injection scheme in [Neelakantan et al. \(2015\)](#) cannot address the large-batch training issue and

² On the one hand, it is non-trivial to re-implement their method due to the unavailable code. On the other hand, it is known that our post-local SGD outperforms their expensive noise injection scheme ([Wen et al., 2019](#)), based on the comparison of their Table 1 and our Table 11.6-11.7.

Chapter 11. Appendix for Local SGD Variants

Table 11.9 – Top-1 **test accuracy** of adding isotropic noise for large-batch training. We revisit the case of ResNet-20 in Table 2.2 (2K for CIFAR-10) and Table 11.6 (4K for CIFAR-100), where adding isotropic noise as in Neelakantan et al. (2015) cannot address the large-batch training difficulty. The \star indicates a fine-tuned learning rate.

	CIFAR-10, $KB = 2K$	CIFAR-100, $KB = 4K$
Neelakantan et al. (2015) \star	92.46 ± 0.27	67.15 ± 0.26
mini-batch SGD \star	92.48 ± 0.17	67.38 ± 0.34
Post-local SGD	93.02 ± 0.24	68.30 ± 0.48

could even deteriorate the generalization performance. Note that we also tried to search the hyperparameters at around the values reported in Neelakantan et al. (2015) for our state-of-the-art CNNs, but it will directly result in the severe quality loss or even divergence.

Compared to another local SGD variant. Recently, Wang and Joshi (2019) proposed to decrease the number of local update steps H during training. However, their scheme is inherited from the convergence analysis for optimization (not the generalization for deep learning), and in principle opposite to our interpolation and proposed strategy (i.e. increasing the local update steps during the training).

Their evaluation (ResNet-50 on CIFAR-10 for $K = 4$ with $B_{\text{loc}} = 128$) also does not cover the difficult large batch training scenario (Scenario 2), e.g. $H = 16, K = 16, B_{\text{loc}} = 128$. For the same CIFAR-10 task and $K=4$ as in Wang and Joshi (2019), our smaller ResNet-20 with local SGD can simply reach a better accuracy with less communication³ (Figure 2.3).

Post-local SGD with Other Compression Schemes

In this subsection, we demonstrate that (post-)local SGD can be integrated with other compression techniques for better training efficiency (further reduced communication cost) and improved generalization performance (w.r.t. the gradient compression methods).

We use sign-based compression scheme (i.e. signSGD (Bernstein et al., 2018) and EF-signSGD (Karimireddy et al., 2019)) for the demonstration. The pseudo-code can be found in Algorithm 9 and Algorithm 10.

Post-local SGD with signSGD. We slightly adapt the original signSGD (Bernstein et al., 2018) to fit in the local SGD framework. The local model will be firstly updated by the sign of the local update directions (e.g. gradients, or gradients with weight decay and momentum acceleration), and then be synchronized to reach the global consensus model for the next local updates.

³ We directly compare to the reported values of Wang and Joshi (2019), as some missing descriptions and hyperparameters prevented us from reproducing the results ourselves. H in their paper forms a decreasing sequence starting with 10 while our local SGD uses constant $H=16$ during training.

Algorithm 9 (Post-)Local SGD with the compression scheme in signSGD.

Requires: the initial model $\mathbf{x}_{(0)}$; training data with labels \mathcal{S} ; mini-batch of size B_{loc} per local model; step size η , and momentum m (optional); number of synchronization steps T , and the first learning rate decay is performed at T' ; number of eventual local steps H ; number of nodes K .

```

1: procedure WORKER- $k$ 
2:   for  $t \in \{1, \dots, T\}$  do
3:     if  $t < T'$  then
4:        $H_{(t)} = 1$ 
5:     else
6:        $H_{(t)} = H$ 
7:     for  $h := 1, \dots, H_{(t)}$  do
8:       sample a mini-batch from  $\mathcal{S}_{(t)+h-1}^k$ .
9:       compute the gradient  $\mathbf{g}_{(t)+h-1}^k := \frac{1}{B_{\text{loc}}} \sum_{i \in \mathcal{S}_{(t)+h-1}^k} \nabla f_i(\mathbf{x}_{(t)+h-1}^k)$ .
10:      update the local model to  $\mathbf{x}_{(t)+h}^k := \mathbf{x}_{(t)+h-1}^k - \eta_{(t)} \text{sign}(\mathbf{g}_{(t)+h-1}^k)$ .
11:      get model difference  $\Delta_{(t)}^k := \mathbf{x}_{(t)}^k - \mathbf{x}_{(t)+H}^k$ .
12:      compress the model difference:  $\mathbf{s}_{(t)}^k = \text{sign}(\Delta_{(t)}^k)$  and  $\mathbf{p}_{(t)}^k = \frac{\|\Delta_{(t)}^k\|_1}{d}$ .
13:      get new global model  $\mathbf{x}_{(t+1)}^k$  for all nodes:  $\mathbf{x}_{(t+1)}^k := \mathbf{x}_{(t)}^k - \frac{1}{K} \sum_{i=1}^K \mathbf{s}_{(t)}^i \mathbf{p}_{(t)}^i$ .
14:   return  $\mathbf{x}_i^{(T)}$ 

```

Table 11.10 – Top-1 test accuracy of training ResNet-20 on CIFAR ($K B_{\text{loc}} = 2048$ and $K = 16$).

	CIFAR-10	CIFAR-100
signSGD (average over signs)	91.61 \pm 0.28	67.15 \pm 0.10
signSGD (majority vote over signs)	91.61 \pm 0.26	67.21 \pm 0.11

Note that we can almost recover the signSGD in Bernstein et al. (2018) from Algorithm 9 when $H' = 1$, except that we will average over the sign instead of using the majority vote in Bernstein et al. (2018). Table 11.10 illustrates the trivial generalization performance difference of these two schemes, where the post-local SGD in Algorithm 9 is able to significantly improve the generalization performance (as in Table 2.3) with further improved communication efficiency.

Post-local SGD with EF-signSGD. We extend the single-worker EF-signSGD algorithm of Karimireddy et al. (2019) (i.e. their Algorithm 1) to multiple-workers case by empirically investigating various algorithmic design choices. Our presented Algorithm 10 of EF-signSGD for multiple-workers can reach a similar performance as the mini-batch SGD counterpart (both are after the proper hyperparameters tuning and under the same experimental setup).

Training schemes, hyperparameter tuning procedure for signSGD, signSGD variant in Algorithm 9, and EF-signSGD in Algorithm 10. The training schemes of the signSGD (Bernstein et al., 2018) (i.e. the one using majority vote) and the signSGD variant in Algorithm 9 are

Chapter 11. Appendix for Local SGD Variants

Algorithm 10 (Post-)Local SGD with the compression scheme in EF-signSGD.

Requires: the initial model $\mathbf{x}_{(0)}$; training data with labels \mathcal{S} ; mini-batch of size B_{loc} per local model; step size η , and momentum m (optional); number of synchronization steps T , and the first learning rate decay is performed at T' ; number of eventual local steps H ; number of nodes K .

```

1: procedure WORKER- $k$ 
2:   for  $t \in \{1, \dots, T\}$  do
3:     if  $t < T'$  then
4:        $H_{(t)} = 1$ 
5:     else
6:        $H_{(t)} = H$ 
7:     for  $h := 1, \dots, H_{(t)}$  do
8:       sample a mini-batch from  $\mathcal{S}_{(t)+h-1}^k$ .
9:       compute the gradient  $\mathbf{g}_{(t)+h-1}^k := \frac{1}{B_{\text{loc}}} \sum_{i \in \mathcal{S}_{(t)+h-1}^k} \nabla f_i(\mathbf{x}_{(t)+h-1}^k)$ .
10:      update the local model to  $\mathbf{x}_{(t)+h}^k := \mathbf{x}_{(t)+h-1}^k - \eta_{(t)} \text{sign}(\mathbf{g}_{(t)+h-1}^k)$ .
11:      get model difference  $\Delta_{(t)}^k := \mathbf{x}_{(t)}^k - \mathbf{x}_{(t)+H}^k + \mathbf{e}_{(t)}$ .
12:      compress the model difference:  $\mathbf{s}_{(t)}^k = \text{sign}(\Delta_{(t)}^k)$  and  $\mathbf{p}_{(t)}^k = \frac{\|\Delta_{(t)}^k\|_1}{d}$ .
13:      update local memory:  $\mathbf{e}_{(t+1)}^k = \mathbf{e}_{(t)}^k - \mathbf{s}_{(t)}^k \mathbf{p}_{(t)}^k$ 
14:      get new global model  $\mathbf{x}_{(t+1)}^k$  for all nodes:  $\mathbf{x}_{(t+1)}^k := \mathbf{x}_{(t)}^k - \frac{1}{K} \sum_{i=1}^K \mathbf{s}_{(t)}^k \mathbf{p}_{(t)}^k$ .
15:   return  $\mathbf{x}_i^{(T)}$ 

```

slightly different from the the experimental setup mentioned in 11.1.4, where we only fine-tune the initial learning rate in signSGD and Algorithm 9 (when $H=1$). The optimal learning rate is searched from the grid $\{0.005, 0.10, 0.15, 0.20\}$ with the general fine-tuning principle mentioned in Section 11.1.4. No learning rate scaling up or warming up is required during the training, and the learning rate will be decayed by 10 when accessing 50% and 75% of the total training samples.

The training schemes of the distributed EF-signSGD in Algorithm 10 in general follow the experimental setup mentioned in Section 11.1.4. We grid search the optimal initial learning rate for Algorithm 10 (for the case of $H=1$), and gradually warm up the learning rate from a relatively small value (0.1) to this found initial learning rate during the first 5 epochs of the training.

Note that in our experiments, weight decay and Nesterov momentum are used for the local model updates, for both of Algorithm 9 and Algorithm 10. We empirically found these two techniques can significantly improve the training/test performance under a fixed training epoch budget.

11.4 Hierarchical Local SGD

The idea of local SGD can be leveraged to the more general setting of training on decentralized and heterogeneous systems, which is an increasingly important application area. Such systems have become common in the industry, e.g. with GPUs or other accelerators grouped hierarchi-

cally within machines, racks or even at the level of several data-centers. Hierarchical system architectures such as in Figure 11.13 motivate our hierarchical extension of local SGD. Moreover, end-user devices such as mobile phones form huge heterogeneous networks, where the benefits of efficient distributed and data-local training of machine learning models promises strong benefits in terms of data privacy.

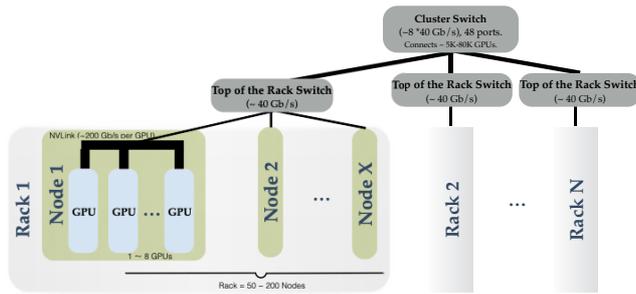


Figure 11.13 – Illustration of a hierarchical network architecture of a cluster in the data center. While GPUs within each node are linked with fast connections (e.g. NVLink), inter-server connections and inter-rack connection have much lower bandwidth and latency (via top-of-the-rack switches and cluster switches). The hierarchy can be extended several layers further and further. Finally, edge switches face the external network at even lower bandwidth.

11.4.1 The Illustration of Hierarchical Local SGD

Real world systems come with various communication bandwidths on several levels. In this scenario, we propose to employ local SGD on each level of the hierarchy, adapted to each corresponding computation vs communication trade-off. The resulting scheme, hierarchical local SGD, can offer significant benefits in system adaptivity and performance.

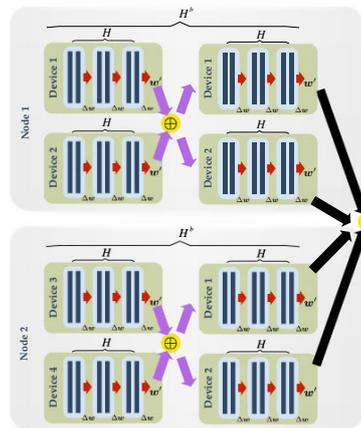


Figure 11.14 – An illustration of hierarchical local SGD, for $B_{loc} = 2$, using $H = 3$ inner local steps and $H^b = 2$ outer ‘block’ steps. Local parameter updates are depicted in red, whereas block and global synchronization is depicted in purple and black respectively.

As the guiding example, we consider compute clusters which typically allocate a large number of GPUs grouped over several machines, and refer to each group as a GPU-block. Hierarchical local SGD continuously updates the local models on each GPU for a number of H local update steps before a (fast) synchronization within a GPU-block. On the outer level, after H^b such block update steps, a (slower) global synchronization over all GPU-blocks is performed. Figure 11.14 and Algorithm 11 depict how the hierarchical local SGD works, and the complete procedure is

Chapter 11. Appendix for Local SGD Variants

Algorithm 11 (Post-)Local SGD with the compression scheme in EF-signSGD.

Requires: the initial model $\mathbf{x}_{(0)}$; training data with labels \mathcal{S} ; mini-batch of size B_{loc} per local model; step size η , and momentum m (optional); number of synchronization steps T ; number of local update steps H , and block update steps H^b ; number of nodes K in total; and nodes K' per GPU-block.

```

1: procedure WORKER- $k$ 
2:   for  $t \in \{1, \dots, T\}$  do
3:     for  $h := 1, \dots, H$  do
4:       for  $l := 1, \dots, H^b$  do
5:         sample a mini-batch from  $\mathcal{S}_{[(t)+l]+h-1}^k$ .
6:         compute the gradient  $\mathbf{g}_{[(t)+l]+h-1}^k := \frac{1}{B_{\text{loc}}} \sum_{i \in \mathcal{S}_{[(t)+l]+h-1}^k} \nabla f_i(\mathbf{x}_{[(t)+l]+h-1}^k)$ .
7:         update the local model to  $\mathbf{x}_{[(t)+l]+h}^k := \mathbf{x}_{[(t)+l]+h-1}^k - \eta_{[(t)]} \mathbf{g}_{[(t)+l]+h-1}^k$ .
8:         inner all-reduce aggregation of the gradients:  $\Delta_{[(t)+l]}^k := \mathbf{x}_{[(t)+l]}^k - \mathbf{x}_{[(t)+l]+H}^k$ .
9:         get new block model  $\mathbf{x}_{[(t)+l+1]}^k$ :  $\mathbf{x}_{[(t)+l+1]}^k := \mathbf{x}_{[(t)+l]}^k - \eta_{[(t)]} \frac{1}{K'} \sum_{k=1}^{K'} \Delta_{[(t)+l]}^k$ .
10:        outer all-reduce aggregation of the gradients:  $\Delta_{[(t)]}^k := \mathbf{w}_{[(t)]}^k - \mathbf{w}_{[(t)+H^b]}^k$ .
11:        get new global model  $\mathbf{x}_{[(t+1)]}^k$ :  $\mathbf{x}_{[(t+1)]}^k := \mathbf{x}_{[(t)]}^k - \eta_{[(t)]} \frac{1}{K} \sum_{i=1}^K \Delta_{[(t)]}^k$ .
12:   return  $\mathbf{x}_i^{(T)}$ 

```

formalized below:

$$\begin{aligned}
\mathbf{x}_{[(t)+l]+H}^k &:= \mathbf{x}_{[(t)+l]}^k - \sum_{h=1}^H \frac{\eta_{[(t)]}}{B_{\text{loc}}} \cdot \sum_{i \in \mathcal{S}_{[(t)+l]+h-1}^k} \nabla f_i(\mathbf{x}_{[(t)+l]+h-1}^k) \\
\mathbf{x}_{[(t)+l+1]}^k &:= \mathbf{x}_{[(t)+l]}^k - \frac{1}{K_i} \sum_{k=1}^{K_i} (\mathbf{x}_{[(t)+l]}^k - \mathbf{x}_{[(t)+l]+H}^k) \\
\mathbf{x}_{[(t+1)]}^k &:= \mathbf{x}_{[(t)]}^k - \frac{1}{K} \sum_{k=1}^K (\mathbf{x}_{[(t)]}^k - \mathbf{x}_{[(t)+H^b]}^k), \tag{11.1}
\end{aligned}$$

where $\mathbf{x}_{[(t)+l]+H}^k$ indicates the model after l block update steps and H local update steps, and K_i is the number of GPUs on the GPU-block i . The definition of $\eta_{[(t)]}$ and $\mathcal{S}_{[(t)+l]+h-1}^k$ follows a similar scheme.

As the number of devices grows to the thousands (Goyal et al., 2017; You et al., 2017c), the difference between ‘within’ and ‘between’ block communication efficiency becomes more drastic. Thus, the performance benefits of our adaptive scheme compared to flat & large mini-batch SGD will be even more pronounced.

11.4.2 Hierarchical Local SGD Training

Now we move to our proposed training scheme for distributed heterogeneous systems. In our experimental setup, we try to mimic the real world setting where several compute devices such as GPUs are grouped over various servers, and where network bandwidth (e.g. Ethernet) limits the communication of updates of large models. The investigation of hierarchical local SGD again trains ResNet-20 on CIFAR-10 and follows the same training procedure as local SGD where we

11.4. Hierarchical Local SGD

Table 11.11 – Training **CIFAR-10** with **ResNet-20** via **local SGD** on a 8×2 -GPU cluster. The local batch size B_{loc} is fixed to 128 with $H^b = 1$, and we scale the number of local steps H from 1 to 1024. The reported **training times** are the average of three runs and all the experiments are under the same training configurations for the equivalent of 300 epochs, without specific tuning.

$H =$	1	2	4	8	16	32	64	128	256	512	1024
Training Time (minutes)	20.07	13.95	10.48	9.20	8.57	8.32	9.22	9.23	9.50	10.30	10.65

re-formulate below.

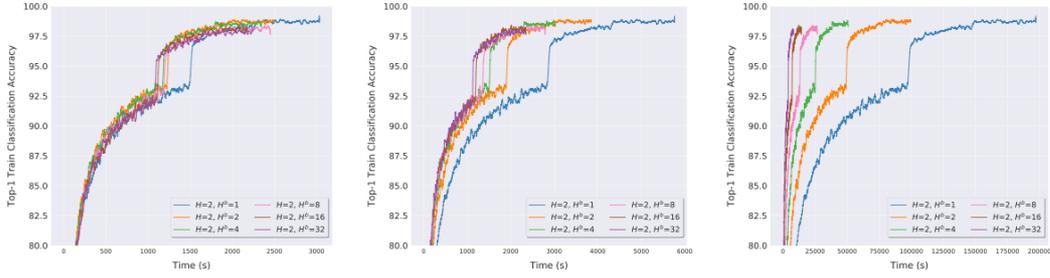
The experiments follow the common mini-batch SGD training scheme for CIFAR (He et al., 2016a,b) and all competing methods access the same total amount of data samples regardless of the number of local steps or block steps. More precisely, the training procedure is terminated when the distributed algorithms have accessed the same number of samples as a standalone worker would access in 300 epochs. The data is partitioned among the GPUs and reshuffled globally every epoch. The local mini-batches are then sampled among the local data available on each GPU. The learning rate scheme is the same as in He et al. (2016a), where the initial learning rate starts from 0.1 and is divided by 10 when the model has accessed 50% and 75% of the total number of training samples. In addition to this, the momentum parameter is set to 0.9 without dampening and applied independently to each local model.

The Performance of Hierarchical Local SGD.

Training time v.s. local number of steps. Table 11.11 shows the performance of local SGD in terms of training time. The communication traffic comes from the global synchronization over 8 nodes, each having 2 GPUs. We can witness that increasing the number of local steps over the “datacenter” scenario cannot infinitely improve the communication performance, or would even reduce the communication benefits brought by a large number of local steps. *Hierarchical local SGD with inner node synchronization reduces the difficulty of synchronizing over the complex heterogeneous environment, and hence enhances the overall system performance of the synchronization. The benefits are further pronounced when scaling up the cluster size.*

Hierarchical local SGD shows high tolerance to network delays. Even in our small-scale experiment of two servers and each with two GPUs, *hierarchical local SGD shows its ability to significantly reduce the communication cost by increasing the number of block step H^b (for a fixed H), with trivial performance degradation. Moreover, hierarchical local SGD with a sufficient number of block steps offers strong robustness to network delays.* For example, for fixed $H = 2$, by increasing the number of H^b , i.e. reducing the number of global synchronizations over all models, we obtain a significant gain in training time as in Figure 11.15(a). The impact of a network of slower communication is further studied in Figure 11.15(b), where the training is simulated in a realistic scenario and each global communication round comes with an additional delay of 1 second. Surprisingly, even for the global synchronization with straggling workers and

Chapter 11. Appendix for Local SGD Variants



(a) Training accuracy v.s. time. (b) Training accuracy v.s. time. (c) Training accuracy v.s. time. The number of local steps is $H = 2$, with 1 second delay for each with 50 seconds delay for each global synchronization.

Figure 11.15 – The performance of **hierarchical local SGD** trained on **CIFAR-10** with **ResNet-20** (2×2 -GPU). Each GPU block of the hierarchical local SGD has 2 GPUs, and we have 2 blocks in total. Each figure fixes the number of local steps but varies the number of block steps from 1 to 32. All the experiments are under the same training configurations without specific tuning.

Table 11.12 – The performance of training **CIFAR-10** with **ResNet-20** via **hierarchical local SGD** on a 16-GPU Kubernetes cluster. We simulate three types of cluster topology, namely 8 nodes with 2 GPUs/node, 4 nodes with 4 GPUs/node, and 2 nodes with 8 GPUs/node. The configuration of hierarchical local SGD satisfies $H \cdot H^b = 16$. All variants either synchronize within each node or over all GPUs, and the communication cost is estimated by only considering $H \cdot H^b = 16$ model updates during the training (the update could come from a different level of the synchronizations). The reported results are the average of three runs and all the experiments are under the same training configurations, training for the equivalent of 300 epochs, without specific tuning.

	$H = 1,$ $H^b = 16$	$H = 2,$ $H^b = 8$	$H = 4,$ $H^b = 4$	$H = 8,$ $H^b = 2$	$H = 16,$ $H^b = 1$
# of sync. over nodes	1	1	1	1	1
# of sync. within node	15	7	3	1	0
Test acc. on 8×2 -GPU	90.02 ± 0.28	90.25 ± 0.08	89.95 ± 0.19	91.41 ± 0.23	91.18 ± 0.02
Test acc. on 4×4 -GPU	91.65 ± 0.06	91.26 ± 0.17	91.46 ± 0.24	91.91 ± 0.16	
Test acc. on 2×8 -GPU	92.14 ± 0.10	92.05 ± 0.14	91.94 ± 0.09	91.56 ± 0.18	

severe 50 seconds delay per global communication round, Figure 11.15(c) demonstrates that a large number of block steps (e.g. $H^b = 16$) still manages to fully overcome the communication bottleneck with no/trivial performance damage.

Hierarchical local SGD offers improved scaling and better test accuracy. Table 11.12 compares the mini-batch SGD with hierarchical local SGD for fixed product $H \cdot H^b = 16$ under various network topologies, with the same training procedure. We can observe that for a heterogeneous system with a sufficient block size, hierarchical local SGD with a sufficient number of

block update steps can further improve the generalization performance of local SGD training. More precisely, when $H \cdot H^b$ is fixed, hierarchical local SGD with more frequent inner-node synchronizations ($H^b > 1$) outperforms local SGD ($H^b = 1$), while still maintaining the benefits of significantly reduced communication by the inner synchronizations within each node. In summary, as witnessed by Table 11.12, *hierarchical local SGD outperforms both local SGD and mini-batch SGD in terms of training speed as well as model performance*, especially for the training across nodes where inter-node connection is slow but intra-node communication is more efficient.

11.5 Discussion and Future Work

Data distribution patterns. In our experiments, the dataset is globally shuffled once per epoch and each local worker only accesses a disjoint part of the training data. Removing shuffling altogether, and instead keeping the disjoint data parts completely local during training might be envisioned for extremely large datasets which can not be shared, or also in a federated scenario where data locality is a must for privacy reasons. This scenario is not covered by the current theoretical understanding of local SGD, but will be interesting to investigate theoretically and practically.

Better learning rate scheduler for local SGD. We have shown in our experiments that local SGD delivers consistent and significant improvements over the state-of-the-art performance of mini-batch SGD. For ImageNet, we simply applied the same configuration of “large-batch learning schemes” by Goyal et al. (2017). However, this set of schemes was specifically developed and tuned for mini-batch SGD only, not for local SGD. For example, scaling the learning rate w.r.t. the global mini-batch size ignores the frequent local updates where each local model only accesses local mini-batches for most of the time. Therefore, it is expected that specifically deriving and tuning a learning rate scheduler for local SGD would lead to even more drastic improvements over mini-batch SGD, especially on larger tasks such as ImageNet.

Adaptive local SGD. As local SGD achieves better generalization than current mini-batch SGD approaches, an interesting question is if the number of local steps H could be chosen adaptively, i.e. change during the training phase. This could potentially eliminate or at least simplify complex learning rate schedules. Furthermore, recent works (Loshchilov and Hutter, 2017; Huang et al., 2017a) leverage cyclic learning rate schedules either improving the general performance of deep neural network training, or using an ensemble multiple neural networks at no additional training cost. Adaptive local SGD could potentially achieve similar goals with reduced training cost.

Hierarchical local SGD design. Hierarchical local SGD provides a simple but efficient training solution for devices over the complex heterogeneous system. However, its performance might be impacted by the cluster topology. For example, the topology of 8×2 -GPU in Table 11.12 fails to further improve the performance of local SGD by using more frequent inner node synchronizations. On contrary, sufficiently large size of the GPU block could easily benefit from the block update of hierarchical local SGD, for both communication efficiency and training quality. The design space of hierarchical local SGD for various cluster topologies should be further investigated, e.g. to investigate the two levels of model averaging frequency (within and between blocks) in terms of convergence, and the interplay of various local minima in the case of very large number of local steps. Another interesting line of work, explores heterogeneous systems by allowing for a varied number of local steps on various clusters, thus making up for slower machines.

12 Appendix for EXTRAP-SGD

12.1 Nonconvex Proof for Nesterov Momentum

One iterate of SGD with Nesterov momentum can be expressed as follows:

$$\mathbf{x}_{t+\frac{1}{2}} = \mathbf{x}_t + u\mathbf{v}_t, \quad \mathbf{v}_{t+1} = u\mathbf{v}_t - \frac{\eta}{BK} \sum_{k=1}^K \sum_{i \in \mathcal{I}_t^k} \nabla f_i(\mathbf{x}_{t+\frac{1}{2}}), \quad \mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{v}_{t+1}, \quad (12.1)$$

where $\mathbf{v}_0 = 0$. In the rest of Section 12.1, we use $\mathbf{g}_t := \frac{1}{KB} \sum_{k=1}^K \sum_{i \in \mathcal{I}_t^k} \nabla f_i(\mathbf{x}_t)$ to simplify notation. With this simplification the iterate (12.1) can be expressed as follows:

$$\mathbf{x}_{t+\frac{1}{2}} = \mathbf{x}_t + u\mathbf{v}_t, \quad \mathbf{v}_{t+1} = u\mathbf{v}_t - \eta\mathbf{g}_{t+\frac{1}{2}}, \quad \mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{v}_{t+1}, \quad (12.2)$$

We follow the idea of [Yu et al. \(2019a\)](#) and define an auxiliary sequence $\bar{\mathbf{y}}_t$ for (12.2):

$$\bar{\mathbf{y}}_t = \begin{cases} \mathbf{x}_1 = \mathbf{x}_0 & \text{if } t = 0 \\ \frac{1}{1-u}\mathbf{x}_{t+\frac{1}{2}} - \frac{u}{1-u}\mathbf{x}_{t-\frac{1}{2}} + \frac{\eta u}{1-u}\mathbf{g}_{t-\frac{1}{2}} & \text{if } t \geq 1 \end{cases}. \quad (12.3)$$

We have the following two auxiliary lemmas:

Lemma 12.1.1. *Consider the sequence $\{\bar{\mathbf{y}}\}$ in (12.3) and for all $t \geq 0$, we have $\bar{\mathbf{y}}_{t+1} - \bar{\mathbf{y}}_t = -\frac{\eta}{1-u}\mathbf{g}_{t+\frac{1}{2}}$.*

Chapter 12. Appendix for EXTRAP-SGD

Proof. For the case $t = 0$, we have

$$\begin{aligned}
 \bar{\mathbf{y}}_{t+1} - \bar{\mathbf{y}}_t &= \bar{\mathbf{y}}_1 - \bar{\mathbf{y}}_0 \\
 &= \frac{1}{1-u} \mathbf{x}_3 - \frac{u}{1-u} \mathbf{x}_2 + \frac{\eta u}{1-u} \mathbf{g}_2 - \mathbf{x}_1 = \frac{1}{1-u} (\mathbf{x}_3 - \mathbf{x}_2) + \frac{\eta u}{1-u} \mathbf{g}_2 \\
 &= \frac{1}{1-u} (u\mathbf{v}_1 - \eta \mathbf{g}_2) + \frac{\eta u}{1-u} \mathbf{g}_2 = \frac{1}{1-u} (-u\eta \mathbf{g}_2 - \eta \mathbf{g}_2) + \frac{\eta u}{1-u} \mathbf{g}_2 \\
 &= -\frac{\eta}{1-u} \mathbf{g}_2
 \end{aligned}$$

For the case $t \geq 1$, we have

$$\begin{aligned}
 \bar{\mathbf{y}}_{t+1} - \bar{\mathbf{y}}_t &= \frac{1}{1-u} (\mathbf{x}_{t+\frac{3}{2}} - \mathbf{x}_{t+\frac{1}{2}}) - \frac{u}{1-u} (\mathbf{x}_{t+\frac{1}{2}} - \mathbf{x}_{t-\frac{1}{2}}) + \frac{\eta u}{1-u} (\mathbf{g}_{t+\frac{1}{2}} - \mathbf{g}_{t-\frac{1}{2}}) \\
 &= \frac{1}{1-u} (u\mathbf{v}_{t+1} - \eta \mathbf{g}_{t+\frac{1}{2}}) - \frac{u}{1-u} (u\mathbf{v}_t - \eta \mathbf{g}_{t-\frac{1}{2}}) + \frac{\eta u}{1-u} (\mathbf{g}_{t+\frac{1}{2}} - \mathbf{g}_{t-\frac{1}{2}}) \\
 &= -\frac{\eta}{1-u} \mathbf{g}_{t+\frac{1}{2}} + \frac{u}{1-u} (\mathbf{v}_{t+1} - u\mathbf{v}_t + \eta \mathbf{g}_{t+\frac{1}{2}}) + \frac{\eta u}{1-u} (\mathbf{g}_{t-\frac{1}{2}} - \mathbf{g}_{t-\frac{1}{2}}) \\
 &= -\frac{\eta}{1-u} \mathbf{g}_{t+\frac{1}{2}}.
 \end{aligned}$$

□

Lemma 12.1.2. For all $t \geq 0$ and $\mathbf{x}_{t+\frac{1}{2}}$ defined in (12.2), we have

$$\sum_{t=0}^{T-1} \|\bar{\mathbf{y}}_t - \mathbf{x}_{t+\frac{1}{2}}\|^2 \leq \frac{u^4 \eta^2}{(1-u)^4} \sum_{t=0}^{T-1} \|\mathbf{g}_{t+\frac{1}{2}}\|^2. \quad (12.4)$$

Proof. Following the definition for $t = 0$, we have

$$\bar{\mathbf{y}}_0 - \mathbf{x}_1 = 0$$

For $t \geq 1$, we have

$$\begin{aligned}
 \bar{\mathbf{y}}_t - \mathbf{x}_{t+\frac{1}{2}} &= \frac{u}{1-u} (\mathbf{x}_{t+\frac{1}{2}} - \mathbf{x}_{t-\frac{1}{2}}) + \frac{\eta u}{1-u} \mathbf{g}_{t-\frac{1}{2}} \\
 &= \frac{u}{1-u} (u\mathbf{v}_t - \eta \mathbf{g}_{t-\frac{1}{2}}) + \frac{\eta u}{1-u} \mathbf{g}_{t-\frac{1}{2}} \\
 &= \frac{u^2}{1-u} \mathbf{v}_t.
 \end{aligned}$$

Letting $s_t := \sum_{i=0}^{t-1} u^{t-i-1} = \frac{1-u^t}{1-u}$ and fixing $T \geq 2$, we have

$$\sum_{t=0}^{T-1} \|\bar{\mathbf{y}}_t - \mathbf{x}_{t+\frac{1}{2}}\|^2 = \sum_{t=1}^{T-1} \|\bar{\mathbf{y}}_t - \mathbf{x}_{t+\frac{1}{2}}\|^2 = \frac{u^4}{(1-u)^2} \sum_{t=1}^{T-1} \|\mathbf{v}_t\|^2$$

12.1. Nonconvex Proof for Nesterov Momentum

Further we bound $\sum_{t=1}^{T-1} \|\mathbf{v}_t\|^2$ as follows:

$$\begin{aligned}
\sum_{t=1}^{T-1} \|\mathbf{v}_t\|^2 &= \eta^2 \sum_{t=1}^{T-1} \left\| \sum_{i=1}^t u^{t-i} \mathbf{g}_{i-\frac{1}{2}} \right\|^2 = \eta^2 \sum_{t=1}^{T-1} s_t^2 \left\| \sum_{i=0}^{t-1} \frac{u^{t-i-1}}{s_t} \mathbf{g}_{i+\frac{1}{2}} \right\|^2 \\
&\leq \eta^2 \sum_{t=1}^{T-1} s_t^2 \sum_{i=0}^{t-1} \frac{u^{t-i-1}}{s_t} \|\mathbf{g}_{i+\frac{1}{2}}\|^2 = \eta^2 \sum_{t=1}^{T-1} s_t \sum_{i=0}^{t-1} u^{t-i-1} \|\mathbf{g}_{i+\frac{1}{2}}\|^2 \\
&\leq \frac{\eta^2}{1-u} \sum_{t=1}^{T-1} \sum_{i=0}^{t-1} u^{t-i-1} \|\mathbf{g}_{i+\frac{1}{2}}\|^2 = \frac{\eta^2}{1-u} \sum_{i=0}^{T-2} \|\mathbf{g}_{i+\frac{1}{2}}\|^2 \sum_{t=i+1}^{T-1} u^{t-i-1} \\
&\leq \frac{\eta^2}{1-u} \sum_{i=0}^{T-1} \|\mathbf{g}_{i+\frac{1}{2}}\|^2 \sum_{t=0}^{\infty} u^t = \frac{\eta^2}{(1-u)^2} \sum_{i=0}^{T-1} \|\mathbf{g}_{i+\frac{1}{2}}\|^2,
\end{aligned}$$

which implies $\sum_{t=0}^{T-1} \|\tilde{\mathbf{y}}_t - \mathbf{x}_{t+\frac{1}{2}}\|^2 \leq \frac{u^4 \eta^2}{(1-u)^4} \sum_{t=0}^{T-1} \|\mathbf{g}_{t+\frac{1}{2}}\|^2$. \square

12.1.1 Main Proof of Theorem 3.5.2

Theorem 12.1.3 (Non-convex convergence of mini-batch SGD with Nesterov momentum). *Under Assumption 1, 2, and 3, for the update rule of mini-batch SGD with Nesterov momentum (12.2), we can show that, under the condition of $\eta \leq \frac{2(1-u)^2}{L(u^3+1)}$,*

$$\mathbb{E} \left[\frac{1}{T} \sum_{t=0}^{T-1} \left\| \nabla f(\mathbf{x}_{t+\frac{1}{2}}) \right\|^2 \right] \leq \frac{1}{1 - \frac{L\eta(u^3+1)}{2(1-u)^2}} \left(\frac{1}{T} \frac{\eta}{1-u} \mathbb{E} [f(\mathbf{x}_0) - f^*] + \frac{\eta L}{2(1-u)^2} \frac{\sigma^2}{BK} \right).$$

Proof. From the standard smoothness condition we have, $\forall t \geq 0$,

$$\begin{aligned}
\mathbb{E} [f(\tilde{\mathbf{y}}_{t+1}) - f(\tilde{\mathbf{y}}_t)] &\leq \mathbb{E} \left[\langle \nabla f(\tilde{\mathbf{y}}_t), \tilde{\mathbf{y}}_{t+1} - \tilde{\mathbf{y}}_t \rangle + \frac{L}{2} \|\tilde{\mathbf{y}}_{t+1} - \tilde{\mathbf{y}}_t\|^2 \right] \\
&\stackrel{(a)}{=} \mathbb{E} \left[-\frac{\eta}{1-u} \langle \nabla f(\tilde{\mathbf{y}}_t), \mathbf{g}_{t+\frac{1}{2}} \rangle + \frac{L}{2} \left\| \frac{\eta}{1-u} \mathbf{g}_{t+\frac{1}{2}} \right\|^2 \right] \\
&= \mathbb{E} \left[-\frac{\eta}{1-u} \langle \nabla f(\mathbf{x}_{t+\frac{1}{2}}) + \nabla f(\tilde{\mathbf{y}}_t) - \nabla f(\mathbf{x}_{t+\frac{1}{2}}), \mathbf{g}_{t+\frac{1}{2}} \rangle + \frac{L}{2} \left\| \frac{\eta}{1-u} \mathbf{g}_{t+\frac{1}{2}} \right\|^2 \right] \\
&= \mathbb{E} \left[-\frac{\eta}{1-u} \|\nabla f(\mathbf{x}_{t+\frac{1}{2}})\|^2 - \frac{\eta}{1-u} \langle \nabla f(\tilde{\mathbf{y}}_t) - \nabla f(\mathbf{x}_{t+\frac{1}{2}}), \nabla f(\mathbf{x}_{t+\frac{1}{2}}) \rangle + \frac{L}{2} \left\| \frac{\eta}{1-u} \mathbf{g}_{t+\frac{1}{2}} \right\|^2 \right],
\end{aligned}$$

where we use Lemma 12.1.1 for (a).

We can note that

$$\begin{aligned}
&-\frac{\eta}{1-u} \langle \nabla f(\tilde{\mathbf{y}}_t) - \nabla f(\mathbf{x}_{t+\frac{1}{2}}), \nabla f(\mathbf{x}_{t+\frac{1}{2}}) \rangle \\
&= \left\langle -\frac{\sqrt{1-u}}{\sqrt{Lu}^{3/2}} (\nabla f(\tilde{\mathbf{y}}_t) - \nabla f(\mathbf{x}_{t+\frac{1}{2}})), \frac{\eta\sqrt{Lu}^{3/2}}{(1-u)^{3/2}} \nabla f(\mathbf{x}_{t+\frac{1}{2}}) \right\rangle \\
&\stackrel{(b)}{\leq} \frac{1-u}{2Lu^3} \|\nabla f(\tilde{\mathbf{y}}_t) - \nabla f(\mathbf{x}_{t+\frac{1}{2}})\|^2 + \frac{\eta^2 Lu^3}{2(1-u)^3} \|\nabla f(\mathbf{x}_{t+\frac{1}{2}})\|^2,
\end{aligned}$$

where (b) follows the basic inequality $\langle \mathbf{a}, \mathbf{b} \rangle \leq \frac{1}{2} \|\mathbf{a}\|^2 + \frac{1}{2} \|\mathbf{b}\|^2$.

Thus, we have

$$\begin{aligned} & \mathbb{E}[f(\tilde{\mathbf{y}}_{t+1}) - f(\tilde{\mathbf{y}}_t)] \\ & \leq \mathbb{E} \left[-\frac{\eta}{1-u} \|\nabla f(\mathbf{x}_{t+\frac{1}{2}})\|^2 + \frac{1-u}{2Lu^3} \|\nabla f(\tilde{\mathbf{y}}_t) - \nabla f(\mathbf{x}_{t+\frac{1}{2}})\|^2 + \frac{\eta^2 Lu^3}{2(1-u)^3} \|\nabla f(\mathbf{x}_{t+\frac{1}{2}})\|^2 + \frac{\eta^2 L}{2(1-u)^2} \|\mathbf{g}_{t+\frac{1}{2}}\|^2 \right] \\ & \leq \mathbb{E} \left[-\frac{\eta}{1-u} \|\nabla f(\mathbf{x}_{t+\frac{1}{2}})\|^2 + \frac{(1-u)L}{2u^3} \|\tilde{\mathbf{y}}_t - \mathbf{x}_{t+\frac{1}{2}}\|^2 + \frac{\eta^2 Lu^3}{2(1-u)^3} \|\nabla f(\mathbf{x}_{t+\frac{1}{2}})\|^2 + \frac{\eta^2 L}{2(1-u)^2} \|\mathbf{g}_{t+\frac{1}{2}}\|^2 \right] \\ & = \mathbb{E} \left[\left(-\frac{\eta}{1-u} + \frac{\eta^2 Lu^3}{2(1-u)^3}\right) \|\nabla f(\mathbf{x}_{t+\frac{1}{2}})\|^2 + \frac{(1-u)L}{2u^3} \|\tilde{\mathbf{y}}_t - \mathbf{x}_{t+\frac{1}{2}}\|^2 + \frac{\eta^2 L}{2(1-u)^2} \|\mathbf{g}_{t+\frac{1}{2}}\|^2 \right]. \end{aligned}$$

Taking sum over t and averaging by $\frac{1}{T}$ yields:

$$\begin{aligned} & \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}[f(\tilde{\mathbf{y}}_{t+1}) - f(\tilde{\mathbf{y}}_t)] = \frac{1}{T} \mathbb{E}[f(\tilde{\mathbf{y}}_T) - f(\tilde{\mathbf{y}}_0)] \\ & \leq \mathbb{E} \left[\left(-\frac{\eta}{1-u} + \frac{L\eta^2 u^3}{2(1-u)^3}\right) \frac{1}{T} \sum_{t=0}^{T-1} \|\nabla f(\mathbf{x}_{t+\frac{1}{2}})\|^2 + \frac{(1-u)L}{2u^3} \frac{1}{T} \sum_{t=0}^{T-1} \|\tilde{\mathbf{y}}_t - \mathbf{x}_{t+\frac{1}{2}}\|^2 + \frac{\eta^2 L}{2(1-u)^2} \frac{1}{T} \sum_{t=0}^{T-1} \|\mathbf{g}_{t+\frac{1}{2}}\|^2 \right] \\ & \stackrel{(c)}{\leq} \left(-\frac{\eta}{1-u} + \frac{L\eta^2 u^3}{2(1-u)^3}\right) \frac{1}{T} \sum_{t=0}^{T-1} \|\nabla f(\mathbf{x}_{t+\frac{1}{2}})\|^2 + \mathbb{E} \left[\frac{(1-u)L\eta^2}{2u^3} \frac{u^4}{(1-u)^4} \frac{1}{T} \sum_{t=0}^{T-1} \|\mathbf{g}_{t+\frac{1}{2}}\|^2 \right] \\ & \quad + \mathbb{E} \left[\frac{\eta^2 L}{2(1-u)^2} \frac{1}{T} \sum_{t=0}^{T-1} \|\mathbf{g}_{t+\frac{1}{2}}\|^2 \right] \\ & \stackrel{(d)}{\leq} \left(-\frac{\eta}{1-u} + \frac{L\eta^2 u^3}{2(1-u)^3}\right) \frac{1}{T} \sum_{t=0}^{T-1} \|\nabla f(\mathbf{x}_{t+\frac{1}{2}})\|^2 + \left(\frac{\eta^2 L}{2(1-u)^2} + \frac{Lu\eta^2}{2(1-u)^3}\right) \left(\frac{1}{T} \sum_{t=0}^{T-1} \|\nabla f(\mathbf{x}_{t+\frac{1}{2}})\|^2 + \frac{\sigma^2}{BK}\right) \\ & = \left(-\frac{\eta}{1-u} + \frac{L\eta^2 u^3}{2(1-u)^3} + \frac{\eta^2 L}{2(1-u)^2} + \frac{Lu\eta^2}{2(1-u)^3}\right) \frac{1}{T} \sum_{t=0}^{T-1} \|\nabla f(\mathbf{x}_{t+\frac{1}{2}})\|^2 + \left(\frac{\eta^2 L}{2(1-u)^2} + \frac{Lu\eta^2}{2(1-u)^3}\right) \frac{\sigma^2}{BK}, \end{aligned}$$

where the inequality (c) comes from Lemma 12.1.2. For (d), by using $\mathbb{E}[\|X\|^2] = \text{var}[X] + \|\mathbb{E}[X]\|^2$ and the fact that $\text{var}[\sum_i X_i] = \sum_i \text{var}[X_i]$ if X_i 's are independent, we get

$$\begin{aligned} \mathbb{E}[\|\mathbf{g}_{t+\frac{1}{2}}\|^2] & = \mathbb{E} \left[\left\| \frac{1}{BK} \sum_{k=1}^K \sum_{i \in \mathcal{J}_t^k} \mathbf{g}_{t+\frac{1}{2}, i} \right\|^2 \right] \\ & = \frac{1}{B^2 K^2} \sum_{k=1}^K \sum_{i \in \mathcal{J}_t^k} \text{var}[\mathbf{g}_{t+\frac{1}{2}, i}] + \left\| \mathbb{E} \left[\frac{1}{BK} \sum_{k=1}^K \sum_{i \in \mathcal{J}_t^k} \mathbf{g}_{t+\frac{1}{2}, i} \right] \right\|_2^2 \leq \frac{1}{BK} \sigma^2 + \|\nabla f(\mathbf{x}_{t+\frac{1}{2}})\|^2. \end{aligned}$$

Note that the coefficient of the gradient norm of $-\frac{\eta}{1-u} + \frac{L\eta^2 u^3}{2(1-u)^3} + \frac{\eta^2 L}{2(1-u)^2} + \frac{Lu\eta^2}{2(1-u)^3}$ can be simplified as

$$\frac{\eta}{1-u} \left(-1 + \frac{L\eta u^3}{2(1-u)^2} + \frac{L\eta}{2(1-u)} + \frac{Lu\eta}{2(1-u)^2} \right) = \frac{\eta}{1-u} \left(-1 + \frac{L\eta(u^3 + 1)}{2(1-u)^2} \right).$$

By rearranging, we have

$$\begin{aligned}
 \frac{1}{T} \sum_{t=0}^{T-1} \left\| \nabla f(\mathbf{x}_{t+\frac{1}{2}}) \right\|^2 &\leq \frac{1}{T \frac{\eta}{1-u} \left(1 - \frac{L\eta(u^3+1)}{2(1-u)^2}\right)} \mathbb{E} [f(\tilde{\mathbf{y}}_0) - f(\tilde{\mathbf{y}}_T)] + \frac{\left(\frac{\eta^2 L}{2(1-u)^2} + \frac{Lu\eta^2}{2(1-u)^3}\right) \sigma^2}{\frac{\eta}{1-u} \left(1 - \frac{L\eta(u^3+1)}{2(1-u)^2}\right) BK} \\
 &\leq \frac{1}{T \frac{\eta}{1-u} \left(1 - \frac{L\eta(u^3+1)}{2(1-u)^2}\right)} \mathbb{E} [f(\mathbf{x}_0) - f^*] + \frac{\left(\frac{\eta L}{2(1-u)} + \frac{Lu\eta}{2(1-u)^2}\right) \sigma^2}{\left(1 - \frac{L\eta(u^3+1)}{2(1-u)^2}\right) BK} \\
 &\leq \frac{1}{1 - \frac{L\eta(u^3+1)}{2(1-u)^2}} \left(\frac{1}{T \frac{\eta}{1-u}} \mathbb{E} [f(\mathbf{x}_0) - f^*] + \frac{\eta L}{2(1-u)^2} \frac{\sigma^2}{BK} \right),
 \end{aligned}$$

where we have to make the overall coefficient of the RHS positive, i.e. $\eta \leq \frac{2(1-u)^2}{L(u^3+1)}$. \square

Lemma 12.1.4 (Lemma 13 of [Stich and Karimireddy \(2020\)](#)). *For every non-negative sequence $\{r_t\}_{t \geq 0}$ and any parameters $d \geq 0, c \geq 0, T \geq 0$, there exists a constant $\eta \leq \frac{1}{d}$, such that for constant stepsizes $\{\eta_t = \eta\}_{t \geq 0}$ it holds*

$$\Psi_T := \frac{1}{T+1} \sum_{t=0}^T \left(\frac{r_t}{\eta_t} - \frac{r_{t+1}}{\eta_t} + c\eta_t \right) \leq \frac{dr_0}{\eta(T+1)} + c\eta.$$

Corollary 12.1.5. *Consider the Theorem 3.5.2 and Lemma 12.1.4 with $\eta \leq \frac{2(1-u)^2}{L(u^3+1)}$, we can rewrite the convergence rate of Theorem 3.5.2 as*

$$\mathbb{E} \frac{1}{T} \sum_{t=0}^{T-1} \left\| \nabla f(\mathbf{x}_{t+\frac{1}{2}}) \right\|^2 = \mathcal{O} \left(\frac{Lr_0(u^3+1)}{T(1-u)} + \sqrt{\frac{2Lr_0\sigma^2}{KBT(1-u)}} \right).$$

Proof of Corollary 12.1.5. We first simplify the notations and constraints in Theorem 3.5.2 by considering $\Psi_T := \frac{1}{T\eta} r_0 + \frac{\eta L \sigma^2}{KB}$, where $r_0 := f(\mathbf{x}_0) - f^*$ and $\eta \leq \frac{1}{L}$. Following the techniques in the proof of the Lemma 13 in [Stich and Karimireddy \(2020\)](#), we consider two cases: (1) if $\frac{r_0 KB}{L\sigma^2 T} \leq \frac{1}{L^2}$, then we choose the stepsize $\eta = \sqrt{\frac{r_0 KB}{L\sigma^2 T}}$ and get $\Psi_T \leq 2\sqrt{\frac{Lr_0\sigma^2}{KBT}}$; (2) if $\frac{r_0 KB}{L\sigma^2 T} > \frac{1}{L^2}$, then we choose $\eta = \frac{1}{L}$ and get $\Psi_T \leq \frac{Lr_0}{T} + \frac{\sigma^2}{KB} \leq \frac{Lr_0}{T} + \frac{Lr_0}{T}$. These two bounds together show that $\Psi_T \leq 2\sqrt{\frac{Lr_0\sigma^2}{KBT}} + 2\frac{Lr_0}{T}$.

We then evaluate the exact case of Theorem 3.5.2 by considering $\Psi'_T := \frac{1}{T \frac{\eta}{1-u}} r_0 + \frac{\eta L}{2(1-u)^2} \frac{\sigma^2}{KB}$. Similarly, we have the potential η to minimize Ψ'_T , where $\eta = \sqrt{\frac{2r_0 KB(1-u)^3}{L\sigma^2 T}}$ and $\eta = \frac{2(1-u)^2}{L(u^3+1)}$. Thus, by considering two cases as in previous paragraph, we have $\Psi'_T \leq \frac{Lr_0(u^3+1)}{T(1-u)} + \sqrt{\frac{2Lr_0\sigma^2}{KBT(1-u)}}$. \square

12.2 Nonconvex Proof for EXTRAP-SGD

In the main text we propose to reuse past local gradients (with mini-batch size B) for extrapolation. For the convergence analysis illustrated in this section, we consider a more general extrapolation framework.

Chapter 12. Appendix for EXTRAP-SGD

Recall that a general form of EXTRAP-SGD can be defined as:

$$\mathbf{x}_{t+\frac{1}{4}} = \mathbf{x}_t + u\mathbf{v}_t, \mathbf{x}_{t+\frac{1}{2}}^k = \mathbf{x}_{t+\frac{1}{4}} - \frac{\hat{\eta}}{b} \sum_{i \in \hat{\mathcal{J}}_t^k} \boldsymbol{\xi}_{t,i}^k, \mathbf{v}_{t+1} = u\mathbf{v}_t - \frac{\eta}{BK} \sum_{k=1}^K \sum_{i \in \hat{\mathcal{J}}_t^k} \nabla f_i(\mathbf{x}_{t+\frac{1}{2}}^k), \mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{v}_{t+1}, \quad (12.5)$$

where $\mathbf{v}_0 = \mathbf{0}$ and $\boldsymbol{\xi}_{0,i}^k = \mathbf{0}$. The $\hat{\mathcal{J}}_t^k \subseteq \mathcal{J}_t^k$ indicates the possibility of using the samples from the subset of \mathcal{J}_t^k , and $b = |\hat{\mathcal{J}}_t^k|$. The $\boldsymbol{\xi}_{t,i}^k$ could be either fresh local gradient $\nabla f_i(\mathbf{x}_t^k)$, or old local gradient $\nabla f_i(\mathbf{x}_{t-\frac{1}{2}}^k)$. The later choice corresponds to Algorithm 1 demonstrated in the main paper.

Furthermore we could choose $\frac{1}{b} \sum_{i \in \hat{\mathcal{J}}_t^k} \boldsymbol{\xi}_{t,i}^k := \zeta_t^k$ as various types of i.i.d. noise e.g. Gaussian noise, uniform noise and stochastic noise as discussed in Section 3.4.2. For ease of exposition, we adopt the following notations:

$$\bar{\boldsymbol{\xi}}_t := \frac{1}{bK} \sum_{k=1}^K \sum_{i \in \hat{\mathcal{J}}_t^k} \boldsymbol{\xi}_{t,i}^k, \quad \mathbf{g}_{t,i}^k := \nabla f_i(\mathbf{x}_t^k), \quad \bar{\mathbf{g}}_t := \frac{1}{bK} \sum_{k=1}^K \sum_{i \in \hat{\mathcal{J}}_t^k} \mathbf{g}_{t,i}^k, \quad \bar{\mathbf{g}}_{t+\frac{1}{2}} := \frac{1}{BK} \sum_{k=1}^K \sum_{i \in \hat{\mathcal{J}}_t^k} \mathbf{g}_{t+\frac{1}{2},i}^k. \quad (12.6)$$

We follow the idea of using virtual sequence (Yu et al., 2019a; Stich and Karimireddy, 2020) and define two auxiliary sequences for our own setup (12.5):

$$\bar{\mathbf{x}}_{t+\frac{1}{2}} := \frac{1}{K} \sum_{k=1}^K \mathbf{x}_{t+\frac{1}{2}}^k, \quad (12.7)$$

and

$$\bar{\mathbf{y}}_t := \begin{cases} \bar{\mathbf{x}}_{\frac{1}{2}} = \mathbf{x}_0 & \text{if } t = 0 \\ \frac{1}{1-u} \bar{\mathbf{x}}_{t+\frac{1}{2}} - \frac{u}{1-u} \bar{\mathbf{x}}_{t-\frac{1}{2}} + \frac{\eta u}{1-u} \bar{\mathbf{g}}_{t-\frac{1}{2}} + \frac{\hat{\eta}}{1-u} (\bar{\boldsymbol{\xi}}_t - u \bar{\boldsymbol{\xi}}_{t-1}) & \text{if } t \geq 1 \end{cases}. \quad (12.8)$$

Following the definition of the virtual sequence in (12.7), the update scheme in (12.5) can be rewritten as

$$\begin{aligned} \bar{\mathbf{x}}_{t+\frac{1}{2}} &= \frac{1}{K} \sum_{k=1}^K \mathbf{x}_{t+\frac{1}{2}}^k = \frac{1}{K} \sum_{k=1}^K \left(\mathbf{x}_t + u\mathbf{v}_t - \frac{\hat{\eta}}{b} \sum_{i \in \hat{\mathcal{J}}_t^k} \boldsymbol{\xi}_{t,i}^k \right) = \mathbf{x}_t + u\mathbf{v}_t - \frac{\hat{\eta}}{bK} \sum_{k=1}^K \sum_{i \in \hat{\mathcal{J}}_t^k} \boldsymbol{\xi}_{t,i}^k = \mathbf{x}_t + u\mathbf{v}_t - \hat{\eta} \bar{\boldsymbol{\xi}}_t, \\ \mathbf{v}_{t+1} &= u\mathbf{v}_t - \frac{\eta}{BK} \sum_{k=1}^K \sum_{i \in \hat{\mathcal{J}}_t^k} \mathbf{g}_{t+\frac{1}{2},i}^k = u\mathbf{v}_t - \eta \bar{\mathbf{g}}_{t+\frac{1}{2}}, \\ \mathbf{x}_{t+1} &= \mathbf{x}_t + \mathbf{v}_{t+1}, \end{aligned} \quad (12.9)$$

We have the following three lemmas:

Lemma 12.2.1. Consider the sequence $\{\bar{\mathbf{y}}\}$ in (12.8) and for $t \geq 0$, we have

$$\bar{\mathbf{y}}_{t+1} - \bar{\mathbf{y}}_t = -\frac{\eta}{1-u} \bar{\mathbf{g}}_{t+\frac{1}{2}}.$$

Proof. For the case $t = 0$, we have

$$\begin{aligned} \bar{\mathbf{y}}_{t+1} - \bar{\mathbf{y}}_t &= \bar{\mathbf{y}}_1 - \bar{\mathbf{y}}_0 \\ &= \frac{1}{1-u} (\bar{\mathbf{x}}_{\frac{3}{2}} - \bar{\mathbf{x}}_{\frac{1}{2}}) + \frac{\eta u}{1-u} \bar{\mathbf{g}}_{\frac{1}{2}} + \frac{\hat{\eta}}{1-u} \bar{\boldsymbol{\xi}}_1 = \frac{1}{1-u} (u\mathbf{v}_1 - \eta \bar{\mathbf{g}}_{\frac{1}{2}} - \hat{\eta} \bar{\boldsymbol{\xi}}_1) + \frac{\eta u}{1-u} \bar{\mathbf{g}}_{\frac{1}{2}} + \frac{\hat{\eta}}{1-u} \bar{\boldsymbol{\xi}}_1 \\ &= \frac{1}{1-u} (-u\eta \bar{\mathbf{g}}_{\frac{1}{2}} - \eta \bar{\mathbf{g}}_{\frac{1}{2}} - \hat{\eta} \bar{\boldsymbol{\xi}}_1) + \frac{\eta u}{1-u} \bar{\mathbf{g}}_{\frac{1}{2}} + \frac{\hat{\eta}}{1-u} \bar{\boldsymbol{\xi}}_1 \\ &= -\frac{\eta}{1-u} \bar{\mathbf{g}}_{\frac{1}{2}}. \end{aligned}$$

For the case $t \geq 1$ we have

$$\begin{aligned} \bar{\mathbf{y}}_{t+1} - \bar{\mathbf{y}}_t &= \frac{1}{1-u} (\bar{\mathbf{x}}_{t+\frac{3}{2}} - \bar{\mathbf{x}}_{t+\frac{1}{2}}) - \frac{u}{1-u} (\bar{\mathbf{x}}_{t+\frac{1}{2}} - \bar{\mathbf{x}}_{t-\frac{1}{2}}) + \frac{\eta u}{1-u} (\bar{\mathbf{g}}_{t+\frac{1}{2}} - \bar{\mathbf{g}}_{t-\frac{1}{2}}) + \frac{\hat{\eta}}{1-u} (\bar{\boldsymbol{\xi}}_{t+1} - \bar{\boldsymbol{\xi}}_t - u\bar{\boldsymbol{\xi}}_t + u\bar{\boldsymbol{\xi}}_{t-1}) \\ &= \frac{1}{1-u} (u\mathbf{v}_{t+1} - \eta \bar{\mathbf{g}}_{t+\frac{1}{2}} + \hat{\eta} \bar{\boldsymbol{\xi}}_t - \hat{\eta} \bar{\boldsymbol{\xi}}_{t+1}) - \frac{u}{1-u} (u\mathbf{v}_t - \eta \bar{\mathbf{g}}_{t-\frac{1}{2}} + \hat{\eta} \bar{\boldsymbol{\xi}}_{t-1} - \hat{\eta} \bar{\boldsymbol{\xi}}_t) \\ &\quad + \frac{\eta u}{1-u} (\bar{\mathbf{g}}_{t+\frac{1}{2}} - \bar{\mathbf{g}}_{t-\frac{1}{2}}) + \frac{\hat{\eta}}{1-u} (\bar{\boldsymbol{\xi}}_{t+1} - \bar{\boldsymbol{\xi}}_t - u\bar{\boldsymbol{\xi}}_t + u\bar{\boldsymbol{\xi}}_{t-1}) \\ &= \frac{1}{1-u} (u\mathbf{v}_{t+1} - \eta \bar{\mathbf{g}}_{t+\frac{1}{2}}) + \frac{\hat{\eta}}{1-u} (\bar{\boldsymbol{\xi}}_t - \bar{\boldsymbol{\xi}}_{t+1}) - \frac{u}{1-u} (u\mathbf{v}_t - \eta \bar{\mathbf{g}}_{t-\frac{1}{2}}) - \frac{u\hat{\eta}}{1-u} (\bar{\boldsymbol{\xi}}_{t-1} - \bar{\boldsymbol{\xi}}_t) \\ &\quad + \frac{\eta u}{1-u} (\bar{\mathbf{g}}_{t+\frac{1}{2}} - \bar{\mathbf{g}}_{t-\frac{1}{2}}) + \frac{\hat{\eta}}{1-u} (\bar{\boldsymbol{\xi}}_{t+1} - \bar{\boldsymbol{\xi}}_t) - \frac{u\hat{\eta}}{1-u} (\bar{\boldsymbol{\xi}}_t - \bar{\boldsymbol{\xi}}_{t-1}) \\ &= \frac{1}{1-u} (u\mathbf{v}_{t+1} - \eta \bar{\mathbf{g}}_{t+\frac{1}{2}}) - \frac{u}{1-u} (u\mathbf{v}_t - \eta \bar{\mathbf{g}}_{t-\frac{1}{2}}) + \frac{\eta u}{1-u} (\bar{\mathbf{g}}_{t+\frac{1}{2}} - \bar{\mathbf{g}}_{t-\frac{1}{2}}) \\ &= -\frac{\eta}{1-u} \bar{\mathbf{g}}_{t+\frac{1}{2}} + \frac{u}{1-u} \mathbf{v}_{t+1} - \frac{u}{1-u} (u\mathbf{v}_t - \eta \bar{\mathbf{g}}_{t+\frac{1}{2}}) + \frac{\eta u}{1-u} (\bar{\mathbf{g}}_{t-\frac{1}{2}} - \bar{\mathbf{g}}_{t-\frac{1}{2}}) \\ &= -\frac{\eta}{1-u} \bar{\mathbf{g}}_{t+\frac{1}{2}}. \end{aligned}$$

□

Lemma 12.2.2. For $T \geq 2$ and $\bar{\mathbf{x}}_{t+\frac{1}{2}}$ defined in (12.7), we have

$$\mathbb{E} \left[\sum_{t=0}^{T-1} \left\| \bar{\mathbf{y}}_t - \bar{\mathbf{x}}_{t+\frac{1}{2}} \right\|^2 \right] \leq (1+\beta) \frac{u^4 \eta^2}{(1-u)^4} \mathbb{E} \left[\sum_{t=0}^{T-1} \left\| \bar{\mathbf{g}}_{t+\frac{1}{2}} \right\|^2 \right] + \left(1 + \frac{1}{\beta}\right) \hat{\eta}^2 \mathbb{E} \left[\sum_{t=0}^{T-1} \left\| \bar{\boldsymbol{\xi}}_t \right\|^2 \right], \forall \beta > 0.$$

If we reuse past local stochastic gradients with mini-batch size B , i.e. $\bar{\boldsymbol{\xi}}_t = \begin{cases} \mathbf{0} & \text{if } t = 0 \\ \bar{\mathbf{g}}_{t-\frac{1}{2}} & \text{if } t \geq 1 \end{cases}$, by

Chapter 12. Appendix for EXTRAP-SGD

choosing $\beta = 1$ and $\hat{\eta} \leq \frac{u^2}{(1-u)^2} \eta$, we obtain the simplified expression:

$$\mathbb{E} \left[\sum_{t=0}^{T-1} \left\| \bar{\mathbf{y}}_t - \bar{\mathbf{x}}_{t+\frac{1}{2}} \right\|^2 \right] \leq \frac{4u^4 \eta^2}{(1-u)^4} \mathbb{E} \left[\sum_{t=0}^{T-1} \left\| \bar{\mathbf{g}}_{t+\frac{1}{2}} \right\|^2 \right] \leq \frac{4u^4 \eta^2}{(1-u)^4} \frac{1}{BK} T \sigma^2 + \frac{4u^4 \eta^2}{(1-u)^4} \sum_{t=0}^{T-1} \left\| \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\|_2^2.$$

Proof. The proof starts from

$$\begin{aligned} \bar{\mathbf{y}}_t - \bar{\mathbf{x}}_{t+\frac{1}{2}} &= \frac{u}{1-u} (\bar{\mathbf{x}}_{t+\frac{1}{2}} - \bar{\mathbf{x}}_{t-\frac{1}{2}}) + \frac{\eta u}{1-u} \bar{\mathbf{g}}_{t-\frac{1}{2}} + \frac{\hat{\eta}}{1-u} (\bar{\xi}_t - u \bar{\xi}_{t-1}) \\ &= \frac{u}{1-u} (u \mathbf{v}_t - \eta \bar{\mathbf{g}}_{t-\frac{1}{2}} + \hat{\eta} \bar{\xi}_{t-1} - \hat{\eta} \bar{\xi}_t) + \frac{\eta u}{1-u} \bar{\mathbf{g}}_{t-\frac{1}{2}} + \frac{\hat{\eta}}{1-u} (\bar{\xi}_t - u \bar{\xi}_{t-1}) \\ &= \frac{u^2}{1-u} \mathbf{v}_t - \frac{u\eta}{1-u} \bar{\mathbf{g}}_{t-\frac{1}{2}} + \frac{u\hat{\eta}}{1-u} \bar{\xi}_{t-1} - \frac{u\hat{\eta}}{1-u} \bar{\xi}_t + \frac{\eta u}{1-u} \bar{\mathbf{g}}_{t-\frac{1}{2}} + \frac{\hat{\eta}}{1-u} \bar{\xi}_t - \frac{\hat{\eta} u}{1-u} \bar{\xi}_{t-1} \\ &= \frac{u^2}{1-u} \mathbf{v}_t - \frac{u\hat{\eta}}{1-u} \bar{\xi}_t + \frac{\hat{\eta}}{1-u} \bar{\xi}_t \\ &= \frac{u^2}{1-u} \mathbf{v}_t + \hat{\eta} \bar{\xi}_t. \end{aligned}$$

Thus, we have $\left\| \bar{\mathbf{y}}_t - \bar{\mathbf{x}}_{t+\frac{1}{2}} \right\|^2 = \left\| \frac{u^2}{1-u} \mathbf{v}_t + \hat{\eta} \bar{\xi}_t \right\|^2 \leq (1+\beta) \frac{u^4}{(1-u)^2} \|\mathbf{v}_t\|^2 + (1+\frac{1}{\beta}) \hat{\eta}^2 \|\bar{\xi}_t\|^2$, where $\|\mathbf{a} + \mathbf{b}\|^2 \leq (1+\beta) \|\mathbf{a}\|^2 + (1+\beta^{-1}) \|\mathbf{b}\|^2$ for $\beta > 0$.

Summing over $\{0, \dots, T-1\}$, we then have

$$\sum_{t=0}^{T-1} \left\| \bar{\mathbf{y}}_t - \bar{\mathbf{x}}_{t+\frac{1}{2}} \right\|^2 \leq (1+\beta) \frac{u^4}{(1-u)^2} \sum_{t=0}^{T-1} \|\mathbf{v}_t\|^2 + (1+\frac{1}{\beta}) \hat{\eta}^2 \sum_{t=0}^{T-1} \|\bar{\xi}_t\|^2.$$

We define $s_t := \sum_{i=0}^{t-1} u^{t-i-1} = \frac{1-u^t}{1-u} \leq \frac{1}{1-u}$. Noticing $\mathbf{v}_0 = 0$ and $T \geq 2$, we have

$$\begin{aligned} \sum_{t=0}^{T-1} \|\mathbf{v}_t\|^2 &= \sum_{t=1}^{T-1} \|\mathbf{v}_t\|^2 \\ &= \eta^2 \sum_{t=1}^{T-1} \left\| \sum_{i=1}^t u^{t-i} \bar{\mathbf{g}}_{i-\frac{1}{2}} \right\|^2 = \eta^2 \sum_{t=1}^{T-1} s_t^2 \left\| \sum_{i=0}^{t-1} \frac{u^{t-i-1}}{s_t} \bar{\mathbf{g}}_{i+\frac{1}{2}} \right\|^2 \\ &\leq \eta^2 \sum_{t=1}^{T-1} s_t^2 \sum_{i=0}^{t-1} \frac{u^{t-i-1}}{s_t} \|\bar{\mathbf{g}}_{i+\frac{1}{2}}\|^2 = \eta^2 \sum_{t=1}^{T-1} s_t \sum_{i=0}^{t-1} u^{t-i-1} \|\bar{\mathbf{g}}_{i+\frac{1}{2}}\|^2 \\ &\leq \frac{\eta^2}{1-u} \sum_{t=1}^{T-1} \sum_{i=0}^{t-1} u^{t-i-1} \|\bar{\mathbf{g}}_{i+\frac{1}{2}}\|^2 = \frac{\eta^2}{1-u} \sum_{i=0}^{T-2} \|\bar{\mathbf{g}}_{i+\frac{1}{2}}\|^2 \sum_{t=i+1}^{T-1} u^{t-i-1} \\ &\leq \frac{\eta^2}{1-u} \sum_{i=0}^{T-1} \|\bar{\mathbf{g}}_{i+\frac{1}{2}}\|^2 \sum_{t=0}^{\infty} u^t = \frac{\eta^2}{(1-u)^2} \sum_{t=0}^{T-1} \|\bar{\mathbf{g}}_{t+\frac{1}{2}}\|^2, \end{aligned}$$

which implies

$$\sum_{t=0}^{T-1} \|\mathbf{y}_t - \mathbf{x}_{t+\frac{1}{2}}\|^2 \leq (1+\beta) \frac{u^4 \eta^2}{(1-u)^4} \sum_{t=0}^{T-1} \|\tilde{\mathbf{g}}_{t+\frac{1}{2}}\|^2 + (1+\frac{1}{\beta}) \hat{\eta}^2 \sum_{t=0}^{T-1} \|\tilde{\xi}_t\|^2.$$

Further if we reuse the past local gradients with mini-batch B , i.e. $\tilde{\xi}_t = \begin{cases} \mathbf{0} & \text{if } t = 0 \\ \tilde{\mathbf{g}}_{t-\frac{1}{2}} & \text{if } t \geq 1 \end{cases}$, and

choose $\beta = 1$ and $\hat{\eta} \leq \frac{u^2}{(1-u)^2} \eta$, we obtain

$$\sum_{t=0}^{T-1} \|\mathbf{y}_t - \mathbf{x}_{t+\frac{1}{2}}\|^2 \leq \frac{4u^4 \eta^2}{(1-u)^4} \sum_{t=0}^{T-1} \|\tilde{\mathbf{g}}_{t+\frac{1}{2}}\|^2.$$

By using the fact that $\mathbb{E}[\|X\|^2] = \text{var}[X] + \|\mathbb{E}[X]\|^2$ and that $\text{var}[\sum_i X_i] = \sum_i \text{var}[X_i]$ if X_i 's are independent, we get

$$\begin{aligned} \mathbb{E} \left[\|\tilde{\mathbf{g}}_{t+\frac{1}{2}}\|^2 \right] &= \mathbb{E} \left[\left\| \frac{1}{BK} \sum_{k=1}^K \sum_{i \in \mathcal{I}_t^k} \mathbf{g}_{t+\frac{1}{2},i}^k \right\|^2 \right] = \frac{1}{B^2 K^2} \sum_{k=1}^K \sum_{i \in \mathcal{I}_t^k} \text{var} \left[\mathbf{g}_{t+\frac{1}{2},i}^k \right] + \left\| \mathbb{E} \left[\frac{1}{BK} \sum_{k=1}^K \sum_{i \in \mathcal{I}_t^k} \mathbf{g}_{t+\frac{1}{2},i}^k \right] \right\|_2^2 \\ &\leq \frac{1}{BK} \sigma^2 + \left\| \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\|_2^2, \end{aligned}$$

where we rely on Assumption 1, 2. Thus,

$$\begin{aligned} \mathbb{E} \left[\sum_{t=0}^{T-1} \|\tilde{\mathbf{y}}_t - \tilde{\mathbf{x}}_{t+\frac{1}{2}}\|^2 \right] &\leq \frac{4u^4 \eta^2}{(1-u)^4} \sum_{t=0}^{T-1} \left(\frac{1}{BK} \sigma^2 + \left\| \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\|_2^2 \right) \\ &= \frac{4u^4 \eta^2}{(1-u)^4} \frac{1}{BK} T \sigma^2 + \frac{4u^4 \eta^2}{(1-u)^4} \sum_{t=0}^{T-1} \left\| \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\|_2^2. \end{aligned}$$

□

Lemma 12.2.3. For all $t \geq 0$, $\mathbf{x}_{t+\frac{1}{2}}^k$ defined in (12.5) and $\tilde{\mathbf{x}}_{t+\frac{1}{2}}$ in (12.9), we have

$$\mathbb{E} \left[\frac{1}{K} \sum_{k=1}^K \|\tilde{\mathbf{x}}_{t+\frac{1}{2}} - \mathbf{x}_{t+\frac{1}{2}}^k\|^2 \right] \leq \mathbb{E} \left[\hat{\eta}^2 \left\| \tilde{\xi}_t - \frac{1}{b} \sum_{i \in \mathcal{I}_t^k} \xi_{t,i}^k \right\|^2 \right], \quad (12.10)$$

where $b := |\mathcal{I}_t^k|$.

If we reuse past local gradients, i.e. $\tilde{\xi}_t = \begin{cases} \mathbf{0} & \text{if } t = 0 \\ \tilde{\mathbf{g}}_{t-\frac{1}{2}} & \text{if } t \geq 1 \end{cases}$, we have $\mathbb{E} \left[\frac{1}{K} \sum_{k=1}^K \|\tilde{\mathbf{x}}_{t+\frac{1}{2}} - \mathbf{x}_{t+\frac{1}{2}}^k\|^2 \right] \leq \frac{4\hat{\eta}^2}{b} \sigma^2$.

Chapter 12. Appendix for EXTRAP-SGD

Alternatively if we use i.i.d. noise, i.e. $\frac{1}{b} \sum_{i \in \hat{\mathcal{J}}_t^k} \xi_{t,i}^k = \begin{cases} \mathbf{0} & \text{if } t = 0 \\ \zeta_t^k & \text{if } t \geq 1 \end{cases}$, with ζ_t^k being i.i.d. $\mathbb{E}[\zeta_t^k] = \mathbf{0}$

and $\mathbb{E}[\|\zeta_t^k\|^2] \leq \hat{\sigma}^2$, we have $\mathbb{E}\left[\frac{1}{K} \sum_{k=1}^K \|\bar{\mathbf{x}}_{t+\frac{1}{2}} - \mathbf{x}_{t+\frac{1}{2}}^k\|^2\right] \leq 2\hat{\eta}^2 \hat{\sigma}^2$.

Proof. By definition, for $t = 0$, $\mathbb{E}\left[\|\bar{\mathbf{x}}_{\frac{1}{2}} - \mathbf{x}_{\frac{1}{2}}^k\|^2\right] = 0$.

For $t \geq 1$, we have the following

$$\mathbb{E}\left[\|\bar{\mathbf{x}}_{t+\frac{1}{2}} - \mathbf{x}_{t+\frac{1}{2}}^k\|^2\right] = \mathbb{E}\left[\left\|\mathbf{x}_t + u\mathbf{v}_t - \hat{\eta}\bar{\xi}_t - \left(\mathbf{x}_t + u\mathbf{v}_t - \frac{\hat{\eta}}{b} \sum_{i \in \hat{\mathcal{J}}_t^k} \xi_{t,i}^k\right)\right\|^2\right] = \mathbb{E}\left[\hat{\eta}^2 \left\|\bar{\xi}_t - \frac{1}{b} \sum_{i \in \hat{\mathcal{J}}_t^k} \xi_{t,i}^k\right\|^2\right]$$

If we reuse past local gradients i.e. $\xi_{t,i}^k = \mathbf{g}_{t-\frac{1}{2}}^k$, then we have

$$\begin{aligned} \mathbb{E}\left[\hat{\eta}^2 \left\|\bar{\xi}_t - \frac{1}{b} \sum_{i \in \hat{\mathcal{J}}_t^k} \xi_{t,i}^k\right\|^2\right] &= \mathbb{E}\left[\hat{\eta}^2 \left\|\bar{\mathbf{g}}_{t-\frac{1}{2}} - \nabla f(\mathbf{x}_{t-\frac{1}{2}}) + \nabla f(\mathbf{x}_{t-\frac{1}{2}}) - \mathbf{g}_{t-\frac{1}{2}}^k\right\|^2\right] \\ &\leq 2\hat{\eta}^2 \mathbb{E}\left[\left(\left\|\bar{\mathbf{g}}_{t-\frac{1}{2}} - \nabla f(\mathbf{x}_{t-\frac{1}{2}})\right\|^2 + \left\|\nabla f(\mathbf{x}_{t-\frac{1}{2}}) - \mathbf{g}_{t-\frac{1}{2}}^k\right\|^2\right)\right] \\ &\stackrel{(a)}{\leq} 2\hat{\eta}^2 \left(\frac{1}{Kb} + \frac{1}{b}\right) \sigma^2 \leq \frac{4\hat{\eta}^2}{b} \sigma^2, \end{aligned}$$

where (a) is from the Assumption 2 and the independence between data samples.

Alternatively if we use i.i.d. noise, i.e. $\frac{1}{b} \sum_{i \in \hat{\mathcal{J}}_t^k} \xi_{t,i}^k = \begin{cases} \mathbf{0} & \text{if } t = 0 \\ \zeta_t^k & \text{if } t \geq 1 \end{cases}$, with ζ_t^k being i.i.d., $\mathbb{E}[\zeta_t^k] = \mathbf{0}$

and $\mathbb{E}[\|\zeta_t^k\|^2] \leq \hat{\sigma}^2$, we have

$$\begin{aligned} \mathbb{E}\left[\hat{\eta}^2 \left\|\bar{\xi}_t - \frac{1}{b} \sum_{i \in \hat{\mathcal{J}}_t^k} \xi_{t,i}^k\right\|^2\right] &= \hat{\eta}^2 \mathbb{E}\left[\left\|\zeta_t^k - \frac{1}{K} \sum_{k=1}^K \zeta_t^k\right\|^2\right] \\ &= \hat{\eta}^2 \mathbb{E}\left[\left\|\zeta_t^k\right\|^2 + \left\|\frac{1}{K} \sum_{k=1}^K \zeta_t^k\right\|^2 - \frac{2}{K} \sum_{i=1}^K \langle \zeta_t^i, \zeta_t^k \rangle\right] \\ &= \hat{\eta}^2 \mathbb{E}\left[\left\|\zeta_t^k\right\|^2 + \frac{1}{K^2} \sum_{k=1}^K \left\|\zeta_t^k\right\|^2 - \frac{2}{K} \left\|\zeta_t^k\right\|^2\right] \\ &\leq \hat{\eta}^2 \left(\hat{\sigma}^2 + \frac{1}{K} \hat{\sigma}^2\right) \leq 2\hat{\eta}^2 \hat{\sigma}^2. \end{aligned}$$

Combining these yields the lemma. □

12.2.1 Main Proof of Theorem 3.5.4

We duplicate the Theorem 3.5.4 appearing in the main paper as the Theorem 12.2.4.

Theorem 12.2.4. Consider the update rule of EXTRAP-SGD with notations defined in (12.6),

$$\mathbf{x}_{t+\frac{1}{4}} = \mathbf{x}_t + u\mathbf{v}_t, \mathbf{x}_{t+\frac{1}{2}}^k = \mathbf{x}_{t+\frac{1}{4}} - \hat{\eta}\mathbf{g}_{t-\frac{1}{2}}^k, \mathbf{v}_{t+1} = u\mathbf{v}_t - \eta\bar{\mathbf{g}}_{t+\frac{1}{2}}, \mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{v}_{t+1}.$$

We define $\bar{\mathbf{x}}_{t+\frac{1}{2}} := \frac{1}{K} \sum_{k=1}^K \mathbf{x}_{t+\frac{1}{2}}^k$ as well as the following virtual sequence

$$\bar{\mathbf{y}}_t := \begin{cases} \bar{\mathbf{x}}_{\frac{1}{2}} = \mathbf{x}_0 & \text{if } t = 0 \\ \frac{1}{1-u}\bar{\mathbf{x}}_{t+\frac{1}{2}} - \frac{u}{1-u}\bar{\mathbf{x}}_{t-\frac{1}{2}} + \frac{\eta u}{1-u}\bar{\mathbf{g}}_{t-\frac{1}{2}} + \frac{\hat{\eta}}{1-u}(\bar{\mathbf{g}}_{t-\frac{1}{2}} - u\bar{\mathbf{g}}_{t-\frac{3}{2}}) & \text{if } t \geq 1 \end{cases},$$

where $\bar{\mathbf{g}}_{-\frac{1}{2}} = 0$ by default.

Under Assumption 1- 3, the convergence rate of $\bar{\mathbf{x}}_{t+\frac{1}{2}}$ for a non-convex function follows

$$\mathbb{E} \left[\frac{1}{T} \sum_{t=0}^{T-1} \left\| \nabla f(\bar{\mathbf{x}}_{t+\frac{1}{2}}) \right\|^2 \right] \leq \frac{2(1-u)}{\eta T} \mathbb{E} [f(\bar{\mathbf{x}}_0) - f^*] + \left(\frac{4\hat{\eta}^2 L^2}{B} + \frac{\eta L(1+3u)}{(1-u)^2 BK} \right) \sigma^2,$$

where $\hat{\eta} \leq \frac{u^2}{(1-u)^2} \eta$ and $\eta \leq \frac{(1-u)^2}{L(1+3u+u^3)}$.

Proof. We start from the standard smoothness inequality

$$\begin{aligned} \mathbb{E} [f(\bar{\mathbf{y}}_{t+1}) - f(\bar{\mathbf{y}}_t)] &\leq \mathbb{E} \left[\langle \nabla f(\bar{\mathbf{y}}_t), \bar{\mathbf{y}}_{t+1} - \bar{\mathbf{y}}_t \rangle + \frac{L}{2} \|\bar{\mathbf{y}}_{t+1} - \bar{\mathbf{y}}_t\|^2 \right] \\ &\stackrel{(1)}{=} \mathbb{E} \left[\langle \nabla f(\bar{\mathbf{y}}_t), -\frac{\eta}{1-u}\bar{\mathbf{g}}_{t+\frac{1}{2}} \rangle \right] + \mathbb{E} \left[\frac{\eta^2 L}{2(1-u)^2} \|\bar{\mathbf{g}}_{t+\frac{1}{2}}\|^2 \right] \\ &= \mathbb{E} \left[\left\langle \nabla f(\bar{\mathbf{y}}_t) - \nabla f(\bar{\mathbf{x}}_{t+\frac{1}{2}}) + \nabla f(\bar{\mathbf{x}}_{t+\frac{1}{2}}), -\frac{\eta}{1-u} \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\rangle \right] + \mathbb{E} \left[\frac{\eta^2 L}{2(1-u)^2} \|\bar{\mathbf{g}}_{t+\frac{1}{2}}\|^2 \right] \\ &= \mathbb{E} \left[\underbrace{\left\langle \nabla f(\bar{\mathbf{y}}_t) - \nabla f(\bar{\mathbf{x}}_{t+\frac{1}{2}}), -\frac{\eta}{1-u} \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\rangle}_{(a)} \right] \\ &\quad + \mathbb{E} \left[\underbrace{\left\langle \nabla f(\bar{\mathbf{x}}_{t+\frac{1}{2}}), -\frac{\eta}{1-u} \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\rangle}_{(b)} \right] \\ &\quad + \frac{\eta^2 L}{2(1-u)^2} \underbrace{\mathbb{E} \left[\|\bar{\mathbf{g}}_{t+\frac{1}{2}}\|^2 \right]}_{(c)}, \end{aligned}$$

where we use Lemma 12.2.1 for (1).

For (a), we have

$$\begin{aligned}
 & \mathbb{E} \left[\left\langle \nabla f(\bar{\mathbf{y}}_t) - \nabla f(\bar{\mathbf{x}}_{t+\frac{1}{2}}), -\frac{\eta}{1-u} \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\rangle \right] \\
 & \stackrel{(2)}{\leq} \mathbb{E} \left[\frac{1-u}{2u^3 L} \left\| \nabla f(\bar{\mathbf{y}}_t) - \nabla f(\bar{\mathbf{x}}_{t+\frac{1}{2}}) \right\|^2 \right] + \mathbb{E} \left[\frac{L\eta^2 u^3}{2(1-u)^3} \left\| \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\|^2 \right] \\
 & \stackrel{(3)}{\leq} \mathbb{E} \left[\frac{(1-u)L}{2u^3} \left\| \bar{\mathbf{y}}_t - \bar{\mathbf{x}}_{t+\frac{1}{2}} \right\|^2 \right] + \mathbb{E} \left[\frac{L\eta^2 u^3}{2(1-u)^3} \left\| \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\|^2 \right],
 \end{aligned}$$

where the inequality (2) is due to the basic inequality $\langle \mathbf{a}, \mathbf{b} \rangle \leq \frac{\beta}{2} \|\mathbf{a}\|^2 + \frac{1}{2\beta} \|\mathbf{b}\|^2$ with $\beta = \frac{1-u}{Lu^3}$ and (3) is from Assumption 3.

For (b), we have

$$\begin{aligned}
 & \mathbb{E} \left[\left\langle \nabla f(\bar{\mathbf{x}}_{t+\frac{1}{2}}), -\frac{\eta}{1-u} \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\rangle \right] \\
 & = -\frac{\eta}{1-u} \mathbb{E} \left[\left\langle \nabla f(\bar{\mathbf{x}}_{t+\frac{1}{2}}), \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\rangle \right] \\
 & \stackrel{(4)}{=} -\frac{\eta}{2(1-u)} \mathbb{E} \left[\left\| \nabla f(\bar{\mathbf{x}}_{t+\frac{1}{2}}) \right\|^2 + \left\| \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\|^2 - \left\| \nabla f(\bar{\mathbf{x}}_{t+\frac{1}{2}}) - \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\|^2 \right] \\
 & \stackrel{(5)}{\leq} -\frac{\eta}{2(1-u)} \mathbb{E} \left[\left\| \nabla f(\bar{\mathbf{x}}_{t+\frac{1}{2}}) \right\|^2 + \left\| \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\|^2 - \frac{L^2}{K} \mathbb{E} \left[\sum_{k=1}^K \left\| \bar{\mathbf{x}}_{t+\frac{1}{2}} - \mathbf{x}_{t+\frac{1}{2}}^k \right\|^2 \right] \right] \\
 & \stackrel{(6)}{\leq} -\frac{\eta}{2(1-u)} \left(\mathbb{E} \left[\left\| \nabla f(\bar{\mathbf{x}}_{t+\frac{1}{2}}) \right\|^2 \right] + \mathbb{E} \left[\left\| \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\|^2 \right] - \frac{4\hat{\eta}^2 L^2}{B} \sigma^2 \right),
 \end{aligned}$$

where the equality (4) comes from $\langle \mathbf{a}, \mathbf{b} \rangle = \frac{1}{2} (\|\mathbf{a}\|^2 + \|\mathbf{b}\|^2 - \|\mathbf{a} - \mathbf{b}\|^2)$, the inequality (5) is due to Assumption 3 and (6) is from Lemma 12.2.3.

For (c), by using $\mathbb{E}[\|X\|^2] = \text{var}[X] + \|\mathbb{E}[X]\|^2$ and the fact that $\text{var}[\sum_i X_i] = \sum_i \text{var}[X_i]$ if X_i 's are independent, we get

$$\begin{aligned}
 \mathbb{E} \left[\left\| \bar{\mathbf{g}}_{t+\frac{1}{2}} \right\|^2 \right] & = \mathbb{E} \left[\left\| \frac{1}{BK} \sum_{k=1}^K \sum_{i \in \mathcal{I}_t^k} \mathbf{g}_{t+\frac{1}{2}, i}^k \right\|^2 \right] \\
 & = \frac{1}{B^2 K^2} \sum_{k=1}^K \sum_{i \in \mathcal{I}_t^k} \text{var}[\mathbf{g}_{t+\frac{1}{2}, i}^k] + \left\| \mathbb{E} \left[\frac{1}{BK} \sum_{k=1}^K \sum_{i \in \mathcal{I}_t^k} \mathbf{g}_{t+\frac{1}{2}, i}^k \right] \right\|_2^2 \\
 & \stackrel{(7)}{\leq} \frac{1}{BK} \sigma^2 + \left\| \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\|_2^2,
 \end{aligned}$$

where (7) follows from Assumption 2.

Thus the smoothness inequality becomes

$$\begin{aligned}
 \mathbb{E}[f(\bar{\mathbf{y}}_{t+1}) - f(\bar{\mathbf{y}}_t)] &\leq \frac{(1-u)L}{2u^3} \mathbb{E} \left[\left\| \bar{\mathbf{y}}_t - \bar{\mathbf{x}}_{t+\frac{1}{2}} \right\|^2 \right] + \left(\frac{L\eta^2 u^3}{2(1-u)^3} - \frac{\eta}{2(1-u)} \right) \mathbb{E} \left[\left\| \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\|^2 \right] \\
 &\quad - \frac{\eta}{2(1-u)} \mathbb{E} \left[\left\| \nabla f(\bar{\mathbf{x}}_{t+\frac{1}{2}}) \right\|^2 \right] + \frac{\eta}{2(1-u)} \frac{4\hat{\eta}^2 L^2}{B} \sigma^2 + \frac{\eta^2 L}{2(1-u)^2} \left(\frac{1}{BK} \sigma^2 + \mathbb{E} \left[\left\| \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\|^2 \right] \right) \\
 &= \frac{(1-u)L}{2u^3} \mathbb{E} \left[\left\| \bar{\mathbf{y}}_t - \bar{\mathbf{x}}_{t+\frac{1}{2}} \right\|^2 \right] + \left(\frac{L\eta^2 u^3}{2(1-u)^3} - \frac{\eta}{2(1-u)} + \frac{\eta^2 L}{2(1-u)^2} \right) \mathbb{E} \left[\left\| \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\|^2 \right] \\
 &\quad - \frac{\eta}{2(1-u)} \mathbb{E} \left[\left\| \nabla f(\bar{\mathbf{x}}_{t+\frac{1}{2}}) \right\|^2 \right] + \left(\frac{\eta}{2(1-u)} \frac{4\hat{\eta}^2 L^2}{B} + \frac{\eta^2 L}{2(1-u)^2} \frac{1}{BK} \right) \sigma^2.
 \end{aligned}$$

By rearranging, summing over t , and diving both sides by $\frac{\eta}{2(1-u)}$, we have

$$\begin{aligned}
 \mathbb{E} \left[\sum_{t=0}^{T-1} \left\| \nabla f(\bar{\mathbf{x}}_{t+\frac{1}{2}}) \right\|^2 \right] &\leq \frac{2(1-u)}{\eta} \mathbb{E}[f(\bar{\mathbf{y}}_0) - f(\bar{\mathbf{y}}_T)] + \frac{(1-u)^2 L}{\eta u^3} \mathbb{E} \left[\sum_{t=0}^{T-1} \left\| \bar{\mathbf{y}}_t - \bar{\mathbf{x}}_{t+\frac{1}{2}} \right\|^2 \right] \\
 &\quad + \left(\frac{\eta L u^3 + \eta L (1-u)}{(1-u)^2} - 1 \right) \sum_{t=0}^{T-1} \mathbb{E} \left[\left\| \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\|^2 \right] + \left(\frac{4\hat{\eta}^2 L^2}{B} + \frac{\eta L}{1-u} \frac{1}{BK} \right) T \sigma^2.
 \end{aligned}$$

Through Lemma 12.2.2, we have

$$\mathbb{E} \left[\sum_{t=0}^{T-1} \left\| \bar{\mathbf{y}}_t - \bar{\mathbf{x}}_{t+\frac{1}{2}} \right\|^2 \right] \leq \mathbb{E} \left[\frac{4u^4 \eta^2}{(1-u)^4} \sum_{t=0}^{T-1} \left\| \bar{\mathbf{g}}_{t+\frac{1}{2}} \right\|^2 \right] \leq \frac{4u^4 \eta^2}{(1-u)^4} \frac{1}{BK} T \sigma^2 + \frac{4u^4 \eta^2}{(1-u)^4} \sum_{t=0}^{T-1} \mathbb{E} \left[\left\| \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\|^2 \right],$$

thus,

$$\begin{aligned}
 \mathbb{E} \left[\sum_{t=0}^{T-1} \left\| \nabla f(\bar{\mathbf{x}}_{t+\frac{1}{2}}) \right\|^2 \right] &\leq \frac{2(1-u)}{\eta} \mathbb{E}[f(\bar{\mathbf{y}}_0) - f(\bar{\mathbf{y}}_T)] + \frac{(1-u)^2 L}{\eta u^3} \left(\frac{4u^4 \eta^2}{(1-u)^4} \frac{1}{BK} T \sigma^2 + \frac{4u^4 \eta^2}{(1-u)^4} \sum_{t=0}^{T-1} \left\| \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\|^2 \right) \\
 &\quad + \left(\frac{\eta L u^3 + \eta L (1-u)}{(1-u)^2} - 1 \right) \sum_{t=0}^{T-1} \left\| \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\|^2 + \left(\frac{4\hat{\eta}^2 L^2}{B} + \frac{\eta L}{1-u} \frac{1}{BK} \right) T \sigma^2 \\
 &\leq \frac{2(1-u)}{\eta} \mathbb{E}[f(\bar{\mathbf{y}}_0) - f(\bar{\mathbf{y}}_T)] \\
 &\quad + \left(\frac{\eta L u^3 + \eta L (1-u)}{(1-u)^2} + \frac{4u\eta L}{(1-u)^2} - 1 \right) \sum_{t=0}^{T-1} \left\| \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\|^2 \\
 &\quad + \left(\frac{4\hat{\eta}^2 L^2}{B} + \frac{\eta L}{1-u} \frac{1}{BK} + \frac{4u\eta L}{(1-u)^2} \frac{1}{BK} \right) T \sigma^2 \\
 &\leq \frac{2(1-u)}{\eta} \mathbb{E}[f(\bar{\mathbf{y}}_0) - f(\bar{\mathbf{y}}_T)] \\
 &\quad + \left(\frac{\eta L (u^3 + 3u + 1)}{(1-u)^2} - 1 \right) \sum_{t=0}^{T-1} \left\| \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\|^2 \\
 &\quad + \left(\frac{4\hat{\eta}^2 L^2}{B} + \frac{\eta L (1+3u)}{(1-u)^2 BK} \right) T \sigma^2.
 \end{aligned}$$

Dividing both sides by T , we have

$$\begin{aligned}
 &\mathbb{E} \left[\frac{1}{T} \sum_{t=0}^{T-1} \left\| \nabla f(\bar{\mathbf{x}}_{t+\frac{1}{2}}) \right\|^2 \right] \\
 &\leq \frac{2(1-u)}{\eta T} \mathbb{E}[f(\bar{\mathbf{y}}_0) - f(\bar{\mathbf{y}}_T)] + \frac{1}{T} \left(\frac{\eta L (u^3 + 3u + 1)}{(1-u)^2} - 1 \right) \sum_{t=0}^{T-1} \left\| \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\|^2 + \left(\frac{4\hat{\eta}^2 L^2}{B} + \frac{\eta L (1+3u)}{(1-u)^2 BK} \right) \sigma^2.
 \end{aligned}$$

Further we choose η to ensure the coefficient of the gradient norm on the RHS nonpositive (i.e. $\frac{L\eta(1+3u+u^3)}{(1-u)^2} - 1 \leq 0$, which implies $\eta \leq \frac{(1-u)^2}{L(1+3u+u^3)}$). As a result, we can conclude that

$$\mathbb{E} \left[\frac{1}{T} \sum_{t=0}^{T-1} \left\| \nabla f(\bar{\mathbf{x}}_{t+\frac{1}{2}}) \right\|^2 \right] \leq \frac{2(1-u)}{\eta T} \mathbb{E}[f(\bar{\mathbf{y}}_0) - f(\bar{\mathbf{y}}_T)] + \left(\frac{4\hat{\eta}^2 L^2}{B} + \frac{\eta L (1+3u)}{(1-u)^2 BK} \right) \sigma^2.$$

□

Theorem 12.2.5. *Under the extrapolation framework, if we use i.i.d. noise, i.e. $\frac{1}{b} \sum_{i \in \hat{\mathcal{S}}_t^k} \boldsymbol{\zeta}_{t,i}^k = \begin{cases} \mathbf{0} & \text{if } t=0 \\ \boldsymbol{\zeta}_t^k & \text{if } t \geq 1 \end{cases}$, with $\boldsymbol{\zeta}_t^k$ being i.i.d., $\mathbb{E}[\boldsymbol{\zeta}_t^k] = \mathbf{0}$ and $\mathbb{E}[\|\boldsymbol{\zeta}_t^k\|^2] \leq \hat{\sigma}^2$. Under the same assumption as in Theorem 3.5.4, we have the following convergence rate for a non-convex function:*

$$\mathbb{E} \left[\frac{1}{T} \sum_{t=0}^{T-1} \left\| \nabla f(\bar{\mathbf{x}}_{t+\frac{1}{2}}) \right\|^2 \right] \leq \frac{2(1-u)}{\eta T} \mathbb{E}[f(\bar{\mathbf{y}}_0) - f(\bar{\mathbf{y}}_T)] + \frac{\eta L (1+u)}{(1-u)^2 BK} \hat{\sigma}^2 + \left(L^2 + \frac{(1-u)^2 L}{\eta u^3 K} \right) 2\hat{\eta}^2 T \hat{\sigma}^2,$$

where $\eta \leq \frac{(1-u)^2}{L(1+u+u^3)}$.

Proof. Following similar procedures in the proof of Theorem 12.2.4, we have

$$\begin{aligned} \mathbb{E} \left[\sum_{t=0}^{T-1} \left\| \nabla f(\bar{\mathbf{x}}_{t+\frac{1}{2}}) \right\|^2 \right] &\leq \frac{2(1-u)}{\eta} \mathbb{E} [f(\bar{\mathbf{y}}_0) - f(\bar{\mathbf{y}}_T)] + \frac{(1-u)^2 L}{\eta u^3} \mathbb{E} \left[\sum_{t=0}^{T-1} \left\| \bar{\mathbf{y}}_t - \bar{\mathbf{x}}_{t+\frac{1}{2}} \right\|^2 \right] \\ &\quad + \left(\frac{\eta L u^3 + \eta L(1-u)}{(1-u)^2} - 1 \right) \sum_{t=0}^{T-1} \left\| \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\|^2 + 2\hat{\eta}^2 L^2 T \hat{\sigma}^2 + \frac{\eta L}{1-u} \frac{1}{BK} T \sigma^2. \end{aligned}$$

Simplifying Lemma 12.2.2 by choosing $\beta = 1$ and $\bar{\boldsymbol{\xi}}_t = \left\| \frac{1}{K} \sum_{k=1}^K \boldsymbol{\zeta}_t^k \right\|^2$ gives:

$$\begin{aligned} \mathbb{E} \left[\sum_{t=0}^{T-1} \left\| \mathbf{y}_t - \mathbf{x}_{t+\frac{1}{2}} \right\|^2 \right] &\leq \frac{2u^4 \eta^2}{(1-u)^4} \mathbb{E} \left[\sum_{t=0}^{T-1} \left\| \bar{\mathbf{g}}_{t+\frac{1}{2}} \right\|^2 \right] + \frac{2\hat{\eta}^2 T}{K} \hat{\sigma}^2 \\ &\leq \frac{2\hat{\eta}^2 T}{K} \hat{\sigma}^2 + \frac{2u^4 \eta^2}{(1-u)^4} \frac{1}{BK} T \sigma^2 + \frac{2u^4 \eta^2}{(1-u)^4} \sum_{t=0}^{T-1} \left\| \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\|^2, \end{aligned}$$

we have the main proof as follows:

$$\begin{aligned} &\mathbb{E} \left[\sum_{t=0}^{T-1} \left\| \nabla f(\bar{\mathbf{x}}_{t+\frac{1}{2}}) \right\|^2 \right] \\ &\leq \frac{2(1-u)}{\eta} \mathbb{E} [f(\bar{\mathbf{y}}_0) - f(\bar{\mathbf{y}}_T)] + \frac{(1-u)^2 L}{\eta u^3} \left(\frac{2\hat{\eta}^2 T}{K} \hat{\sigma}^2 + \frac{2u^4 \eta^2}{(1-u)^4} \frac{1}{BK} T \sigma^2 + \frac{2u^4 \eta^2}{(1-u)^4} \sum_{t=0}^{T-1} \left\| \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\|^2 \right) \\ &\quad + \left(\frac{\eta L u^3 + \eta L(1-u)}{(1-u)^2} - 1 \right) \sum_{t=0}^{T-1} \left\| \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\|^2 + \left(\frac{\eta L}{1-u} \frac{1}{BK} \right) T \sigma^2 + 2\hat{\eta}^2 L^2 T \hat{\sigma}^2 \\ &\leq \frac{2(1-u)}{\eta} \mathbb{E} [f(\bar{\mathbf{y}}_0) - f(\bar{\mathbf{y}}_T)] + \left(\frac{\eta L u^3 + \eta L(1-u)}{(1-u)^2} + \frac{2u\eta L}{(1-u)^2} - 1 \right) \sum_{t=0}^{T-1} \left\| \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\|^2 \\ &\quad + \left(\frac{\eta L}{1-u} + \frac{2u\eta L}{(1-u)^2} \right) \frac{T}{BK} \sigma^2 + \left(L^2 + \frac{(1-u)^2 L}{\eta u^3 K} \right) 2\hat{\eta}^2 T \hat{\sigma}^2 \\ &\leq \frac{2(1-u)}{\eta} \mathbb{E} [f(\bar{\mathbf{y}}_0) - f(\bar{\mathbf{y}}_T)] \\ &\quad + \left(\frac{\eta L(u^3 + u + 1)}{(1-u)^2} - 1 \right) \sum_{t=0}^{T-1} \left\| \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k) \right\|^2 + \frac{\eta L(1+u)}{(1-u)^2 BK} T \sigma^2 + \left(L^2 + \frac{(1-u)^2 L}{\eta u^3 K} \right) 2\hat{\eta}^2 T \hat{\sigma}^2 \end{aligned}$$

Dividing both sides by T and choosing η such that $\frac{L\eta(1+u+u^3)}{(1-u)^2} - 1 < 0$ yields:

$$\mathbb{E} \left[\frac{1}{T} \sum_{t=0}^{T-1} \left\| \nabla f(\bar{\mathbf{x}}_{t+\frac{1}{2}}) \right\|^2 \right] \leq \frac{2(1-u)}{\eta T} \mathbb{E} [f(\bar{\mathbf{y}}_0) - f(\bar{\mathbf{y}}_T)] + \frac{\eta L(1+u)}{(1-u)^2 BK} \sigma^2 + \left(L^2 + \frac{(1-u)^2 L}{\eta u^3 K} \right) 2\hat{\eta}^2 T \hat{\sigma}^2$$

where we have $\eta \leq \frac{(1-u)^2}{L(1+u+u^3)}$. □

12.2.2 Proof of Corollary 3.5.6

Proof. Following the proof of Corollary 12.1.5, we define $r_0 := f(\mathbf{x}_0) - f^*$ and $\Psi'_T := \frac{2}{T} \frac{\eta}{1-u} r_0 + \left(\frac{4\hat{\eta}^2 L^2}{B} + \frac{\eta L(1+3u)}{(1-u)^2 BK} \right) \sigma^2$.

$$\begin{aligned}
 \Psi'_T &\stackrel{(a)}{\leq} \frac{2}{T} \frac{\eta}{1-u} r_0 + \left(\frac{4u^2 \eta^2 L^2}{BK(1-u)^2} + \frac{\eta L(1+3u)}{(1-u)^2 BK} \right) \sigma^2 \\
 &\stackrel{(b)}{\leq} \frac{2}{T} \frac{\eta}{1-u} r_0 + \left(\frac{4u^2 \eta L}{BK(1-u)^2} \frac{(1-u)^2}{u^3+3u+1} + \frac{\eta L(1+3u)}{(1-u)^2 BK} \right) \sigma^2 \\
 &= \frac{2}{T} \frac{\eta}{1-u} r_0 + \frac{\eta L \sigma^2}{BK} \left(\frac{4u^2}{u^3+3u+1} + \frac{1+3u}{(1-u)^2} \right) \\
 &= \frac{2}{T} \frac{\eta}{1-u} r_0 + \frac{\eta L \sigma^2}{BK} \frac{7u^4 - 7u^3 + 13u^2 + 6u + 1}{(u^3+3u+1)(1-u)^2} \\
 &\stackrel{(c)}{\leq} \frac{2}{T} \frac{\eta}{1-u} r_0 + \frac{\eta L \sigma^2}{BK} \frac{19u+1}{(u^3+3u+1)(1-u)^2},
 \end{aligned}$$

where (a) follows from $\hat{\eta}^2 \leq \frac{u^2}{K(1-u)^2} \eta^2$ (because of $\hat{\eta} \leq \frac{\eta}{K}$ and $\hat{\eta} \leq \frac{u^2}{(1-u)^2} \eta$), (b) is from $\eta L \leq \frac{(1-u)^2}{1+3u+u^3}$ and (c) comes from $u^4 \leq u^3 \leq u^2 \leq u$.

Similarly, we could choose two values of η as the minimizer, where $\eta = \sqrt{\frac{2r_0 KB}{L\sigma^2 T} \frac{(u^3+3u+1)(1-u)^3}{19u+1}}$ and $\eta = \frac{(1-u)^2}{L(1+3u+u^3)}$. Thus, by considering two cases as in Lemma 12.1.4, we have $\Psi'_T \leq \frac{4Lr_0(u^3+3u+1)}{T(1-u)} + 2\sqrt{\frac{2Lr_0\sigma^2(19u+1)}{KBT(u^3+3u+1)(1-u)}}$. \square

12.3 Detailed Experimental Setup

Datasets. We evaluate all methods on the following two tasks: (1) Image classification for CIFAR-10/100 (Krizhevsky and Hinton, 2009) (50K training samples and 10K test samples with 10/100 classes) with the standard data augmentation and preprocessing scheme (He et al., 2016a; Huang et al., 2016); (2) Language modeling for WikiText2 (Merity et al., 2017) (the vocabulary size is 33K, and its train and validation set have 2 million tokens and 217K tokens respectively); and (3) Neural Machine Translation for Multi30k (Elliott et al., 2016).

Large-batch training in practice. We detail the SOTA large-batch training techniques used in our experiment evaluation:

- **Better Optimization:** Goyal et al. (2017) from optimization aspect propose to linearly scale the learning rate with a few epochs warmup. (1) multiply the learning rate by K when the mini-batch size is multiplied by K ; (2) warmup the learning rate from η to $K\eta$ through H epochs, where the incremental learning rate for each iteration is calculated from $\frac{K\eta - \eta}{HN/(KB)}$. Note that N is the number of total training samples and B is the local mini-batch size.

LARS (You et al., 2017c, 2020b) argue the layerwise difference in the weight magnitude and propose to scale the gradient of each layer accordingly for better optimization. The scaled gradient for j -th layer of \mathbf{x}_t at i -th sample follows $\nabla f_{i,j}(\mathbf{x}_t) \times \left(\tilde{\eta} \times \frac{\|\mathbf{x}_{t,j}\|}{\|\nabla f_{i,j}(\mathbf{x}_t)\| + \lambda \|\mathbf{x}_{t,j}\|} \right)$, where $\tilde{\eta}$ defines how much we trust the layer to change its weights during one update.

- **Better Generalization:**

On top of these optimization techniques (Goyal et al., 2017; You et al., 2017c), Post-local SGD (Lin et al., 2020d) propose to further inject stochastic noise when converging to the local minima, targeting better generalization. The stochastic noise is introduced by performing local SGD updates.

Models and training schemes. Three models are used in our experimental evaluation. (1) ResNet-20 (He et al., 2016a) and VGG-11¹ (Simonyan and Zisserman, 2015) on CIFAR for image classification, (2) two-layer LSTM² (Merity et al., 2018) with hidden dimension of size 128 on WikiText-2 for language modeling, and (3) a down-scaled transformer (factor of 2 w.r.t. the transformer base model in Vaswani et al. (2017)) for neural machine translation. Weight initialization schemes for the three models follow Goyal et al. (2017); He et al. (2015), Merity et al. (2018) and Vaswani et al. (2017) respectively.

¹ Due to the resource constraints (GPU memory bound), we down-scaled the original VGG-11 by reducing the number of filters by the factor of 2.

² We borrowed and adapted the general experimental setup of Merity et al. (2018). The gradient clip magnitude is 0.4, and dropout rate is 0.40. The loss is averaged over all examples and timesteps.

We use mini-batch SGD with a Nesterov momentum of 0.9 without dampening for image classification and language modeling tasks, and Adam for neural machine translation tasks. Unless mentioned otherwise in the following experiment section, the term “mini-batch SGD” indicates the mini-batch SGD with Nesterov momentum.

For experiments on image classification and language modeling, the models are trained for 300 epochs; the local mini-batch sizes are set to 256 and 64 respectively. By default all related experiments will use learning rate scaling and warmup scheme³ (Goyal et al., 2017; Hoffer et al., 2017). The learning rate will gradually warm up from a relative small value (e.g. 0.1) for the first few epochs. The weight decay of ResNet-20 and LSTM are $1e-4$ (He et al., 2016a) and 0 (Merity et al., 2018) respectively. For the Batch Normalization (BN) (Ioffe and Szegedy, 2015) for distributed training we follow Goyal et al. (2017) and compute the BN statistics independently for each worker; we also do not apply weight decay on the learnable BN coefficients (He et al., 2016a). In addition, the learning rate η in image classification task will be dropped by a factor of 10 when the model has accessed 50% and 75% of the total number of training samples (He et al., 2016a; Huang et al., 2017b). The LARS is only applied on image classification task⁴ (You et al., 2017c, 2020b).

For experiments on neural machine translation, we use standard inverse square root learning rate schedule as in Vaswani et al. (2017). The warmup step is set to 4000 for mini-batch size of 64 and will be linearly scaled down by the global mini-batch size.⁵ Other hyperparameters follow the default setting in Vaswani et al. (2017). The first and second moment factor of the adam are set to 0.90 and 0.98 respectively, with the epsilon 10^{-9} . The values of dropout rate and the label smoothing factor are set to 0.1. The weight decay factor is set to 0.

Implementation and platform. Our algorithms are implemented in PyTorch⁶ (Paszke et al., 2017) and the distributed training is supported by MPI and Kubernetes.

12.3.1 Hyperparameter Tuning Procedure and the Corresponding Values

We carefully tune the learning rate, the trust term $\tilde{\eta}$ in You et al. (2017c) and our extrapolation term $\hat{\eta}$ for each experimental setup. For example, for ResNet-20 on CIFAR-10 we tune the optimal unscaled learning rate (i.e. η/K) for a fixed mini-batch size B in the range of $\{0.05, 0.10, 0.15, 0.20\}$ and then linearly scale (and warmup) the learning rate by the factor of K . The $\tilde{\eta}$ is initially searched within $\{0.01, 0.02, 0.03\}$, where 0.02 is the default hyperparameter used in NVIDIA apex. The extrapolation term $\hat{\eta}$ for EXTRAP-SGD and EXTRAP-ADAM is tuned by scaling the $\frac{\eta}{K}$ with the factor searched from $\{1, 2, 4\}$; we perform extensive hyperparameter tuning for the noise variants

³ Since we will fine-tune the (to be scaled) learning rate, there is no difference between learning rate linear scaling (Goyal et al., 2017) and square root scaling (Hoffer et al., 2017) in our case.

⁴ Our implementation relies on the PyTorch extension of NVIDIA apex for mixed precision and distributed training.

⁵ We follow an instruction from NVIDIA.

⁶ Our code is included in the submission for reproducibility.

of the extrapolated SGD, where the scaling factor of $\frac{\eta}{K}$ is searched from $\{0.1, 0.25, 0.5, 1, 2, 4\}$. The tuning procedure of the hyperparameters ensures that the best hyperparameter lies in the middle of our search grids; otherwise we extend our search grid.

12.4 Algorithmic Details

12.4.1 EXTRAP-SGD with Post-local SGD

We combine EXTRAP-SGD with post-local SGD in Algorithm 12. We omit the extrapolation step in line 3 and line 8 when $t=0$. The original form of post-local SGD refers to [Lin et al. \(2020d\)](#).

Algorithm 12 EXTRAP-SGD integrated with post-local SGD.

Requires: learning rate η ; inner learning rate $\hat{\eta}$; momentum factor u ; initial parameter \mathbf{x}_0 ; initial moment vector $\mathbf{v}_0 = 0$; time step $t = 0$; worker index k ; transition phase (iteration t_0) for post-local SGD; and local update steps H .

```

1: while  $\mathbf{x}_t$  not converged do
2:   if  $t \leq t_0$  then
3:      $\mathbf{x}_{t+\frac{1}{4}}^k = \mathbf{x}_t - \hat{\eta} \nabla f(\mathbf{x}_{t-\frac{1}{2}}^k)$ 
4:      $\mathbf{x}_{t+\frac{1}{2}}^k = \mathbf{x}_{t+\frac{1}{4}}^k + u\mathbf{v}_t$ 
5:      $\mathbf{v}_{t+1} = u\mathbf{v}_t - \frac{\eta}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k)$ 
6:      $\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{v}_{t+1}$ 
7:   else
8:      $\mathbf{x}_{t+\frac{1}{4}}^k = \mathbf{x}_t^k - \hat{\eta} \nabla f(\mathbf{x}_{t-\frac{1}{2}}^k)$ 
9:      $\mathbf{x}_{t+\frac{1}{2}}^k = \mathbf{x}_{t+\frac{1}{4}}^k + u\mathbf{v}_t^k$ 
10:     $\mathbf{v}_{t+1}^k = u\mathbf{v}_t^k - \eta \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k)$ 
11:     $\mathbf{x}_{t+1}^k = \mathbf{x}_t^k + \mathbf{v}_{t+1}^k$ 
12:    if  $t \bmod H = 0$  then
13:       $\mathbf{x}_{t+1}^k = \frac{1}{K} \sum_{k=1}^K \mathbf{x}_{t+1}^k$ 
14:  return  $\mathbf{x}_{t+1}$ 

```

12.4.2 Generalized EXTRAP-SGD with Adam

We present the generalized EXTRAP-SGD with Adam (EXTRAP-ADAM) in Algorithm 13. We omit the extrapolation step in line 2 when $t=0$.

12.5 Additional Results

12.5.1 ResNet-20 on CIFAR-10

The top-1 test accuracy for all related methods on CIFAR-10 is shown in Figure 12.1.

Chapter 12. Appendix for EXTRAP-SGD

Algorithm 13 EXTRAP-ADAM.

Requires: learning rate η ; inner learning rate $\hat{\eta}$; initial parameter \mathbf{x}_0 ; initial first-order moment vector $\mathbf{m}_0 = \mathbf{0}$; initial second-order moment vector $\mathbf{v}_0 = \mathbf{0}$; time step $t = 0$; worker index k .

```

1: while  $\mathbf{x}_t$  not converged do
2:    $\mathbf{x}_{t+\frac{1}{2}}^k = \mathbf{x}_t - \hat{\eta} \frac{\beta_1 \mathbf{m}_{t-1} + (1-\beta_1) \nabla f(\mathbf{x}_{t-\frac{1}{2}}^k)}{\sqrt{\beta_2 \mathbf{v}_{t-1} + (1-\beta_2) \nabla f(\mathbf{x}_{t-\frac{1}{2}}^k)^2 + \epsilon}}$ 
3:    $\mathbf{g}_t = \frac{1}{K} \sum_{k=1}^K \nabla f(\mathbf{x}_{t+\frac{1}{2}}^k)$ 
4:    $\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1) \mathbf{g}_t$ 
5:    $\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1-\beta_2) \mathbf{g}_t^2$ 
6:    $\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \frac{\mathbf{m}_t}{\sqrt{\mathbf{v}_t + \epsilon}}$ 
7: return  $\mathbf{x}_{t+1}$ 

```

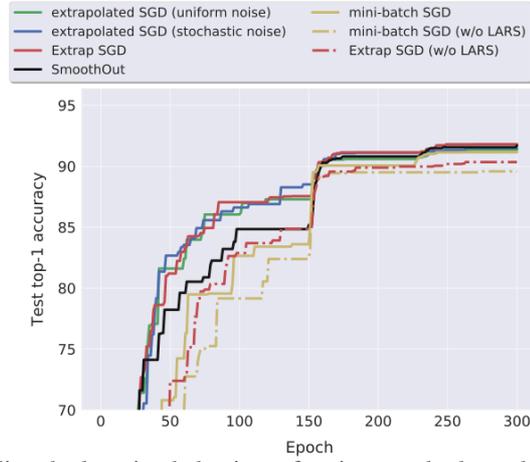


Figure 12.1 – Understanding the learning behaviors of various methods on the large-batch training (with mini-batch size 8,192 on 32 workers) for training ResNet-20 on CIFAR-10. The hyperparameters are fine-tuned, which are corresponding to the results shown in Table 3.1; by default the learning rate is decayed by 10 at epoch 150 and 225.

Table 12.1 – The test top-1 accuracy for ResNet-20 on CIFAR-10 under various combinations of local mini-batch size B and number of workers K . The global mini-batch size is always set to 8,192 and we vary K workers (MPI processes). We individually finetune η and $\hat{\eta}$ for each method on various setups, and the reported results are averaged over three seeds.

	$(B=512, K=16)$	$(B=256, K=32)$	$(B=128, K=64)$	$(B=64, K=128)$
Mini-batch SGD	91.35 ± 0.19	91.36 ± 0.19	91.29 ± 0.13	91.32 ± 0.17
EXTRAP-SGD	91.62 ± 0.32	91.72 ± 0.11	91.88 ± 0.27	91.89 ± 0.24

The effects of various combinations between local batch sizes and worker numbers are evaluated in Table 12.1.

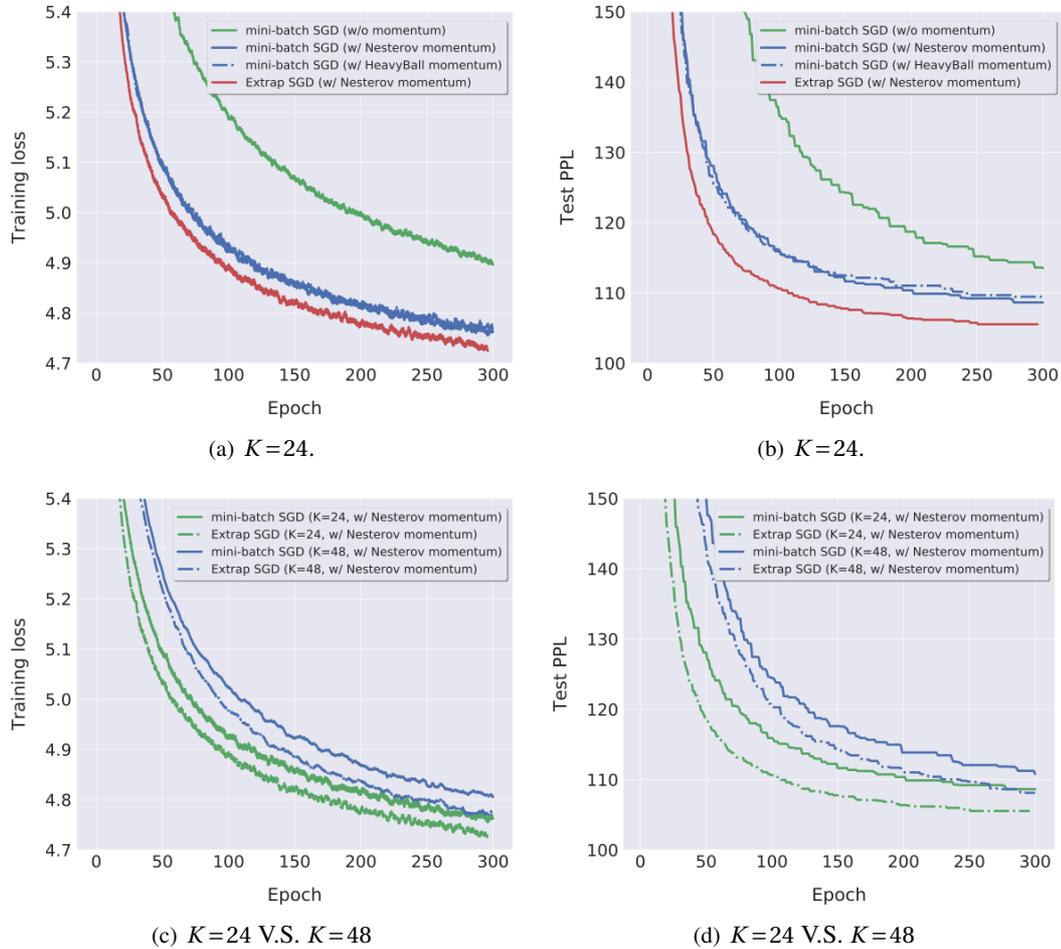


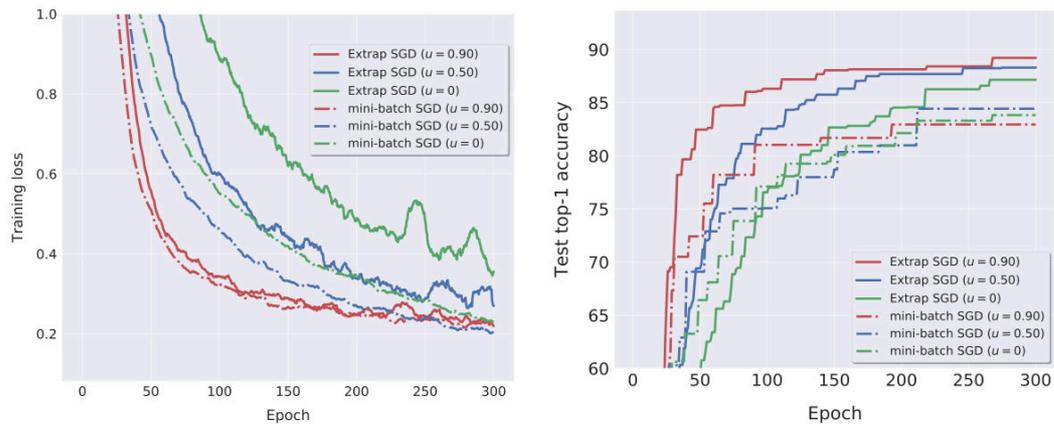
Figure 12.2 – The learning curves and perplexity (PPL, the lower the better) of training Wikitext-2 on LSTM. The global mini-batch size are 1536 and 3072 for $K=24$ and $K=48$ respectively, accounting for 2% and 4% of the total training data. We use the learning rate scaling and warmup in Goyal et al. (2017). We finetune the η for various variants of mini-batch SGD and EXTRAP-SGD have no additional tuning. The results of the inline table are averaged over three seeds.

12.5.2 LSTM on WikiText2

The learning curves of EXTRAP-SGD and mini-batch SGD with LSTM model on WikiText2 dataset are presented in Figure 12.2.

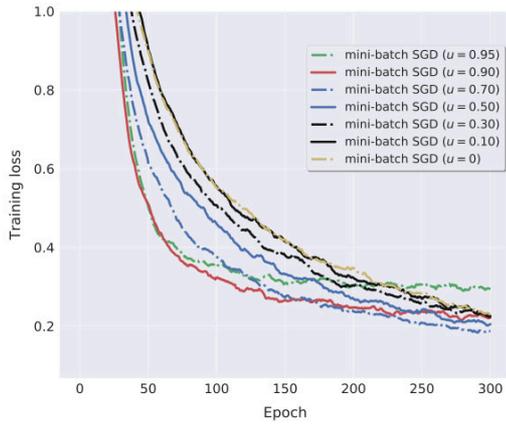
12.5.3 Impact of Momentum Factors

The impacts of momentum factors on EXTRAP-SGD and mini-batch SGD are demonstrated in Figure 12.3 and Figure 12.4.

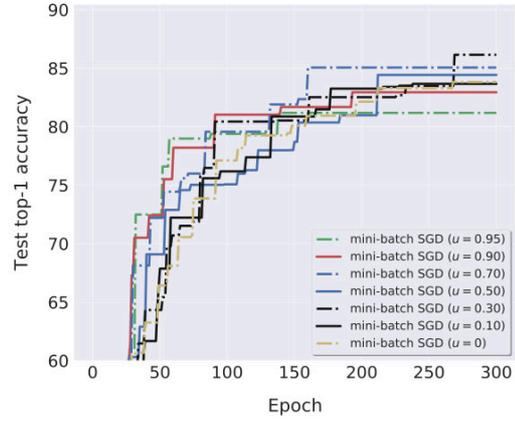


(a) The training loss of EXTRAP-SGD v.s. Mini-batch SGD. (b) The test top-1 accuracy of EXTRAP-SGD v.s. Mini-batch SGD.

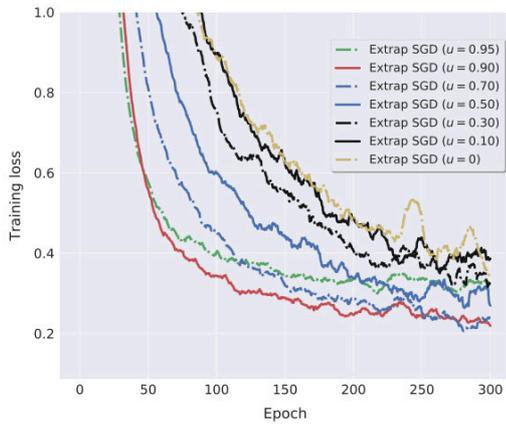
Figure 12.3 – Understanding the behaviors of EXTRAP-SGD and mini-batch SGD for various momentum factors u . The curves are evaluated on ResNet-20 with CIFAR-10 for mini-batch size 8,192; we use constant learning rate η and the LARS trust term $\hat{\eta}$ over the whole training procedure, for both of mini-batch SGD and EXTRAP-SGD (with default extrapolation term $\hat{\eta}$). The value of η and $\hat{\eta}$ correspond to the tuned optimal value in Table 3.1.



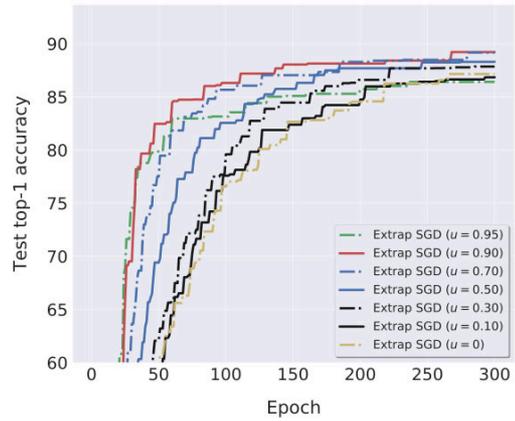
(a) The training loss of Mini-batch SGD.



(b) The test top-1 accuracy of Mini-batch SGD.



(c) The training loss of EXTRAP-SGD.



(d) The test top-1 accuracy of EXTRAP-SGD.

Figure 12.4 – The performance of EXTRAP-SGD and mini-batch SGD for various momentum factors u . The curves are evaluated on ResNet-20 with CIFAR-10 for mini-batch size 8,192; we use constant learning rate η and the LARS trust term $\tilde{\eta}$ over the whole training procedure, for both of mini-batch SGD and EXTRAP-SGD (with default extrapolation term $\hat{\eta}$). The value of η and $\tilde{\eta}$ correspond to the tuned optimal value in Table 3.1.

13 Appendix for CHOCO-SGD

13.1 Convergence of CHOCO-SGD

In this section we present the proof of Theorem 4.5.1. For this, we will first derive a slightly more general statement: in Theorem 13.1.3 we analyze CHOCO-SGD for arbitrary stepsizes η , and then derive Theorem 4.5.1 as a special case.

The structure of the proof follows [Koloskova et al. \(2019\)](#). That is, we first show that Algorithm 2 is a special case of a more general class of algorithms (given in Algorithm 14): Observe that Algorithm 2 consists of two main components: ② the stochastic gradient update, performed locally on each node, and ① the (quantized) averaging among the nodes. We can show convergence of all algorithms of this type—i.e. stochastic gradient updates ② followed by an arbitrary averaging step ①—as long as the averaging scheme exhibits linear convergence. For the specific averaging used in CHOCO-SGD, linear convergence has been shown in ([Koloskova et al., 2019](#)) and we will use their estimate of the convergence rate of the averaging scheme.

13.1.1 A General Framework for Decentralized SGD with Arbitrary Averaging

For convenience, we use the following matrix notation in this subsection.

$$\begin{aligned} X^{(t)} &:= [\mathbf{x}_1^{(t)}, \dots, \mathbf{x}_n^{(t)}] \in \mathbb{R}^{d \times n}, & \bar{X}^{(t)} &:= [\bar{\mathbf{x}}^{(t)}, \dots, \bar{\mathbf{x}}^{(t)}] \in \mathbb{R}^{d \times n}, \\ \partial F(X^{(t)}, \xi^{(t)}) &:= [\nabla F_1(\mathbf{x}_1^{(t)}, \xi_1^{(t)}), \dots, \nabla F_n(\mathbf{x}_n^{(t)}, \xi_n^{(t)})] \in \mathbb{R}^{d \times n}. \end{aligned}$$

Decentralized SGD with arbitrary averaging is given in Algorithm 14.

Assumption 10. For an averaging scheme $h: \mathbb{R}^{d \times n} \times \mathbb{R}^{d \times n} \rightarrow \mathbb{R}^{d \times n} \times \mathbb{R}^{d \times n}$ let $(X^+, Y^+) := h(X, Y)$ for $X, Y \in \mathbb{R}^{d \times n}$. Assume that h preserves the average of iterates:

$$X^+ \frac{\mathbf{1}\mathbf{1}^\top}{n} = X \frac{\mathbf{1}\mathbf{1}^\top}{n}, \quad \forall X, Y \in \mathbb{R}^{d \times n}, \quad (13.1)$$

Chapter 13. Appendix for CHOCO-SGD

Algorithm 14 Decentralized SGD with arbitrary averaging scheme.

Requires: $X^{(0)} = [\mathbf{x}^{(0)}, \dots, \mathbf{x}^{(0)}]$; stepsize η ; averaging function $h: \mathbb{R}^{d \times n} \times \mathbb{R}^{d \times n} \rightarrow \mathbb{R}^{d \times n} \times \mathbb{R}^{d \times n}$; initialize $Y^{(0)} = 0$.

-
- 1: **for** $t \in \{0, \dots, T-1\}$ **do**
 - 2: $X^{(t+\frac{1}{2})} = X^{(t)} - \eta \partial F(X^{(t)}, \xi^{(t)})$ \triangleright stochastic gradient updates
 - 3: $(X^{(t+1)}, Y^{(t+1)}) = h(X^{(t+\frac{1}{2})}, Y^{(t)})$ \triangleright blackbox averaging/gossip
 - 4: **return** $\mathbf{X}^{(T-1)}$
-

and that it converges with linear rate for a parameter $0 < c \leq 1$

$$\mathbb{E}_h \Psi(X^+, Y^+) \leq (1 - c) \Psi(X, Y), \quad \forall X, Y \in \mathbb{R}^{d \times n}, \quad (13.2)$$

and Lyapunov function $\Psi(X, Y) := \|X - \bar{X}\|_F^2 + \|X - Y\|_F^2$ with $\bar{X} := \frac{1}{n} X \mathbf{1} \mathbf{1}^\top$, where \mathbb{E}_h denotes the expectation over internal randomness of averaging scheme h .

Example: Exact Averaging. Setting $X^+ = XW$ and $Y^+ = X^+$ gives an exact consensus averaging algorithm with mixing matrix W (Xiao and Boyd, 2004). It converges at the rate $c = \rho$, where ρ is an eigengap of mixing matrix W , defined in Assumption 4. Substituting it into the Algorithm 14 we recover D-PSGD algorithm, analyzed in Lian et al. (2017).

Example: CHOCO-SGD. To recover CHOCO-SGD, we need to choose CHOCO-GOSSIP (Koloskova et al., 2019) as consensus averaging scheme, which is defined as $X^+ = X + \gamma Y(W - I)$ and $Y^+ = Y + Q(X^+ - Y)$ (in the main text we write \hat{X} instead of Y). This scheme converges with $c = \frac{\rho^2 \delta}{82}$. The results from the main part can be recovered by substituting this $c = \frac{\rho^2 \delta}{82}$ in the more general results below. It is important to note that for Algorithm 2 given in the main text, the order of the communication part ① and the gradient computation part ② is exchanged. We did this to better illustrate that both these parts are independent and that they can be executed in parallel. The effect of this change can be captured by changing the initial values but does not affect the convergence rate.

13.1.2 Proofs

Remark 13.1.1 (Mini-batch variance). *If for functions f_i, F_i defined in (4.1) Assumption 5 holds, i.e. $\mathbb{E}_\xi \|\nabla F_i(\mathbf{x}, \xi) - \nabla f_i(\mathbf{x})\|^2 \leq \sigma_i^2, i \in [n]$, then*

$$\mathbb{E}_{\xi_1^{(t)}, \dots, \xi_n^{(t)}} \left\| \frac{1}{n} \sum_{i=1}^n \left(\nabla f_i(\mathbf{x}_i^{(t)}) - \nabla F_i(\mathbf{x}_i^{(t)}, \xi_i^{(t)}) \right) \right\|^2 \leq \frac{\bar{\sigma}^2}{n}, \quad (13.3)$$

where $\bar{\sigma}^2 = \frac{\sum_{i=1}^n \sigma_i^2}{n}$.

Proof. This follows from

$$\mathbb{E} \left\| \frac{1}{n} \sum_{i=1}^n Y_i \right\|^2 = \frac{1}{n^2} \left(\sum_{i=1}^n \mathbb{E} \|Y_i\|^2 + \sum_{i \neq j} \mathbb{E} \langle Y_i, Y_j \rangle \right) = \frac{1}{n^2} \sum_{i=1}^n \mathbb{E} \|Y_i\|^2 \leq \frac{1}{n^2} \sum_{i=1}^n \sigma_i^2 = \frac{\bar{\sigma}^2}{n}$$

for $Y_i = f_i(\mathbf{x}_i^{(t)}) - \nabla F_i(\mathbf{x}_i^{(t)}, \xi_i^{(t)})$. Expectation of scalar product is equal to zero because ξ_i is independent of ξ_j since $i \neq j$. \square

Lemma 13.1.2. *Under Assumptions 4–10 the iterates of the Algorithm 14 with constant stepsize η satisfy*

$$\sum_{i=1}^n \|\bar{\mathbf{x}}^{(t)} - \mathbf{x}_i^{(t)}\|_2^2 \leq \eta^2 \frac{12nG^2}{c^2}.$$

Proof of Lemma 13.1.2. We start by following the proof of Lemma 21 from [Koloskova et al. \(2019\)](#). Define $r_t = \mathbb{E} \|X^{(t)} - \bar{X}^{(t)}\|^2 + \mathbb{E} \|X^{(t)} - Y^{(t)}\|^2$,

$$\begin{aligned} r_{t+1} &\stackrel{(13.2)}{\leq} (1-c) \mathbb{E} \left\| \bar{X}^{(t+\frac{1}{2})} - X^{(t+\frac{1}{2})} \right\|_F^2 + (1-c) \mathbb{E} \left\| Y^{(t)} - X^{(t+\frac{1}{2})} \right\|_F^2 \\ &= (1-c) \mathbb{E} \left\| \bar{X}^{(t)} - X^{(t)} + \eta \partial F(X^{(t)}, \xi^{(t)}) \left(\frac{\mathbf{1}\mathbf{1}^\top}{n} - I \right) \right\|_F^2 \\ &\quad + (1-c) \mathbb{E} \left\| Y^{(t)} - X^{(t)} + \eta \partial F(X^{(t)}, \xi^{(t)}) \right\|_F^2 \\ &\stackrel{(13.6)}{\leq} (1-c)(1+\alpha^{-1}) \mathbb{E} \left(\left\| \bar{X}^{(t)} - X^{(t)} \right\|_F^2 + \left\| Y^{(t)} - X^{(t)} \right\|_F^2 \right) \\ &\quad + (1-c)(1+\alpha) \eta^2 \mathbb{E} \left(\left\| \partial F(X^{(t)}, \xi^{(t)}) \left(\frac{\mathbf{1}\mathbf{1}^\top}{n} - I \right) \right\|_F^2 + \left\| \partial F(X^{(t)}, \xi^{(t)}) \right\|_F^2 \right) \\ &\leq (1-c) \left((1+\alpha^{-1}) \mathbb{E} \left(\left\| \bar{X}^{(t)} - X^{(t)} \right\|_F^2 + \left\| Y^{(t)} - X^{(t)} \right\|_F^2 \right) + 2n(1+\alpha) \eta^2 G^2 \right) \\ &\stackrel{\alpha=2}{\leq} \left(1 - \frac{c}{2} \right) \mathbb{E} \left(\left\| \bar{X}^{(t)} - X^{(t)} \right\|_F^2 + \left\| Y^{(t)} - X^{(t)} \right\|_F^2 \right) + \frac{6n}{c} \eta^2 G^2. \end{aligned}$$

Define $A = 3nG^2$, we got a recursion

$$r_{t+1} \leq \left(1 - \frac{c}{2} \right) r_t + \frac{2}{c} \eta^2 A,$$

Verifying that $r_t \leq \eta^2 \frac{4A}{c^2}$ satisfy recursion completes the proof as $\mathbb{E} \|X^{(t)} - \bar{X}^{(t)}\|^2 \leq r_t$.

Indeed, $r_0 = 0 \leq \eta^2 \frac{4A}{c^2}$ as $X^{(0)} = \bar{X}^{(0)}$ and $Y^{(0)} = 0$

$$r_{t+1} \leq \left(1 - \frac{c}{2} \right) r_t + \eta^2 \frac{2A}{c} \leq \left(1 - \frac{c}{2} \right) \eta^2 \frac{4A}{c^2} + \eta^2 \frac{2A}{c} = \eta^2 \frac{4A}{c^2}. \quad \square$$

Theorem 13.1.3. *Under Assumptions 4–10 with constant stepsize $\eta < \frac{1}{4L}$, the averaged iterates*

Chapter 13. Appendix for CHOCO-SGD

$\bar{\mathbf{x}}^{(t)} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i^{(t)}$ of Algorithm 14 satisfy:

$$\frac{1}{T+1} \sum_{t=0}^T \left\| \nabla f(\bar{\mathbf{x}}^{(t)}) \right\|_2^2 \leq \frac{4}{\eta(T+1)} \left(f(\bar{\mathbf{x}}^{(0)}) - f^* \right) + \eta \frac{2\bar{\sigma}^2 L}{n} + \eta^2 \frac{36G^2 L^2}{c^2}$$

where c denotes convergence rate of underlying averaging scheme.

Proof of Theorem 13.1.3. By L -smoothness

$$\begin{aligned} \mathbb{E}_{t+1} f(\bar{\mathbf{x}}^{(t+1)}) &= \mathbb{E}_{t+1} f\left(\bar{\mathbf{x}}^{(t)} - \frac{\eta}{n} \sum_{i=1}^n \nabla F_i(\mathbf{x}_i^{(t)}, \xi_i^{(t)})\right) \\ &\leq f(\bar{\mathbf{x}}^{(t)}) - \underbrace{\mathbb{E}_{t+1} \left\langle \nabla f(\bar{\mathbf{x}}^{(t)}), \frac{\eta}{n} \sum_{i=1}^n \nabla F_i(\mathbf{x}_i^{(t)}, \xi_i^{(t)}) \right\rangle}_{=: T_1} + \underbrace{\mathbb{E}_{t+1} \frac{L}{2} \eta^2 \left\| \frac{1}{n} \sum_{i=1}^n \nabla F_i(\mathbf{x}_i^{(t)}, \xi_i^{(t)}) \right\|_2^2}_{=: T_2} \end{aligned}$$

To estimate the second term, we add and subtract $\nabla f(\bar{\mathbf{x}}^{(t)})$

$$\begin{aligned} T_1 &= -\eta \left\| \nabla f(\bar{\mathbf{x}}^{(t)}) \right\|^2 + \eta \left\langle \nabla f(\bar{\mathbf{x}}^{(t)}), \nabla f(\bar{\mathbf{x}}^{(t)}) - \frac{1}{n} \sum_{i=1}^n \nabla f_i(\mathbf{x}_i^{(t)}) \right\rangle \\ &\stackrel{(13.5), \gamma=1}{\leq} -\frac{\eta}{2} \left\| \nabla f(\bar{\mathbf{x}}^{(t)}) \right\|^2 + \frac{\eta}{2n} \sum_{i=1}^n \left\| \nabla f(\bar{\mathbf{x}}^{(t)}) - \nabla f_i(\mathbf{x}_i^{(t)}) \right\|^2 \end{aligned}$$

For the last term, we add and subtract $\nabla f(\bar{\mathbf{x}}^{(t)})$ and the sum of $\nabla f_i(\mathbf{x}_i^{(t)})$

$$\begin{aligned} T_2 &= \mathbb{E}_{t+1} \left\| \frac{1}{n} \sum_{i=1}^n \left(\nabla F_i(\mathbf{x}_i^{(t)}, \xi_i^{(t)}) - \nabla f_i(\mathbf{x}_i^{(t)}) \right) \right\|_2^2 + \left\| \frac{1}{n} \sum_{i=1}^n \nabla f_i(\mathbf{x}_i^{(t)}) \pm \nabla f(\bar{\mathbf{x}}^{(t)}) \right\|_2^2 \\ &\stackrel{(13.3), (13.6), (13.4)}{\leq} \frac{\bar{\sigma}^2}{n} + \frac{2}{n} \sum_{i=1}^n \left\| \nabla f(\bar{\mathbf{x}}^{(t)}) - \nabla f_i(\mathbf{x}_i^{(t)}) \right\|_2^2 + 2 \left\| \nabla f(\bar{\mathbf{x}}^{(t)}) \right\|^2 \end{aligned}$$

Combining this together and using L -smoothness to estimate $\left\| \nabla f(\bar{\mathbf{x}}^{(t)}) - \nabla f_i(\mathbf{x}_i^{(t)}) \right\|_2^2$,

$$\mathbb{E}_{t+1} f(\bar{\mathbf{x}}^{(t+1)}) \leq f(\bar{\mathbf{x}}^{(t)}) - \eta \left(\frac{1}{2} - L\eta \right) \left\| \nabla f(\bar{\mathbf{x}}^{(t)}) \right\|_2^2 + \left(\frac{1}{2} \eta L^2 + \eta^2 L^3 \right) \frac{1}{n} \sum_{i=1}^n \left\| \bar{\mathbf{x}}^{(t)} - \mathbf{x}_i^{(t)} \right\|_2^2 + \frac{L\eta^2 \bar{\sigma}^2}{2n}.$$

Using Lemma 13.1.2 to bound the third term and using that $\eta \leq \frac{1}{4L}$ in the second and in the third terms

$$\mathbb{E}_{t+1} f(\bar{\mathbf{x}}^{(t+1)}) \leq f(\bar{\mathbf{x}}^{(t)}) - \frac{\eta}{4} \left\| \nabla f(\bar{\mathbf{x}}^{(t)}) \right\|_2^2 + \eta^3 \frac{9L^2 G^2}{c^2} + \eta^2 \frac{L\bar{\sigma}^2}{2n},$$

Rearranging terms and averaging over t

$$\begin{aligned} \frac{1}{T+1} \sum_{t=0}^T \left\| \nabla f(\bar{\mathbf{x}}^{(t)}) \right\|_2^2 &\stackrel{(13.6)}{\leq} \frac{4}{\eta} \frac{1}{T+1} \sum_{t=0}^T \left(\mathbb{E} f(\bar{\mathbf{x}}^{(t)}) - \mathbb{E} f(\bar{\mathbf{x}}^{(t+1)}) \right) + \eta^2 \frac{36G^2L^2}{c^2} + \eta \frac{2L\bar{\sigma}^2}{n} \\ &\leq \frac{4}{\eta(T+1)} \left(f(\bar{\mathbf{x}}^{(0)}) - f^* \right) + \eta \frac{2\bar{\sigma}^2L}{n} + \eta^2 \frac{36G^2L^2}{c^2} \end{aligned}$$

□

Corollary 13.1.4. *Under Assumptions 4–10 with constant stepsize $\eta = \sqrt{\frac{n}{T+1}}$ for $T \geq 16nL^2$, the averaged iterates $\bar{\mathbf{x}}^{(t)} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i^{(t)}$ of Algorithm 14 satisfy:*

$$\frac{1}{T+1} \sum_{t=0}^T \left\| \nabla f(\bar{\mathbf{x}}^{(t)}) \right\|_2^2 \leq \frac{4 \left(f(\bar{\mathbf{x}}^{(0)}) - f^* \right) + 2\bar{\sigma}^2L}{\sqrt{n(T+1)}} + \frac{36G^2nL^2}{(T+1)c^2}$$

where c denotes convergence rate of underlying averaging scheme.

The first term shows a linear speed up compared to SGD on one node, whereas the underlying averaging scheme affects only the second-order term. Substituting the convergence rate for exact averaging with W gives the rate $\mathcal{O}(1/\sqrt{nT} + n/(T\rho^2))$.

CHOCO-SGD with the underlying CHOCO-GOSSIP averaging scheme converges at the rate $\mathcal{O}(1/\sqrt{nT} + n/(T\rho^4\delta^2))$. The dependence on ρ (eigengap of the mixing matrix W) is worse than in the exact case. This might either just be an artifact of our proof technique or a consequence of supporting arbitrary high compression.

13.1.3 Convergence for Arbitrary T

The previous result holds only for T larger than $16nL^2$. This is not necessary and can be relaxed by carefully tuning the stepsize.

Lemma 13.1.5. *For any parameters $r_0 \geq 0, b \geq 0, e \geq 0, d \geq 0$ there exists constant stepsize $\eta \leq \frac{1}{d}$ such that*

$$\Psi_T := \frac{r_0}{\eta(T+1)} + b\eta + e\eta^2 \leq 2 \left(\frac{br_0}{T+1} \right)^{\frac{1}{2}} + 2e^{1/3} \left(\frac{r_0}{T+1} \right)^{\frac{2}{3}} + \frac{dr_0}{T+1}$$

Proof. Choosing $\eta = \min \left\{ \left(\frac{r_0}{b(T+1)} \right)^{\frac{1}{2}}, \left(\frac{r_0}{e(T+1)} \right)^{\frac{1}{3}}, \frac{1}{d} \right\} \leq \frac{1}{d}$ we have three cases

- $\eta = \frac{1}{d}$ and is smaller than both $\left(\frac{r_0}{b(T+1)} \right)^{\frac{1}{2}}$ and $\left(\frac{r_0}{e(T+1)} \right)^{\frac{1}{3}}$, then

$$\Psi_T \leq \frac{dr_0}{T+1} + \frac{b}{d} + \frac{e}{d^2} \leq \left(\frac{br_0}{T+1} \right)^{\frac{1}{2}} + \frac{dr_0}{T+1} + e^{1/3} \left(\frac{r_0}{T+1} \right)^{\frac{2}{3}}$$

- $\eta = \left(\frac{r_0}{b(T+1)}\right)^{\frac{1}{2}} < \left(\frac{r_0}{e(T+1)}\right)^{\frac{1}{3}}$, then

$$\Psi_T \leq 2 \left(\frac{r_0 b}{T+1}\right)^{\frac{1}{2}} + e \left(\frac{r_0}{b(T+1)}\right) \leq 2 \left(\frac{r_0 b}{T+1}\right)^{\frac{1}{2}} + e^{\frac{1}{3}} \left(\frac{r_0}{T+1}\right)^{\frac{2}{3}},$$

- The last case, $\eta = \left(\frac{r_0}{e(T+1)}\right)^{\frac{1}{3}} < \left(\frac{r_0}{b(T+1)}\right)^{\frac{1}{2}}$

$$\Psi_T \leq 2e^{\frac{1}{3}} \left(\frac{r_0}{T+1}\right)^{\frac{2}{3}} + b \left(\frac{r_0}{e(T+1)}\right)^{\frac{1}{3}} \leq 2e^{\frac{1}{3}} \left(\frac{r_0}{T+1}\right)^{\frac{2}{3}} + \left(\frac{br_0}{T+1}\right)^{\frac{1}{2}}$$

□

Corollary 13.1.6 (Generalized Theorem 4.5.1). *Under Assumptions 4–10 with constant stepsize η tuned as in Lemma 13.1.5, the averaged iterates $\bar{\mathbf{x}}^{(t)} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i^{(t)}$ of Algorithm 14 satisfy:*

$$\frac{1}{T+1} \sum_{t=0}^T \left\| \nabla f(\bar{\mathbf{x}}^{(t)}) \right\|_2^2 \leq 4 \sqrt{\frac{2L\bar{\sigma}^2}{n(T+1)}} + 17 \left(\frac{GLF_0}{c(T+1)}\right)^{\frac{2}{3}} + \frac{16LF_0}{T+1}$$

where c denotes convergence rate of underlying averaging scheme, $F_0 = f(\bar{\mathbf{x}}^{(0)}) - f^*$.

Proof. The result follows from Theorem 13.1.3 and Lemma 13.1.5 with $r_0 = 4(f(\bar{\mathbf{x}}^{(0)}) - f^*)$, $b = \frac{2\bar{\sigma}^2 L}{n}$, $e = \frac{36G^2 L^2}{c^2}$ and $d = 4L$. □

The corollary gives guarantees for the averaged vector of parameters $\bar{\mathbf{x}}$, however in a decentralized setting it is very expensive and sometimes impossible to average all the parameters distributed across several machines, especially when the number of machines and the model size is large. We can get similar guarantees on the individual iterates \mathbf{x}_i as e.g. in (Assran et al., 2019). We summarize these briefly below.

Corollary 13.1.7 (Convergence of local weights). *Under the same setting as in Corollary 13.1.6,*

$$\frac{1}{T+1} \sum_{t=0}^T \frac{1}{n} \sum_{i=1}^n \left\| \nabla f(\mathbf{x}_i^{(t)}) \right\|_2^2 \leq 8 \sqrt{\frac{2L\bar{\sigma}^2}{n(T+1)}} + 37 \left(\frac{GLF_0}{c(T+1)}\right)^{\frac{2}{3}} + \frac{32LF_0}{T+1}$$

Proof of Corollary 13.1.7.

$$\begin{aligned} \frac{1}{T+1} \sum_{t=0}^T \frac{1}{n} \sum_{i=1}^n \left\| \nabla f(\mathbf{x}_i^{(t)}) \right\|_2^2 &\leq \frac{1}{T+1} \sum_{t=0}^T \frac{1}{n} \sum_{i=1}^n \left(2 \left\| \nabla f(\mathbf{x}_i^{(t)}) - \nabla f(\bar{\mathbf{x}}^{(t)}) \right\|_2^2 + 2 \left\| \nabla f(\bar{\mathbf{x}}^{(t)}) \right\|_2^2 \right) \\ &\leq \frac{1}{T+1} \sum_{t=0}^T \frac{1}{n} \sum_{i=1}^n \left(2L^2 \left\| \mathbf{x}_i^{(t)} - \bar{\mathbf{x}}^{(t)} \right\|_2^2 + 2 \left\| \nabla f(\bar{\mathbf{x}}^{(t)}) \right\|_2^2 \right) \end{aligned}$$

where we used L -smoothness of f . Using Theorem 13.1.3 and tuning the stepsize as in Lemma 13.1.5 we get the statement of the corollary. □

13.2 Useful Inequalities

Lemma 13.2.1. For arbitrary set of n vectors $\{\mathbf{a}_i\}_{i=1}^n$, $\mathbf{a}_i \in \mathbb{R}^d$

$$\left\| \sum_{i=1}^n \mathbf{a}_i \right\|^2 \leq n \sum_{i=1}^n \|\mathbf{a}_i\|^2. \quad (13.4)$$

Lemma 13.2.2. For given two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$

$$2 \langle \mathbf{a}, \mathbf{b} \rangle \leq \gamma \|\mathbf{a}\|^2 + \gamma^{-1} \|\mathbf{b}\|^2, \quad \forall \gamma > 0. \quad (13.5)$$

Lemma 13.2.3. For given two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$

$$\|\mathbf{a} + \mathbf{b}\|^2 \leq (1 + \alpha) \|\mathbf{a}\|^2 + (1 + \alpha^{-1}) \|\mathbf{b}\|^2, \quad \forall \alpha > 0. \quad (13.6)$$

This inequality also holds for the sum of two matrices $A, B \in \mathbb{R}^{n \times d}$ in Frobenius norm.

13.3 Compression Schemes

We implement the compression schemes detailed below.

- gsgd_b (Alistarh et al., 2017). The unbiased $\text{gsgd}_b: \mathbb{R}^d \rightarrow \mathbb{R}^d$ compression operator (for $b > 1$) is given as

$$\text{gsgd}_b(\mathbf{x}) := \|\mathbf{x}\|_2 \cdot \text{sig}(\mathbf{x}) \cdot 2^{-(b-1)} \cdot \left\lfloor \frac{2^{(b-1)} |\mathbf{x}|}{\|\mathbf{x}\|_2} + \mathbf{u} \right\rfloor$$

where $\mathbf{u} \sim_{u.a.r.} [0, 1]^d$ is a random dithering vector and $\text{sig}(\mathbf{x})$ assigns the element-wise sign: $(\text{sig}(\mathbf{x}))_i = 1$ if $(\mathbf{x})_i \geq 0$ and $(\text{sig}(\mathbf{x}))_i = -1$ if $(\mathbf{x})_i < 0$. As the value in the right bracket will be rounded to an integer in $\{0, \dots, 2^{(b-1)} - 1\}$, each coordinate can be encoded with at most $(b-1) + 1$ bits (1 for the sign). For more efficient encoding schemes cf. Alistarh et al. (2017).

A biased version is given as

$$\text{gsgd}_b(\mathbf{x}) := \frac{\|\mathbf{x}\|_2}{\tau} \cdot \text{sig}(\mathbf{x}) \cdot 2^{-(b-1)} \cdot \left\lfloor \frac{2^{(b-1)} |\mathbf{x}|}{\|\mathbf{x}\|_2} + \mathbf{u} \right\rfloor$$

for $\tau = 1 + \min \left\{ \frac{d}{2^{2(b-1)}}, \frac{\sqrt{d}}{2^{(b-1)}} \right\}$ and is a $\delta = \frac{1}{\tau}$ compression operator (Koloskova et al., 2019).

- random_a (Wangni et al., 2018). Let $\mathbf{u} \in \{0, 1\}^d$ be a masking vector, sampled uniformly at random from the set $\{\mathbf{u} \in \{0, 1\}^d : \|\mathbf{u}\|_1 = \lfloor ad \rfloor\}$. Then the unbiased $\text{random}_a: \mathbb{R}^d \rightarrow \mathbb{R}^d$

operator is defined as

$$\text{random}_a(\mathbf{x}) := \frac{d}{\lfloor ad \rfloor} \cdot \mathbf{x} \odot \mathbf{u}.$$

The biased version is given as

$$\text{random}_a(\mathbf{x}) := \mathbf{x} \odot \mathbf{u},$$

and is a $\delta = a$ compression operator (Stich et al., 2018).

Only $32\lfloor ad \rfloor$ bits are required to send $\text{random}_a(\mathbf{x})$ to another node—all the values of non-zero entries (we assume that entries are represented as `float32` numbers). Receiver can recover positions of these entries if it knows the random seed of uniform sampling operator used to select these entries. This random seed could be communicated once on preprocessing stage (before starting the algorithm).

- top_a (Alistarh et al., 2018; Stich et al., 2018). The biased $\text{top}_a: \mathbb{R}^d \rightarrow \mathbb{R}^d$ operator is defined as

$$\text{top}_a(\mathbf{x}) := \mathbf{x} \odot \mathbf{u}(\mathbf{x}),$$

where $\mathbf{u}(\mathbf{x}) \in \{0, 1\}^d$, $\|\mathbf{u}\|_1 = \lfloor ad \rfloor$ is a masking vector with $(\mathbf{u})_i = 1$ for indices $i \in \pi^{-1}(\{1, \dots, \lfloor ad \rfloor\})$ where the permutation π is such that $|(\mathbf{x})_{\pi(1)}| \geq |(\mathbf{x})_{\pi(2)}| \geq \dots \geq |(\mathbf{x})_{\pi(d)}|$. The top_a operator is a $\delta = a$ compression operator (Stich et al., 2018).

In the case of top_a compression $2 \cdot 32\lfloor ad \rfloor$ bits are required because along with the values we need to send positions of these values.

- sign (Bernstein et al., 2018; Karimireddy et al., 2019). The biased (scaled) $\text{sign}: \mathbb{R}^d \rightarrow \mathbb{R}$ compression operator is defined as

$$\text{sign}(\mathbf{x}) := \frac{\|\mathbf{x}\|_1}{d} \cdot \text{sgn}(\mathbf{x}).$$

The sign operator is a $\delta = \frac{\|\mathbf{x}\|_1^2}{d\|\mathbf{x}\|_2^2}$ compression operator (Karimireddy et al., 2019).

In total for the sign compression we need to send only $d + 32$ bits—one bit for every entry in \mathbf{x} and 32 bits for $\|\mathbf{x}\|_1$.

13.4 CHOCO-SGD with Momentum

Algorithm 3 demonstrates how to combine CHOCO-SGD with weight decay and momentum. Nesterov momentum can be analogously adapted for our decentralized setting.

13.5 Error Feedback Interpretation of CHOCO-SGD

To better understand how does CHOCO-SGD work, we can interpret it as an error feedback algorithm (Stich et al., 2018; Karimireddy et al., 2019; Stich and Karimireddy, 2020). We can equivalently rewrite CHOCO-SGD (Algorithm 2) as Algorithm 15. The common feature of error feedback algorithms is that quantization errors are saved into the internal memory, which is added to the compressed value at the next iteration. In CHOCO-SGD the value we want to transmit is the difference $\mathbf{x}_i^{(t)} - \mathbf{x}_i^{(t-1)}$, which represents the evolution of local variable \mathbf{x}_i at step t . Before compressing this value on line 4, the internal memory is added on line 3 to correct for the errors. Then, on line 5 internal memory is updated. Note that $\mathbf{m}_i^{(t)} = \mathbf{x}_i^{(t-1)} - \hat{\mathbf{x}}_i^{(t)}$ in the old notation.

Algorithm 15 Decentralized SGD with arbitrary averaging scheme.

Requires: Initial values $\mathbf{x}_i^{(0)} \in \mathbb{R}^d$ on each node $i \in [n]$; consensus stepsize γ ; SGD stepsize η ; communication graph $G = ([n], E)$ and mixing matrix W ; initialize $\hat{\mathbf{x}}_i^{(0)} = \mathbf{x}_i^{(-1)} := \mathbf{0}, \forall i \in [n]$.

```

1: procedure WORKER- $i$ 
2:   for  $t \in \{0, \dots, T-1\}$  do
3:      $\mathbf{x}_i^{(t)} := \mathbf{x}_i^{(t-\frac{1}{2})} + \eta \sum_{j: \{i, j\} \in E} w_{ij} (\hat{\mathbf{x}}_j^{(t)} - \hat{\mathbf{x}}_i^{(t)})$  ▷ modified gossip averaging
4:      $\mathbf{v}_i^{(t)} = \mathbf{x}_i^{(t)} - \mathbf{x}_i^{(t-1)} + \mathbf{m}_i^{(t)}$ 
5:      $\mathbf{q}_i^{(t)} := Q(\mathbf{v}_i^{(t)})$  ▷ compression
6:      $\mathbf{m}_i^{(t+1)} = \mathbf{v}_i^{(t)} - \mathbf{q}_i^{(t)}$  ▷ memory update
7:     for neighbors  $j: \{i, j\} \in E$  (including  $\{i\} \in E$ ) do
8:       Send  $\mathbf{q}_i^{(t)}$  and receive  $\mathbf{q}_j^{(t)}$  ▷ communication
9:        $\hat{\mathbf{x}}_j^{(t+1)} := \mathbf{q}_j^{(t)} + \hat{\mathbf{x}}_j^{(t)}$  ▷ local update
10:    Sample  $\xi_i^{(t)}$ , compute gradient  $\mathbf{g}_i^{(t)} := \nabla F_i(\mathbf{x}_i^{(t)}, \xi_i^{(t)})$ 
11:     $\mathbf{x}_i^{(t+\frac{1}{2})} := \mathbf{x}_i^{(t)} - \eta \mathbf{g}_i^{(t)}$  ▷ stochastic gradient update
12:  return  $\mathbf{x}_i^{(T-1+\frac{1}{2})}$ 

```

13.6 Detailed Experimental Setup and Tuned Hyperparameters

We precise the procedure of model training as well as the hyperparameter tuning in this section.

Social Network Setup. For the comparison we consider CHOCO-SGD with sign compression (this combination achieved the compromise between accuracy and compression level in Table 4.1), decentralized SGD without compression, and centralized SGD without compression. We train two models, firstly ResNet20 (He et al., 2016a) (0.27 million parameters) for image classification on the Cifar10 dataset (50K/10K training/test samples) (Krizhevsky and Hinton, 2009) and secondly, a three-layer LSTM architecture (Hochreiter and Schmidhuber, 1997b) (28.95 million parameters) for a language modeling task on WikiText-2 (600 training and 60 validation articles with a total of 2'088'628 and 217'646 tokens respectively) (Merity et al., 2017). For the

Chapter 13. Appendix for CHOCO-SGD

Table 13.1 – Tuned hyperparameters of CHOCO-SGD for training ResNet-20 on Cifar10, corresponding to the ring topology with 8 nodes in Table 4.1. We randomly split the training data between nodes and shuffle it after every epoch. The per node mini-batch size is 128 and the degree of each node is 3.

Compression schemes	Learning rate	Consensus stepsize
QSGD (16-bit)	1.60	0.2
QSGD (8-bit)	0.96	0.2
QSGD (4-bit)	1.60	0.075
QSGD (2-bit)	0.96	0.025
Sparsification (random-50%)	2.40	0.45
Sparsification (random-10%)	1.20	0.075
Sparsification (random-1%)	0.48	0.00625
Sparsification (top-50%)	1.60	0.45
Sparsification (top-10%)	1.60	0.15
Sparsification (top-1%)	1.20	0.0375
Sign+Norm	1.60	0.45

language modeling task, we borrowed and adapted the general experimental setup of [Merity et al. \(2018\)](#), where we use a three-layer LSTM with hidden dimension of size 650. The loss is averaged over all examples and timesteps. The BPTT length is set to 30. We fine-tune the value of gradient clipping (0.4), and the dropout (0.4) is only applied on the output of LSTM.

We train both of ResNet20 and LSTM for 300 epochs, unless mentioned specifically. The per node mini-batch size is 32 for both datasets. The momentum (with factor 0.9) is only applied on the ResNet20 training.

Social Network and a Datacenter details. For all algorithms, we gradually warmup ([Goyal et al., 2017](#)) the learning rate from a relative small value (0.1) to the fine-tuned initial learning rate for the first 5 training epochs. During the training procedure, the tuned initial learning rate is decayed by the factor of 10 when accessing 50% and 75% of the total training epochs. The learning rate is tuned by finding the optimal initial learning rate (after the scaling).

The optimal $\hat{\eta}$ is searched in a pre-defined grid and we ensure that the best performance was contained in the middle of the grids. For example, if the best performance was ever at one of the extremes of the grid, we would try new grid points. Same searching logic applies to the consensus stepsize.

Table 13.1 demonstrates the fine-tuned hyperparameters of CHOCO-SGD for training ResNet-20 on Cifar10, while Table 13.3 reports our fine-tuned hyperparameters of our baselines. Table 13.2 demonstrates the fine-tuned hyperparameters of CHOCO-SGD for training ResNet-20/LSTM on a social network topology.

We estimate the runtime information (depicted in Figure 4.5) of various methods from three trials of the evaluation on Google Cloud (Kubernetes Engine). More precisely, we create the cluster

Table 13.2 – Tuned hyperparameters of CHOCO-SGD, corresponding to the social network topology with 32 nodes in Table 4.3. We randomly split the training data between the nodes and keep this partition fixed during the entire training (no shuffling). The per node mini-batch size is 32 and the maximum degree of the node is 14.

Configuration	Learning rate	Consensus stepsize
ResNet-20, Cifar10, Sign+Norm	1.0	0.5
LSTM, WikiText-2, Sign+Norm	25	0.6

Table 13.3 – Tuned hyperparameters of DCD, ECD, and DeepSqueeze for training ResNet-20 on Cifar10, corresponding to the ring topology with 8 nodes in Table 4.1. We randomly split the training data between nodes and shuffle it after every epoch. The per node mini-batch size is 128 and the degree of each node is 3. We only report the hyperparameters corresponding to results that can reach to reasonable performance in our experiments.

Compression schemes	Learning rate	Consensus stepsize
DCD, QSGD (16-bit)	2.40	-
DCD, QSGD (8-bit)	1.20	-
DCD, Sparsification (random-50%)	0.80	-
DCD, Sparsification (top-50%)	1.20	-
DCD, Sparsification (top-10%)	1.60	-
DCD, Sparsification (top-1%)	2.40	-
ECD, QSGD (16-bit)	0.96	-
ECD, QSGD (8-bit)	1.20	-
DeepSqueeze, QSGD (4-bit)	0.60	0.01
DeepSqueeze, QSGD (2-bit)	0.80	0.005
DeepSqueeze, Sparsification (top-50%)	0.80	0.05
DeepSqueeze, Sparsification (top-10%)	0.60	0.01
DeepSqueeze, Sparsification (top-1%)	0.40	0.005
DeepSqueeze, Sparsification (random-1%)	0.80	0.0005
DeepSqueeze, Sign+Norm	0.48	0.01

on Google Cloud for three times and each time we estimate the time per mini-batch of various methods (through the first two training epochs).

13.7 Additional Plots

To complement our results for scaling to a large number of nodes, we here additionally depict the learning curves (e.g. test accuracy) for the training on 64 nodes in Figure 13.1. We also mark the levels used for Figure 4.1 (cf. Table 13.4 and Table 13.5).

We additionally visualize the learning curves for the social network topology in Figure 13.2 and Figure 13.3.

We additionally provide the learning curves of training top-1, top-5 accuracy and test top-5 accuracy for the datacenter experiment in Figure 13.4.

Chapter 13. Appendix for CHOCO-SGD

Table 13.4 – The exact epoch for the same bits budget in Figure 4.1.

	$n = 4$	$n = 16$	$n = 36$	$n = 64$
Centralized	5	6	6	6
Decentralized (Ring)	7	17	32	54
Decentralized (Torus)	6	10	18	29
CHOCO (Ring)	105	408	904	1588
CHOCO (Torus)	55	206	454	796

Table 13.5 – The exact transmitted bits (in MB) for the same epoch budget in Figure 4.1.

	$n = 4$	$n = 16$	$n = 36$	$n = 64$
Centralized	139683	140041	144299	142899
Decentralized (Ring)	69841	17505	8016	4554
Decentralized (Torus)	139683	35010	16033	9109
CHOCO (Ring)	2208	564	253	144
CHOCO (Torus)	4417	1129	506	288

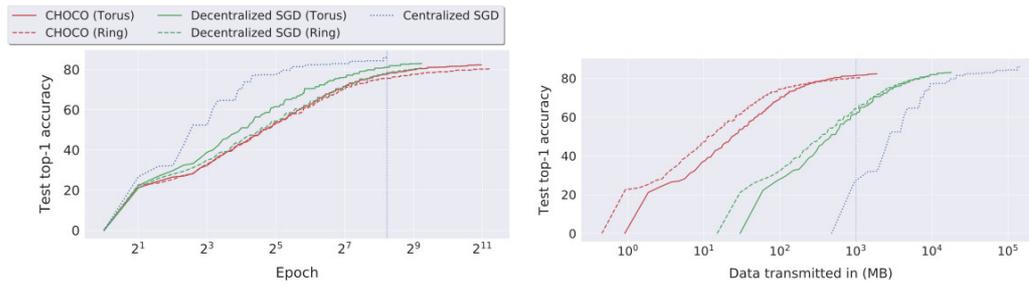
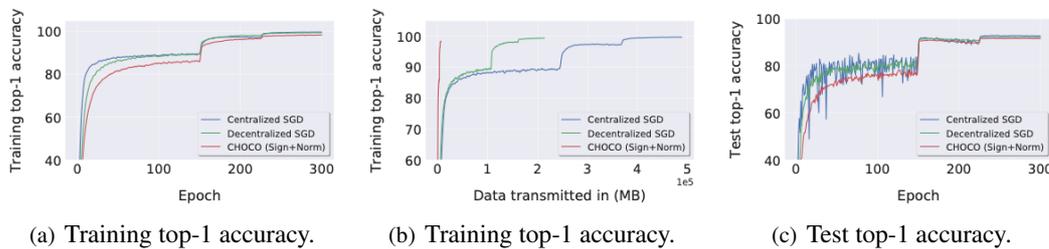


Figure 13.1 – Scaling of CHOCO-SGD with sign compression to large number of devices on Cifar10 dataset. Convergence curves for 64 nodes. Vertical lines corresponds to the epoch/bits budget used in Figure 4.1.



(a) Training top-1 accuracy.

(b) Training top-1 accuracy.

(c) Test top-1 accuracy.

Figure 13.2 – Training ResNet-20 on CIFAR-10 with decentralized algorithm on a real world social network topology. The topology has 32 nodes and we assume each node can only access a disjoint subset of the whole dataset. The local mini-batch size is 32.

13.7. Additional Plots

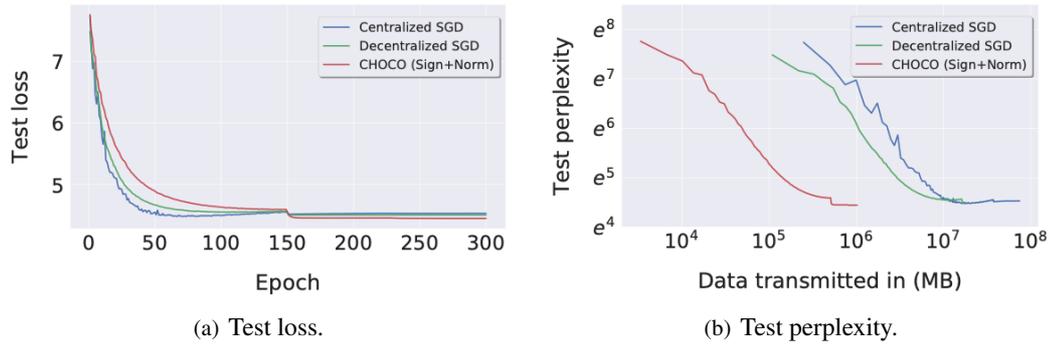


Figure 13.3 – Training LSTM on WikiText2 with decentralized algorithm on a real world social network topology. The topology has 32 nodes and we assume each node can only access a disjoint subset of the whole dataset. The local mini-batch size is 32.

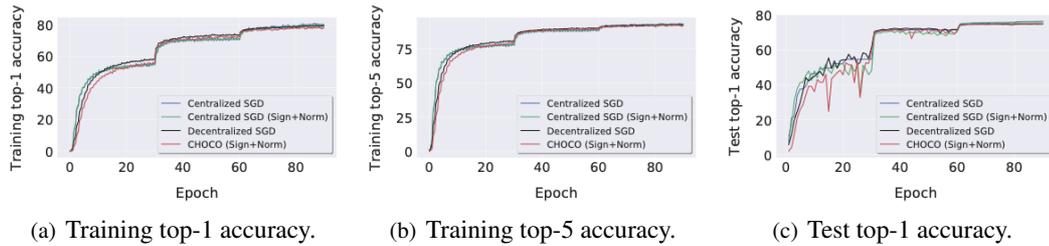


Figure 13.4 – Large-scale training: ResNet-50 on ImageNet in the datacenter.

14 Appendix for Consensus Control in Decentralized Learning

14.1 Efficient Implementation of Consensus Control for Data-Center Training

In our theoretical and experimental investigations in Sections 5.4 and 5.5, in order to understand the effect of decentralization on the final performance, we focused on the controlling the consensus distance Ξ_t^2 . This quantity was inspired by theoretical analysis and naturally measures the decentralization level. In practice, in order to control the consensus distance, one need to know the exact value of it at every iteration. Computing the exact value of the Ξ_t^2 requires all-to-all communications of the parameter vectors \mathbf{x}_i , which is costly and would cancel all the practical benefits of using decentralized algorithms.

In this section we give a more practical way to control the consensus distance without compromising the final test performance. We mainly focus on the data-center training scenario, the most common use case of large-scale deep learning training for both academic and industry. Though the prior arts use All-Reduce to compute the exactly averaged model parameters, recent trends show promising faster training results by using decentralized training with gossip averaging ([Asrhan et al., 2019](#); [Koloskova et al., 2020a](#)), especially for the highly over-parameterized SOTA neural networks with large number of model parameters.

14.1.1 Theoretical Justification

We upper bound the consensus distance Ξ_t^2 with a quantity that is efficiently computable in our scenario and control only this quantity. This quantity additionally requires the centralized all-reduce applied only to one dimensional numbers, that is fast to perform, and it utilizes the information available after decentralized communications step of parameters \mathbf{x}_i performed by the (D-SGD) algorithm. For simplicity, in this section we only analyze the case of the fixed topology, i.e. mixing matrix W is constant. Our result can be generalized for the randomized mixing matrix (Assumption 6) and in later sections we provide the proofs under the general Assumption 6.

Chapter 14. Appendix for Consensus Control in Decentralized Learning

Lemma 14.1.1 (Upper bound on the consensus distance). *Let $\Theta_t^2 := \frac{1}{n} \sum_{i=1}^n \left\| \sum_{j=1}^n w_{ij} \mathbf{x}_j^{(t)} - \mathbf{x}_i^{(t)} \right\|_2^2 = \frac{1}{n} \sum_{i=1}^n \theta_i^{(t)}$, where w_{ij} are the weights of the (fixed) mixing matrix W . We can upper bound the consensus distance as*

$$\Xi_t \leq \frac{2}{p} \Theta_t, \quad \forall \mathbf{x}_1^{(t)}, \dots, \mathbf{x}_n^{(t)} \in \mathbb{R}^d,$$

where p is consensus rate of matrix W (Assumption 6).

To ensure small consensus distance Ξ_t^2 it is sufficient to make small the quantity Θ_t^2 . In particular by ensuring that $\Theta_t^2 \leq \frac{p^2}{4} \eta_t^2$ we automatically get that CCD condition holds: $\Xi_t^2 \leq \eta_t^2$ (Proposition 5.4.2).

Practical way to compute Θ_t^2 . Recall that $\Theta_t^2 = \frac{1}{n} \sum_{i=1}^n \left\| \sum_{j=1}^n w_{ij} \mathbf{x}_j^{(t)} - \mathbf{x}_i^{(t)} \right\|_2^2$. Each term i , $i \in \{1, \dots, n\}$ of this sum is locally available to the node i after one round of decentralized communication with mixing matrix W because $w_{ij} \neq 0$ only for the neighbours j of the node i . So each node i can locally compute the norm $\left\| \sum_{j=1}^n w_{ij} \mathbf{x}_j^{(t)} - \mathbf{x}_i^{(t)} \right\|_2^2$ and then obtain the average Θ_t^2 using centralized all-reduce on only 1-dimensional numbers, that is much faster than averaging full vectors from \mathbb{R}^d .

Proof of the Lemma 14.1.1

Proof. Using matrix notation we can re-write $\Xi_t^2 = \frac{1}{n} \left\| \mathbf{X}^{(t)} - \bar{\mathbf{X}}^{(t)} \right\|_F^2$ and $\Theta_t^2 = \frac{1}{n} \left\| \mathbf{X}^{(t)} \mathbf{W} - \mathbf{X}^{(t)} \right\|_F^2$.

Since $\bar{\mathbf{X}}^{(t)} \mathbf{W} = \bar{\mathbf{X}}^{(t)}$ and $\mathbf{X}^{(t)} \frac{\mathbf{1}\mathbf{1}^\top}{n} = \bar{\mathbf{X}}^{(t)} \frac{\mathbf{1}\mathbf{1}^\top}{n} = \bar{\mathbf{X}}^{(t)}$ we have that

$$\mathbf{X}^{(t)} \mathbf{W} - \mathbf{X}^{(t)} = (\mathbf{X}^{(t)} - \bar{\mathbf{X}}^{(t)}) \left(\mathbf{W} - \frac{\mathbf{1}\mathbf{1}^\top}{n} - \mathbf{I} \right)$$

Using Frobenius norm property (14.1),

$$\left\| \mathbf{X}^{(t)} \mathbf{W} - \mathbf{X}^{(t)} \right\|_F \geq \lambda_{\min} \left(\mathbf{W} - \frac{\mathbf{1}\mathbf{1}^\top}{n} - \mathbf{I} \right) \left\| \mathbf{X}^{(t)} - \bar{\mathbf{X}}^{(t)} \right\|_F \stackrel{(14.2)}{\geq} \frac{p}{2} \left\| \mathbf{X}^{(t)} - \bar{\mathbf{X}}^{(t)} \right\|_F \quad \square$$

Used Inequalities

Lemma 14.1.2. *For $\mathbf{A} \in \mathbb{R}^{d \times n}$, $\mathbf{B} \in \mathbb{R}^{n \times n}$, and \mathbf{B} symmetric*

$$\|\mathbf{A}\mathbf{B}\|_F \geq |\lambda_{\min}(\mathbf{B})| \|\mathbf{A}\|_F, \quad (14.1)$$

where $\lambda_{\min}(\mathbf{B})$ is the smallest eigenvalue by the absolute value.

14.1. Efficient Implementation of Consensus Control for Data-Center Training

Lemma 14.1.3. *Let \mathbf{W} be a fixed mixing matrix satisfying Assumption 6. Then,*

$$\lambda_{\min}\left(\mathbf{W} - \frac{\mathbf{1}\mathbf{1}^\top}{n} - \mathbf{I}\right) \geq \frac{p}{2}. \quad (14.2)$$

Proof. Let $\mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$ be SVD-decomposition of \mathbf{W} . By Assumption 6, \mathbf{W} is symmetric and doubly stochastic. Because of the stochastic property of \mathbf{W} , one of the eigenvalues is equal to 1 with the corresponding eigenvector $u_1 = \frac{1}{\sqrt{n}}\mathbf{1}$.

We can represent the matrix $\frac{\mathbf{1}\mathbf{1}^\top}{n}$ and \mathbf{I} as

$$\frac{\mathbf{1}\mathbf{1}^\top}{n} = u_1 u_1^\top = \mathbf{U} \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \dots & & & \\ 0 & 0 & \dots & 0 \end{pmatrix} \mathbf{U}^\top \quad \mathbf{I} = \mathbf{U}\mathbf{U}^\top.$$

Therefore,

$$\mathbf{W} - \frac{\mathbf{1}\mathbf{1}^\top}{n} - \mathbf{I} = \mathbf{U} \left[\mathbf{\Lambda} - \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \dots & & & \\ 0 & 0 & \dots & 0 \end{pmatrix} - \mathbf{I} \right] \mathbf{U}^\top = \mathbf{U} \begin{pmatrix} -1 & 0 & \dots & 0 \\ 0 & \lambda_2 - 1 & \dots & 0 \\ \dots & & & \\ 0 & 0 & \dots & \lambda_n - 1 \end{pmatrix} \mathbf{U}^\top$$

$$\lambda_{\min}\left(\mathbf{W} - \frac{\mathbf{1}\mathbf{1}^\top}{n} - \mathbf{I}\right) = \min\{1, |1 - \lambda_i(\mathbf{W})|\}, \quad i \geq 2$$

We will prove now that every $\lambda_i(\mathbf{W}), i \geq 2$ is smaller than $\sqrt{1-p}$. Then it would follow that $\lambda_{\min}\left(\mathbf{W} - \frac{\mathbf{1}\mathbf{1}^\top}{n} - \mathbf{I}\right) \geq 1 - \sqrt{1-p} \geq \frac{p}{2}$ for $0 \leq p \leq 1$.

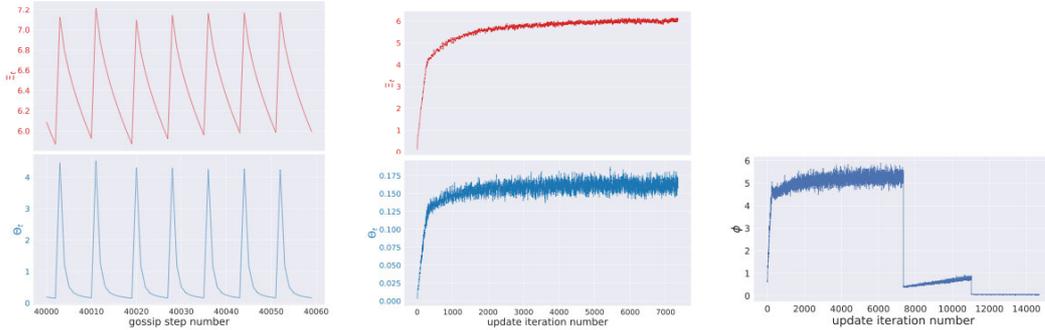
Lets assume that one of the $\lambda_i(\mathbf{W})$ is greater than $\sqrt{1-p}$. W.l.o.g. lets call this eigenvalue λ_2 . Its corresponding eigenvector is u_2 . Since the eigenvectors are orthogonal to each other and the first is $u_1 = \mathbf{1}$, it should hold that $u_1^\top u_2 = \mathbf{1}^\top u_2 = 0$. Let's take \mathbf{X} such that its every column is equal to $1/n + u_2$. Then $\bar{\mathbf{X}} = \frac{\mathbf{1}\mathbf{1}^\top}{n}$ and $(\mathbf{I} - \frac{\mathbf{1}\mathbf{1}^\top}{n})\mathbf{X} = \mathbf{X} - \bar{\mathbf{X}} = (u_2, \dots, u_2) =: \mathbf{U}_2$ is a matrix with all columns equal to u_2 .

$$\|\mathbf{W}\mathbf{X} - \bar{\mathbf{X}}\|_F^2 = \left\| \left(\mathbf{W} - \frac{\mathbf{1}\mathbf{1}^\top}{n}\right) \left(\mathbf{I} - \frac{\mathbf{1}\mathbf{1}^\top}{n}\right) \mathbf{X} \right\|_F^2 = \left\| \left(\mathbf{W} - \frac{\mathbf{1}\mathbf{1}^\top}{n}\right) \mathbf{U}_2 \right\|_F^2 = \|\lambda_2 \mathbf{U}_2\|_F^2 = \lambda_2^2 \|\mathbf{X} - \bar{\mathbf{X}}\|_F^2$$

Since the Assumption 6 holds for the \mathbf{W} for all \mathbf{X} , it should also hold for our chosen above \mathbf{X} and

$$\lambda_2^2 \|\mathbf{X} - \bar{\mathbf{X}}\|_F^2 = \|\mathbf{W}\mathbf{X} - \bar{\mathbf{X}}\|_F^2 = \left\| \left(\mathbf{W} - \frac{\mathbf{1}\mathbf{1}^\top}{n}\right) \left(\mathbf{I} - \frac{\mathbf{1}\mathbf{1}^\top}{n}\right) \mathbf{X} \right\|_F^2 \leq (1-p) \left\| \left(\mathbf{I} - \frac{\mathbf{1}\mathbf{1}^\top}{n}\right) \mathbf{X} \right\|_F^2$$

We got a contradiction which concludes the proof. □



(a) Consensus distance Ξ (top figure) and the local estimator Θ (bottom figure) over gossip steps. (b) Consensus distance Ξ (top figure) and the local estimator Θ (bottom figure) over training iterations of phase-1. (c) The average norm of local gradients ϕ

Figure 14.1 – Dec-phase-1: ResNet-20 on CIFAR-10 with a ring topology ($n = 32$), under the efficient implementation of the consensus control with the ratio $q = 1e-3$. To illustrate the evolution of the consensus distance, the plots are made over gossip steps. Note, typically several gossip steps correspond to one training iteration for consensus control. In Figure 14.1(b), both Ξ and Θ are plotted over update iterations which correspond to the last gossip steps of all iterations (i.e. the troughs in Figure 14.1(a)); the gossip steps in Figure 14.1(a) correspond to iteration steps 4824 – 4831 in Figure 14.1(b), and we only showcase an arbitrary interval in Figure 14.1(a) due to the consistent pattern over the entire phase 1.

Table 14.1 – **Efficient consensus control of dec-phase-1 with local estimates** of training ResNet-20 on CIFAR-10 ($n=32$).

	Centralized	$q = 1e-4$	$q = 1e-3$	$q = 1e-2$	w/o control
dec-phase-1	92.82 ± 0.27	92.85 ± 0.16	92.69 ± 0.31	92.44 ± 0.02	91.78 ± 0.35

14.1.2 Experiments with the Efficient Consensus Control Scheme

We implement the efficient consensus control scheme to train ResNet-20 on CIFAR-10 with a ring topology ($n = 32$). We compute Θ after each gossip step as an indicator of the exact consensus distance Ξ . The gossip continues until $\Theta < q\phi^{ema}$, where q is the control factor and ϕ^{ema} is the exponential moving average estimator of the average norm of local gradients ϕ . Please refer to Section 5.5.2 for other training details.

We validate Lemma 14.1.1 by Figure 14.1(a) and Figure 14.1(b). In Figure 14.1(a), we can observe that during an arbitrary interval of the control phase the high correlation between Ξ and Θ over gossips steps. In Figure 14.1(b), we can observe that this corrected behavior also manifests in a large span of iterations. These observations justify our claim that the Θ can act as a decent and much more inexpensive estimator of Ξ . We also plot ϕ over iterations in Figure 14.1(c) to demonstrate that the critical consensus distance η stays relatively constant within each training phase.

In Table 14.1, we show the test performance of the dec-phase-1 under the control of this efficient implementation. The pattern is consistent with the discovery in the main text. Moreover, in Table 14.2, we follow the “prioritizing the initial training phase” guideline in Section 5.6.

Table 14.2 – **Efficient consensus control for data-center training—combining practical guideline with local estimates**—for training ResNet-20 on CIFAR-10 ($n=32$). Based on our practical guideline (Section 5.6), we control only the initial phase (phase-1) with the local estimate (Θ), while leaving the other phases uncontrolled (normal decentralized training).

	Centralized	$q = 1e-4$	$q = 1e-3$	$q = 1e-2$	w/o control
guideline	92.82 ± 0.27	93.10 ± 0.13	92.79 ± 0.17	92.64 ± 0.14	91.78 ± 0.35

Specifically, we control only the initial phase (phase-1) with the local estimate, while leaving the other phases uncontrolled (normal decentralized training). We can observe that with our guideline, we can recover and surpass the centralized training baseline with only the control on the initial phase. Therefore, combining the insights into the effect of consensus distance and this efficient implementation, we open up the opportunities for practical decentralized training schemes with a desired balance between communication cost and training performance. We leave more sophisticated design for future work.

14.2 Related Work

14.2.1 Connection with Prior Work

Connection with gradient diversity. The connections between the consensus distance and gradient diversity measure (Yin et al., 2018; Johnson et al., 2020) are not obvious and is an interesting direction for future works. On the one hand, low gradient diversity could cause generalization degradation of decentralized methods as in the centralized case; on the other hand, high gradient diversity increases the difficulty of reaching a low consensus distance.

Connection with other methods like SWA/SWAP. Our empirical insights bear similarity to the ones in SWA (Izmailov et al., 2018), SWAP (Gupta et al., 2020), and Post-local SGD (Lin et al., 2020d), but none of them considers decentralized deep learning.

In SWA, models are sampled from the later stages of an SGD training run: the average of these model parameters result in a model with much higher generalization performance. SWAP extends SWA to a parallel fashion: it uses a large batch size to train the model until close to convergence and then switches to several individual runs with a small mini-batch size. These individual runs serve as a way of sampling from a posterior distribution and sampled models are averaged for better generalization performance (i.e. the idea of SWA).

Post-local SGD, SWA, SWAP, as well as the empirical insights presented in our paper, are closely related: we first need sufficient small consensus distance to guarantee the optimization quality (in post-local SGD, SWA, and SWAP, the consensus distance equals 0), and thus various choices of model averaging can be utilized in the later training phase for better generalization. Considering the later training phase, our empirical observations in decentralized learning suggest that we

can improve the generalization through the simultaneous SGD with gossip averaging. This is analogous to SWA and SWAP that sample model independently (i.e. perform SGD) from the well-trained model and average over sampled models; and close to Post-local SGD which performs simultaneous SGD steps with infrequent averaging.

14.2.2 Discussion on “Convergence Analysis v.s. Generalization Performance”

From convergence analysis to better understand generalization. A line of recent research reveals the interference between initial training (optimization) (Jastrzebski et al., 2020; Golatkar et al., 2019; Achille et al., 2019) and the later reached local minima (generalization) (Neyshabur, 2017; Lin et al., 2020d,a; Gupta et al., 2020; Izmailov et al., 2018; Keskar et al., 2017): the generalization of the deep nets cannot be studied alone via vacuous generalization bounds, and its practical performance is contingent on the critical initial learning (optimization) phase, which can be characterized by the conventional convergence analysis (Achille et al., 2019; Izmailov et al., 2018; Golatkar et al., 2019; Lin et al., 2020d; Gupta et al., 2020; Jastrzebski et al., 2020).

This motivates us to derive the metric (i.e. critical consensus distance) from the convergence analysis, for the examination of the consensus distance (on various phases) in decentralized deep learning training. For example, (1) we identify the impact of various consensus distances at the critical learning phase on the quality of initial optimization, and the final generalization (Jastrzebski et al., 2020; Golatkar et al., 2019; Achille et al., 2019; Lin et al., 2020d) (i.e. our studied case of dec-phase-1), and (2) we reveal similar observations as in (Lin et al., 2020d,a; Gupta et al., 2020; Izmailov et al., 2018) when the optimization is no longer a problem (our studied case of dec-phase-2), where the existence of consensus distance can act as a form of noise injection (Lin et al., 2020d) or sampling models from the posterior distribution (Gupta et al., 2020; Izmailov et al., 2018) as discussed above.

14.3 Theory

In this section, we prove the claims from Section 5.4.

14.3.1 Proof of Proposition 5.4.2, Critical Consensus Distance

The proof of this claim follows by the following Lemma:

Lemma 14.3.1 (Koloskova et al. (2020b), Descent lemma for non-convex case). *Under the given assumptions, and for any stepsize $\eta < \frac{1}{4L}$, the iterates of D-SGD satisfy*

$$\mathbb{E}_{t+1} f(\bar{\mathbf{x}}^{(t+1)}) \leq f(\bar{\mathbf{x}}^{(t)}) - \frac{\eta}{4} \|\nabla f(\bar{\mathbf{x}}^{(t)})\|_2^2 + \eta L^2 \Xi_t^2 + \frac{L}{n} \eta^2 \hat{\sigma}^2.$$

Proof. By replacing Ξ_t in the above inequality with (5.4), we simplify:

$$\mathbb{E}_{t+1} f(\bar{\mathbf{x}}^{(t+1)}) \leq f(\bar{\mathbf{x}}^{(t)}) - \frac{\eta}{8} \|\nabla f(\bar{\mathbf{x}}^{(t)})\|_2^2 + \frac{2L}{n} \eta^2 \hat{\sigma}^2.$$

This inequality now matches (up to differences in the constants) the standard recursion that one can derive for C-SGD (Dekel et al., 2012; Bottou et al., 2018; Stich, 2019b; Stich and Karimireddy, 2020). \square

14.3.2 Proof of Proposition 5.4.3, Typical Consensus Distance

We need an auxiliary (but standard) lemma, to estimate the change of the consensus distance between iterations.

Lemma 14.3.2 (Consensus distance). *It holds*

$$\Xi_{t+1}^2 \leq (1 - p/2) \Xi_t^2 + \frac{3(1-p)\eta^2}{p} (\phi_t^2 + p\sigma^2).$$

We give the proof of this statement shortly below. First, let us consider how this lemma allows the proof of the claim. For this, we first consider a particular special case, and assume $\phi_t \leq \phi$, for a constant ϕ . In this case, we can easily verify by unrolling the recursion:

$$\Xi_t^2 \leq \sum_{i=0}^{t-1} (1 - p/2)^i \frac{3(1-p)\eta^2(\phi^2 + p\sigma^2)}{p} \leq 6(1-p)\eta^2 \left(\frac{\phi^2}{p^2} + \frac{\sigma^2}{p} \right).$$

Now, for the claim in the main text, we use assumption that ϕ_t are changing slowly, that is, not decreasing faster than exponentially: $\phi_t^2 \leq (1 + p/4)\phi_{t+1}^2$. With this assumption, and observing $(1 - p/2)^i (1 + p/4)^i \leq (1 - p/4)^i$, we can unroll as before

$$\begin{aligned} \Xi_t^2 &\leq \sum_{i=0}^{t-1} (1 - p/2)^i \frac{3(1-p)\eta^2(\phi_{t-i-1}^2 + p\sigma^2)}{p} \\ &\leq \sum_{i=0}^{t-1} (1 - p/4)^i \frac{3(1-p)\eta^2(\phi_{t-1}^2 + p\sigma^2)}{p} \leq 12(1-p)\eta^2 \left(\frac{\phi_{t-1}^2}{p^2} + \frac{\sigma^2}{p} \right). \end{aligned}$$

Proof of Lemma 14.3.2. We use the following matrix notation here

$$\begin{aligned} \mathbf{X}^{(t)} &:= [\mathbf{x}_1^{(t)}, \dots, \mathbf{x}_n^{(t)}] \in \mathbb{R}^{d \times n}, \\ \bar{\mathbf{X}}^{(t)} &:= [\bar{\mathbf{x}}^{(t)}, \dots, \bar{\mathbf{x}}^{(t)}] = \mathbf{X}^{(t)} \frac{1}{n} \mathbf{1}\mathbf{1}^\top, \\ \nabla F(\mathbf{X}^{(t)}, \xi^{(t)}) &:= [\nabla F_1(\mathbf{x}_1^{(t)}, \xi_1^{(t)}), \dots, \nabla F_n(\mathbf{x}_n^{(t)}, \xi_n^{(t)})], \\ \nabla f(\mathbf{X}^{(t)}) &:= [\nabla f_1(\mathbf{x}_1^{(t)}), \dots, \nabla f_n(\mathbf{x}_n^{(t)})]. \end{aligned}$$

As a reminder, $\Xi_t^2 := \frac{1}{n} \sum_{i=1}^n \|\bar{\mathbf{x}}^{(t)} - \mathbf{x}_i^{(t)}\|^2$, and $\phi_t^2 := \frac{1}{n} \sum_{i=1}^n \|\nabla f_i(\mathbf{x}_i^{(t)})\|^2$.

$$\begin{aligned}
 n\Xi_{t+1}^2 &= \|\bar{\mathbf{X}}^{(t+1)} - \mathbf{X}^{(t+1)}\|_F^2 = \left\| (\mathbf{X}^{(t)} - \eta \nabla F(\mathbf{X}^{(t)}, \xi^{(t)})) \left(\frac{1}{n} \mathbf{1}\mathbf{1}^\top - \mathbf{W} \right) \right\|_F^2 \\
 &= \left\| (\mathbf{X}^{(t)} - \eta \nabla F(\mathbf{X}^{(t)}, \xi^{(t)})) \left(\frac{1}{n} \mathbf{1}\mathbf{1}^\top - \mathbf{I} \right) \left(\frac{1}{n} \mathbf{1}\mathbf{1}^\top - \mathbf{W} \right) \right\|_F^2 \\
 &\leq (1-p) \left\| (\mathbf{X}^{(t)} - \eta \nabla F(\mathbf{X}^{(t)}, \xi^{(t)})) \left(\frac{1}{n} \mathbf{1}\mathbf{1}^\top - \mathbf{I} \right) \right\|_F^2 \\
 &\leq (1-p) \left\| (\mathbf{X}^{(t)} - \eta \nabla f(\mathbf{X}^{(t)})) \left(\frac{1}{n} \mathbf{1}\mathbf{1}^\top - \mathbf{I} \right) \right\|_F^2 + (1-p)\eta^2 \|\nabla f(\mathbf{X}^{(t)}) - \nabla F(\mathbf{X}^{(t)}, \xi^{(t)})\|_F^2 \\
 &\leq (1-p)(1+\alpha) \left\| \mathbf{X}^{(t)} \left(\frac{1}{n} \mathbf{1}\mathbf{1}^\top - \mathbf{I} \right) \right\|_F^2 + (1-p)(1+\alpha^{-1})\eta^2 \|\nabla f(\mathbf{X}^{(t)})\|_F^2 + (1-p)\eta^2 \sigma^2 n \\
 &\stackrel{\alpha=\frac{p}{2}}{\leq} \left(1 - \frac{p}{2}\right) n\Xi_t^2 + \frac{3(1-p)}{p} \eta^2 \|\nabla f(\mathbf{X}^{(t)})\|_F^2 + (1-p)\eta^2 \sigma^2 n. \quad \square
 \end{aligned}$$

14.3.3 Sufficient Bounds to Meet Critical Consensus Distance

In this section, we show that the claimed bounds in Section 5.4.3 are sufficient conditions to reach the CCD.

According to Proposition 5.4.3, there exists an absolute constant C , (w.l.o.g. $C \geq 2$) such that

$$\Xi_t^2 \leq C(1-p)\eta^2 \left(\frac{\phi_t^2}{p^2} + \frac{\sigma^2}{p} \right).$$

By smoothness,

$$\begin{aligned}
 \phi_t^2 &= \frac{1}{n} \sum_{i=1}^n \|\nabla f_i(\mathbf{x}_i^{(t)})\|^2 \\
 &\leq \frac{3}{n} \sum_{i=1}^n \|\nabla f_i(\mathbf{x}_i^{(t)}) - \nabla f(\mathbf{x}_i^{(t)})\|^2 + \frac{3}{n} \sum_{i=1}^n \|\nabla f(\mathbf{x}_i^{(t)}) - \nabla f(\bar{\mathbf{x}}^{(t)})\|^2 + \frac{3}{n} \sum_{i=1}^n \|\nabla f(\bar{\mathbf{x}}^{(t)})\|^2 \\
 &\leq 3\zeta^2 + 3L^2\Xi_t^2 + 3\|\nabla f(\bar{\mathbf{x}}^{(t)})\|^2.
 \end{aligned}$$

Supposing $(1-p)\eta^2 \leq \frac{p^2}{6CL^2}$, we can therefore estimate

$$\begin{aligned}
 \Xi_t^2 &\leq C(1-p)\eta^2 \left(\frac{3\|\nabla f(\bar{\mathbf{x}}^{(t)})\|^2 + 3L^2\Xi_t^2 + 3\zeta^2}{p^2} + \frac{\sigma^2}{p} \right) \\
 &\leq 3C(1-p)\eta^2 \left(\frac{\|\nabla f(\bar{\mathbf{x}}^{(t)})\|^2 + \zeta^2}{p^2} + \frac{\sigma^2}{p} \right) + \frac{1}{2}\Xi_t^2,
 \end{aligned}$$

and hence

$$\Xi_t^2 \leq 6C(1-p)\eta^2 \left(\frac{\|\nabla f(\bar{\mathbf{x}}^{(t)})\|^2}{p^2} + \frac{\zeta^2}{p^2} + \frac{\sigma^2}{p} \right). \quad (14.3)$$

The claimed bounds can now easily be verified, by plugging the provided values into (14.3). For simplicity in the main text we assume that $\zeta = 0$ (we are in the datacenter training scenario).

Small η . By choosing $\eta \leq \frac{p}{4nLC}$, we check that our previous constraint $\eta^2 \stackrel{C \geq 2}{\leq} \frac{p^2}{6CL^2}$ is satisfied, and

$$(14.3) \leq \frac{\|\nabla f(\bar{\mathbf{x}}^{(t)})\|^2}{4n^2CL^2} + \frac{\eta\sigma^2}{nL} \stackrel{C \geq 2}{\leq} (5.4).$$

Small p . By choosing $1-p \leq \frac{1}{5C(1+\eta Ln)}$, we note that $p \stackrel{C \geq 2}{\geq} \frac{9}{10}$. Moreover, our previous constraint $(1-p)\eta^2 \leq \frac{\eta^2}{5C} \leq \frac{p^2}{6L^2C}$ is satisfied (note that $\eta \leq \frac{1}{4L}$ throughout). Hence

$$(14.3) \leq \frac{4\eta^2}{5(1+\eta Ln)} \left(\frac{100\|\nabla f(\bar{\mathbf{x}}^{(t)})\|^2}{81} + \frac{10\sigma^2}{9} \right) \stackrel{\eta \leq 1/(4L)}{\leq} (5.4)$$

In the above calculations we for the simplicity assumed that $\zeta = 0$. For the general non-iid data case when $\zeta > 0$ we can calculate similar bounds on η, p . These bounds would have similar dependence on parameters, and would be stricter. Indeed, the typical consensus distance would be also influenced by non-iidness of the data ζ and it is therefore harder to satisfy the CCD condition.

14.3.4 Proof of Lemma 5.4.4, Repeated Gossip

By the assumption stated in the lemma, it holds for each component \mathbf{W}_i of the product $\mathbf{W} = \mathbf{W}_k \dots \mathbf{W}_1$, $i \in [1, k]$ that

$$\mathbb{E}_{\mathbf{W}_i} \|\mathbf{X}\mathbf{W}_i - \bar{\mathbf{X}}\|_F^2 \leq (1-p) \|\mathbf{X} - \bar{\mathbf{X}}\|_F^2, \forall \mathbf{X} \in \mathbb{R}^{d \times n}.$$

Now lets estimate the parameter $p_{\mathbf{W}}$. Using that \mathbf{W}_i are independent

$$\mathbb{E}_{\mathbf{W}} \|\mathbf{X}\mathbf{W} - \bar{\mathbf{X}}\|_F^2 = \mathbb{E}_{\mathbf{W}_1 \dots \mathbf{W}_k} \|\mathbf{X}\mathbf{W}_k \dots \mathbf{W}_1 - \bar{\mathbf{X}}\|_F^2 = \mathbb{E}_{\mathbf{W}_2 \dots \mathbf{W}_k} \mathbb{E}_{\mathbf{W}_1} \|\mathbf{Y}\mathbf{W}_1 - \bar{\mathbf{Y}}\|_F^2,$$

where we defined $\mathbf{Y} = \mathbf{X}\mathbf{W}_k \dots \mathbf{W}_2$ and used that $\mathbf{W}_i \frac{1}{n} \mathbf{1}\mathbf{1}^\top = \frac{1}{n} \mathbf{1}\mathbf{1}^\top$, so

$$\bar{\mathbf{Y}} = \mathbf{X}\mathbf{W}_k \dots \mathbf{W}_2 \frac{1}{n} \mathbf{1}\mathbf{1}^\top = \mathbf{X} \frac{1}{n} \mathbf{1}\mathbf{1}^\top = \bar{\mathbf{X}}.$$

Therefore,

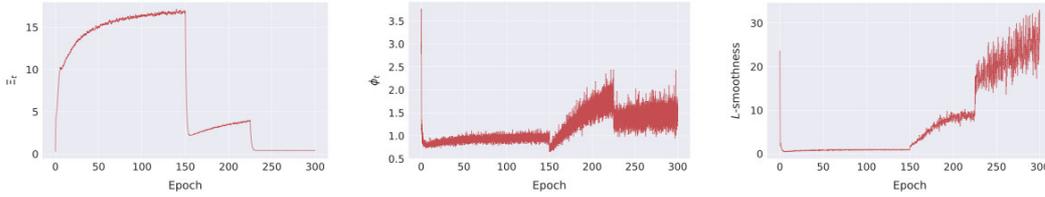
$$\mathbb{E}_{\mathbf{W}} \|\mathbf{XW} - \bar{\mathbf{X}}\|_F^2 \leq (1-p) \mathbb{E}_{\mathbf{W}_2 \dots \mathbf{W}_k} \|\mathbf{XW}_k \dots \mathbf{W}_2 - \bar{\mathbf{X}}\|_F^2.$$

Applying the same calculations to the rest, we conclude that $1 - p_{\mathbf{W}} = (1-p)^k$.

14.4 Detailed Experimental Setup

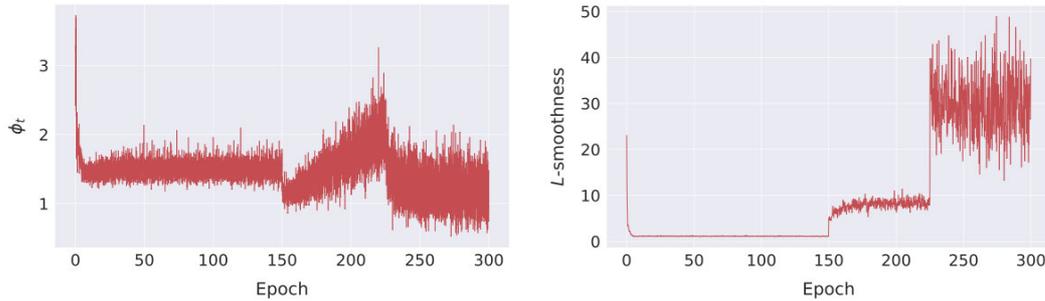
Comments on large-batch training. Coupling the quality loss issue of the decentralized training with the large-batch training difficulty is non-trivial and is out of the scope of this paper. Instead, we use reasonable local mini-batch sizes (together with the number of workers (denoted as n)), as well as the well-developed large-batch training techniques (Goyal et al., 2017), to avoid the difficulty of extreme large-batch training.

Multi-phase experiment justification. The averaged local gradient norm ϕ_t as well as the L -smoothness of ResNet-20 on CIFAR-10 for a ring and a complete graph ($n=32$) are shown in Figure 14.2 and Figure 14.3 respectively.



(a) The consensus distance for decentralized training. (b) The averaged norm of the local gradients for decentralized training. (c) The gradient Lipschitz curve for decentralized training.

Figure 14.2 – **Justification for our multiple-phase experimental design choice** (on ring graph). We run ResNet-20 on CIFAR-10 ($n=32$) with the ring topology. We can observe the three quantities most relevant to optimization all naturally form three phases, dictated by the learning rate schedule.



(a) The averaged norm of the local gradients for centralized training. (b) The gradient Lipschitz curve for centralized training.

Figure 14.3 – **Justification for our multiple-phase experimental design choice** (on complete graph). We run ResNet-20 on CIFAR-10 ($n=32$) with the complete topology. We can again observe the three quantities most relevant to optimization all naturally form three phases, dictated by the learning rate schedule.

The estimation procedure is analogous to that in Santurkar et al. (2018); Lin et al. (2020a): we take 8 additional steps long the direction of current update, each with 0.2 of normal step size. This is calculated at every 8 training steps. The smoothness is evaluated as the maximum value of L satisfying Assumption 7.

14.5 Additional Results

14.5.1 Understanding on Consensus Averaging Problem

We study a host of communication topologies: (1) deterministic topologies (ring, and complete graph) and (2) undirected time-varying topologies (illustrated below).

- **Random matching** (Boyd et al., 2006). At each communication step, all nodes are divided into non-overlapping pairs randomly. Each node connects all other nodes with equal probability.
- **Exponential graph** (Assran et al., 2019). Each is assigned a rank from 0 to $n - 1$. Each node i periodically communicates with a list nodes with rank $i + 2^0, i + 2^1, \dots, i + 2^{\lfloor \log_2(n-1) \rfloor}$. In the one-peer-per-node experiments, each node only communicates to one node by cycling through its list. The formed graph is undirected, i.e. both transmission and reception take place in each communication.
- **Bipartite exponential graph** (Lian et al., 2018; Assran et al., 2019). In order to avert deadlocks (Lian et al., 2018), the node with an odd rank i cycles through nodes with even ranks $i + 2^0 - 1, i + 2^1 - 1, \dots, i + 2^{\lfloor \log_2(n-1) \rfloor} - 1$ by transmitting a message and waiting for a response. while the nodes with even ranks only await messages and reply upon reception.

Table 14.3 displays the spectral gap and node degree of studied topologies, and Figure 14.4 provides the convergence curves for various communication topologies on graph scales. Figure 14.5 in addition visualizes the spectral gap (in expectation) for various communication topologies.

Table 14.4 examines these topologies on a standard deep learning benchmark with various graph scales, while Figure 14.6 visualizes the required communication rounds (per gradient update step) for a range of consensus distance targets.

Table 14.3 – Spectral gap and node degree of studied topologies.

Topologies	Spectral Gaps (in expectation)	Node degrees (n nodes)
Complete	1	n
Fixed ring	$\mathcal{O}(\frac{1}{n^2})$	2
Exponential graph	$\mathcal{O}(1)$	2
Bipartite exponential graph	$\mathcal{O}(1)$	1
Random matching	$\mathcal{O}(1)$	1

Chapter 14. Appendix for Consensus Control in Decentralized Learning

Table 14.4 – **The effect of communication topologies and scales** (ResNet-20 on CIFAR-10). The test top-1 accuracies are over three seeds with fine-tuned learning rates.

	Complete	Fixed ring	Exponential graph	Bipartite exponential graph	Random matching
n=16	92.91 ± 0.12	92.51 ± 0.19	92.63 ± 0.30	92.76 ± 0.04	92.65 ± 0.15
n=32	92.82 ± 0.27	91.93 ± 0.05	92.64 ± 0.04	92.29 ± 0.15	92.27 ± 0.17

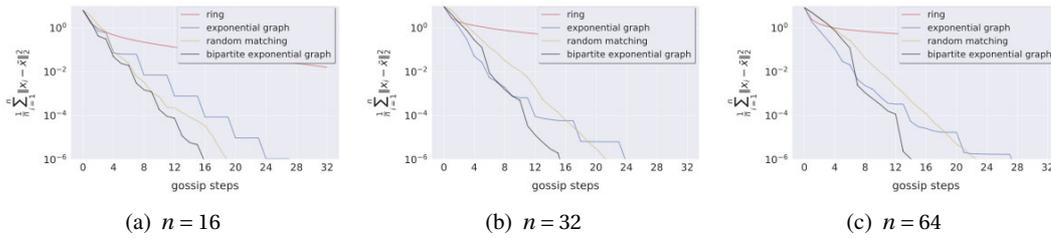


Figure 14.4 – The convergence curves for the consensus averaging problem on various communication topologies and scales (i.e. $n=16$, $n=64$ and $n=128$). This figure complements the Figure 5.4 in the main text.

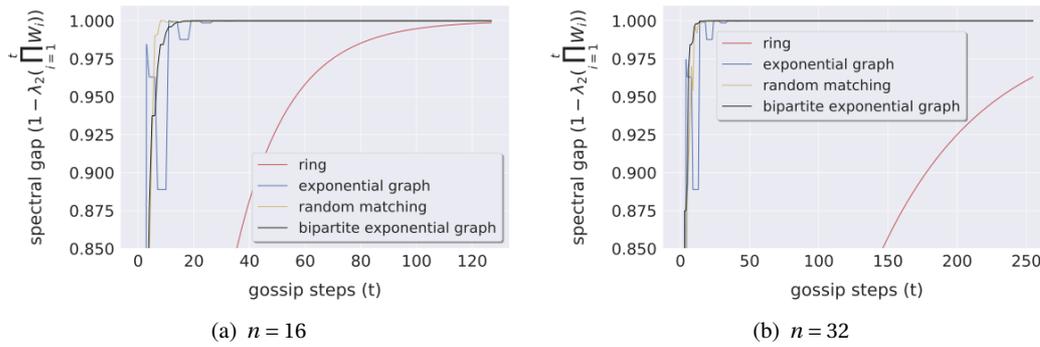


Figure 14.5 – The spectral gap (in expectation) of various communication topologies and graph scales.

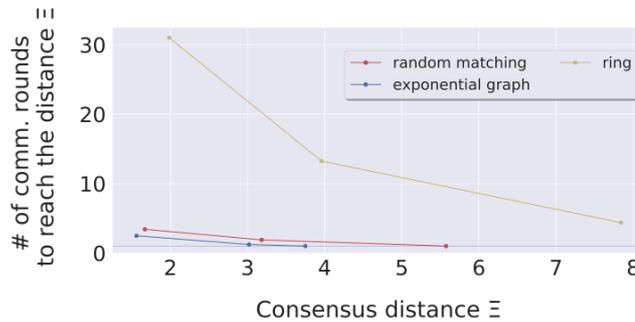


Figure 14.6 – **Target consensus distance v.s. the required communication rounds** (per gradient update step), for training ResNet-20 on CIFAR-10 with various communication topologies. We focus on the setup of dec-phase-1 and vary the target consensus distance for various communication topologies. Due to the changing consensus distance over the training (of the interested phase-1), we consider the averaged consensus distance. The topologies of exponential graph and random matching, empower the capability of fast convergence in gossip averaging and thus only a few steps are required to reach the target consensus distance.

Table 14.5 – **The effect of SlowMo for decentralized learning**, for training ResNet20 on CIFAR-10 ($n = 32$). The results (over three random seeds) use the tuned hyperparameter of SlowMo mentioned in the original paper (Wang et al., 2020d). The centralized baseline performance is 92.82 ± 0.27 .

topology	w/o SlowMo	w/ SlowMo
exponential graph	92.63 ± 0.22	92.42 ± 0.36
ring	91.74 ± 0.15	92.53 ± 0.10

Table 14.6 – **Phase-1 consensus distance control performance with fine-tuned learning rates** of training ResNet-20 on CIFAR-10 ($n = 32$). Setup in this table is identical to that of Table 5.2, except that we fine-tune the learning rate for each case from a grid of linear scaling-up factors $\{30, 28, 26, 24, 22\}$. The results are over three seeds.

	Ξ_{\max}	$1/2 \Xi_{\max}$	$1/4 \Xi_{\max}$
w/ tuned lr from the search grid	91.95 ± 0.26	92.35 ± 0.24	92.54 ± 0.08
w/ default lr	91.78 ± 0.35	92.36 ± 0.21	92.74 ± 0.10

14.5.2 Understanding the Decentralized Deep Learning Training for CV Tasks

We use ring as our underlying decentralized communication topology in this subsection.

Elaborated results on consensus distance control. Table 14.10 is the elaborated version of Table 5.2 with more evaluated consensus distances.

SlowMo cannot fully address the decentralized optimization/generalization difficulty. Table 14.5 studies the effectiveness of using SlowMo for better decentralized training. We can witness that even though the performance of decentralized training can be boosted to some extent, it cannot fully address the quality loss issue brought by decentralized training.

On the ineffectiveness of tuning learning rate. Table 14.6 displays the results of training ResNet-20 on CIFAR-10 (32 nodes), with fine-tuned learning rate on phase-1; learning rate tuning cannot address the test quality loss issue caused by the large consensus distance (i.e. over the CCD).

Prolonged training for dec-phase-2 and dec-phase-3. Table 14.7 shows the results for prolonged dec-phase-2 and dec-phase-3 on CIFAR-10 with ResNet20. We can observe although longer training duration increases the performance, the improvement is rather small.

The impact of half cosine learning rate schedule. Table 14.8 examines the existence of the critical consensus distance with half cosine learning schedule (this scheme is visited in (He et al., 2019a) as a new paradigm for CNN training). We can witness from Table 14.8 that the effect of

Chapter 14. Appendix for Consensus Control in Decentralized Learning

Table 14.7 – **The impact of various numbers of training epochs (at phase-2 and phase-3)** on generalization, for training ResNet-20 on CIFAR-10 (ring topology with $n=32$). The number of epochs at phase-1 is chosen from $\{75, 100, 125\}$, while the rest of the training reuses our default setup. Experiments are run over 2 seeds.

# nodes	target Ξ	dec-phase-2			dec-phase-3		
	Ξ_{\max}	$1/2 \Xi_{\max}$	$1/4 \Xi_{\max}$	Ξ_{\max}	$1/2 \Xi_{\max}$	$1/4 \Xi_{\max}$	
75 epochs	93.04 ± 0.01	92.99 ± 0.30	92.87 ± 0.11	92.60 ± 0.00	92.82 ± 0.21	92.85 ± 0.24	
100 epochs	93.08 ± 0.08	93.05 ± 0.16	92.94 ± 0.03	92.86 ± 0.16	92.90 ± 0.18	92.93 ± 0.19	
125 epochs	93.19 ± 0.16	93.11 ± 0.17	93.06 ± 0.07	92.87 ± 0.23	92.99 ± 0.25	92.97 ± 0.20	

Table 14.8 – **The impact of half cosine learning rate schedule** on generalization, for training ResNet20 on CIFAR-10 (ring topology with $n=32$). The inline figure depicts the uncontrolled consensus distance over the whole training procedure through the half-cosine learning rate schedule. Only one training phase is considered for the consensus distance control and the numerical results in the table are averaged over 3 seeds.

Ring (Ξ_{\max})	Ring ($1/2\Xi_{\max}$)	Ring ($1/4\Xi_{\max}$)	Ring ($1/8\Xi_{\max}$)	Complete
92.10 ± 0.06	92.40 ± 0.10	92.83 ± 0.11	92.78 ± 0.05	92.84 ± 0.22

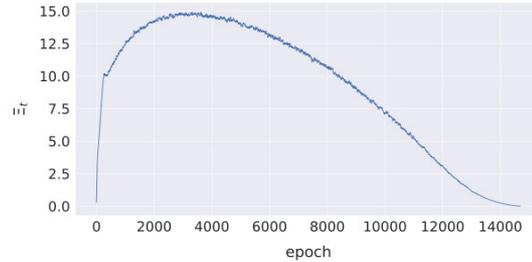


Table 14.9 – **The impact of consensus distances on optimization and/or generalization, for various phases** of training ResNet-20 on CIFAR-10 ($n=32$). The table is almost identical to Table 5.2, except the consensus distance is controlled by the (runtime) averaged norm of the local gradients (i.e. adaptive consensus distance).

	Ξ_{\max}	$4\phi_t^{\text{ema}}$	$2\phi_t^{\text{ema}}$	ϕ_t^{ema}	$0.5\phi_t^{\text{ema}}$
Phase 1	91.78 ± 0.35	91.65 ± 0.31	92.47 ± 0.18	92.63 ± 0.04	92.80 ± 0.16
Phase 2	93.04 ± 0.01	93.05 ± 0.18	93.01 ± 0.03	93.03 ± 0.08	92.95 ± 0.10
Phase 3	92.94 ± 0.07	92.87 ± 0.18	92.83 ± 0.20	-	-

critical consensus distance can be generalized to this learning rate schedule: there exists a critical consensus distance in the initial training phase (as revealed in the inline Figure of Table 14.8) and ensures good optimization and generalization.

Adaptive Consensus Distance Control

In Table 14.9, we apply the adaptive consensus distance control in the experiments. The observations are consistent with those in constant consensus distance control experiments.

14.5.3 Consensus Control with Other Topologies

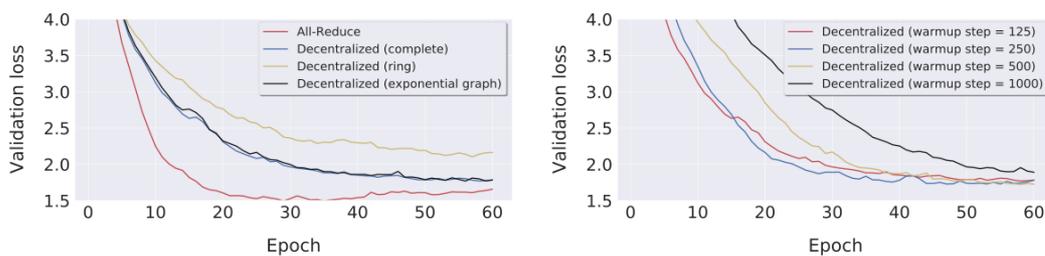
In Table 14.11, we exert consensus control with an exponential graph as the base communication topology; the local update step corresponds to the number of local model update steps per communication round, and we use it as a way to increase discrepancy (consensus distance) among nodes. We can observe that our findings from main experiments with a ring base topology are valid.

The Existence of the Optimal Consensus Distance for Noise Injection.

Table 14.12 uses a different communication topology (i.e. time-varying exponential graph) for decentralized optimization. Here exponential graph with large spectral gap is applied to CIFAR-10 dec-phase-2 training. We apply the adaptive consensus distance control in this set of experiments. We can observe that increasing consensus distance further by taking local steps improves generalization, however, too many local steps diminish the performance. For instance, for ratio=2, the performance peaks at local update steps 2 and drops at local update 4. It points out that an optimal consensus distance is required to inject proper stochastic noise for better generalization.

14.5.4 Results for Training Transformer on Multi30k

We additionally report the decentralized training results, for a downsampled transformer models (by the factor of 2 w.r.t. the base model in Vaswani et al. (2017)) on Multi30k (Elliott et al., 2016). Figure 14.7 shows that the straightforward application of Adam in the decentralized manner does encounter generalization problems, which are attributed to the fact that the local moment buffers (in addition to the weights) become too diverse. Tuning the learning rate schedule cannot address the issue of decentralized Adam, as shown in the Figure 14.7(b).



(a) The limitation of decentralized learning with Adam, caused by the diverse local moment buffers. (b) Tuning the learning rate cannot alleviate the issue of decentralized Adam.

Figure 14.7 – Learning curves for training the transformer model on the Multi30k dataset ($n=32$). In Figure 14.7(b), we tune the the number of warmup steps as as way of tuning the learning rate, as the learning rate used in transformer training (Vaswani et al., 2017) is deterministically controlled by the model’s dimensionality, the current step index, and the number of warmup steps.

15 Appendix for DPF

15.1 Algorithm

Algorithm 16 The detailed training procedure of DPF.

Requires: uncompressed model weights $\mathbf{x} \in \mathbb{R}^d$, pruned weights: $\hat{\mathbf{x}}$, mask: $\mathbf{m} \in \{0, 1\}^d$; reparametrization period: p ; training iterations: T .

```
1: procedure
2:   for  $t \in \{1, \dots, T\}$  do
3:     if  $p \mid t$  (i.e. trigger mask update, per default every  $p = 16$  iterations) then
4:       | compute mask  $\mathbf{m} \leftarrow \mathbf{m}_t(\mathbf{x}_t)$  ▷ by arbitrary pruning scheme
5:       |  $\hat{\mathbf{x}}_t \leftarrow \mathbf{m} \odot \mathbf{x}_t$  ▷ apply (precomputed) mask
6:       | compute (mini-batch) gradient  $\mathbf{g}(\hat{\mathbf{x}}_t)$  ▷ forward/backward with pruned weights  $\hat{\mathbf{x}}_t$ 
7:       |  $\mathbf{x}_{t+1} \leftarrow$  gradient update  $\mathbf{g}(\hat{\mathbf{x}}_t)$  to  $\mathbf{x}_t$  ▷ via arbitrary optimizer
8:   return  $\mathbf{x}_T$  and  $\hat{\mathbf{x}}_T$ 
```

We trigger the mask update every $p = 16$ iterations (see also Figure 15.5) and we keep this parameter fixed throughout all experiments, independent of architecture or task.

We perform pruning across all neural network layers (no layer-wise pruning) using magnitude-based unstructured weight pruning (inherited from Han et al. (2015)). Pruning is applied to all convolutional layers while keeping the last fully-connected layer, biases and batch normalization layers dense.

We gradually increase the sparsity s_t of the mask from 0 to the desired sparsity using the same scheduling as in Zhu and Gupta (2017); see Section 15.2 below.

15.2 Implementation Details

We implemented our DPF in PyTorch (Paszke et al., 2017). All experiments were run on NVIDIA Tesla V100 GPUs. Sparse tensors in our implementation are represented as the dense tensors

multiplied by the corresponding binary masks.

Datasets We evaluate all methods on the following standard image classification tasks:

- Image classification for CIFAR-10 (Krizhevsky and Hinton, 2009). Dataset consists of a training set of 50K and a test set of 10K color images of 32×32 pixels, as well as 10 target classes. We adopt the standard data augmentation and preprocessing scheme (He et al., 2016a; Huang et al., 2016).
- Image classification for ImageNet (Russakovsky et al., 2015). The ILSVRC 2012 classification dataset consists of 1.28 million images for training, and 50K for validation, with 1K target classes. We use ImageNet-1k (Deng et al., 2009) and adopt the same data preprocessing and augmentation scheme as in He et al. (2016a,b); Simonyan and Zisserman (2015).

Gradual Pruning Scheme For Incremental baseline, we tuned their automated gradual pruning scheme $s_t = s_f + (s_i - s_f) \left(1 - \frac{t-t_0}{n\Delta t}\right)^3$ to gradually adjust the pruning sparsity ratio s_t for $t \in \{t_0, \dots, t_0 + n\Delta t\}$. That is, in our setup, we increased from an initial sparsity ratio $s_i = 0$ to the desired target model sparsity ratio s_f over the epoch (n) when performing the second learning rate decay, from the training epoch $t_0 = 0$ and with pruning frequency $\Delta t = 1$ epoch. In our experiments, we used this gradual pruning scheme over various methods, except One-shot P+FT, SNIP, and the methods (DSR, SM) that have their own fine-tuned gradual pruning scheme.

Hyperparameters tuning procedure We grid-searched the optimal learning rate, starting from the range of $\{0.05, 0.10, 0.15, 0.20\}$. More precisely, we will evaluate a linear-spaced grid of learning rates. If the best performance was ever at one of the extremes of the grid, we would try new grid points so that the best performance was contained in the middle of the parameters.

We trained most of the methods by using mini-batch SGD with Nesterov momentum. For baselines involving fine-tuning procedure (e.g. Table 6.2), we grid-searched the optimal results by tuning the optimizers (i.e. mini-batch SGD with Nesterov momentum, or Adam) and the learning rates.

The optimal hyperparameters for DPF The mini-batch size is fixed to 128 for CIFAR-10 and 1024 for ImageNet regardless of datasets and models.

For CIFAR-10, we trained ResNet- a and VGG for 300 epochs and decayed the learning rate by 10 when accessing 50% and 75% of the total training samples (He et al., 2016a; Huang et al., 2017b); and we trained WideResNet- $a-b$ as Zagoruyko and Komodakis (2016) for 200 epochs and decayed the learning rate by 5 when accessing 30%, 60% and 80% of the total training samples. The optimal learning rate for ResNet- a , WideResNet- $a-b$ and VGG are 0.2, 0.1 and 0.2

15.3. Additional Results for Unstructured Pruning

respectively; the corresponding weight decays are $1e-4$, $5e-4$ and $1e-4$ respectively.

For ImageNet training, we used the training scheme in Goyal et al. (2017) for 90 epochs, where we gradually warmup the learning rate from 0.1 to 0.4 and decayed learning rate by 10 at 30; 60; 80 epochs. The used weight decay is $1e-4$.

15.3 Additional Results for Unstructured Pruning

15.3.1 Complete Results of Unstructured Pruning on CIFAR-10

Table 15.1 details the numerical results for training SOTA DNNs on CIFAR-10. Some results of it reconstruct the Table 6.1 and Figure 6.3.

Table 15.1 – Top-1 test accuracy for training (compressed) SOTA DNNs on **CIFAR-10** from scratch. We considered unstructured pruning and the \star indicates the method cannot converge. The results are averaged for three runs.

Model	Baseline on dense model	Methods				Target Pr. ratio
		SNIP (L ⁺ , 2019)	SM (DZ, 2019)	DSR (MW, 2019)	DPF	
VGG16-D	93.74 ± 0.13	93.04 ± 0.26	93.59 ± 0.17	-	93.87 ± 0.15	95%
ResNet-20	92.48 ± 0.20	91.10 ± 0.22	91.98 ± 0.01	92.00 ± 0.19	92.42 ± 0.14	70%
ResNet-20	92.48 ± 0.20	90.53 ± 0.27	91.54 ± 0.16	91.78 ± 0.28	92.17 ± 0.21	80%
ResNet-20	92.48 ± 0.20	88.50 ± 0.13	89.76 ± 0.40	87.88 ± 0.04	90.88 ± 0.07	90%
ResNet-20	92.48 ± 0.20	84.91 ± 0.25	83.03 ± 0.74	\star	88.01 ± 0.30	95%
ResNet-32	93.83 ± 0.12	90.40 ± 0.26	91.54 ± 0.18	91.41 ± 0.23	92.42 ± 0.18	90%
ResNet-32	93.83 ± 0.12	87.23 ± 0.29	88.68 ± 0.22	84.12 ± 0.32	90.94 ± 0.35	95%
ResNet-56	94.51 ± 0.20	91.43 ± 0.34	92.73 ± 0.21	93.78 ± 0.20	93.95 ± 0.11	90%
ResNet-56	94.51 ± 0.20	\star	90.96 ± 0.40	92.57 ± 0.09	92.74 ± 0.08	95%
WideResNet-28-2	95.01 ± 0.04	94.67 ± 0.23	94.73 ± 0.16	94.80 ± 0.14	95.11 ± 0.06	50%
WideResNet-28-2	95.01 ± 0.04	94.47 ± 0.19	94.65 ± 0.16	94.98 ± 0.07	94.90 ± 0.06	60%
WideResNet-28-2	95.01 ± 0.04	94.29 ± 0.22	94.46 ± 0.11	94.80 ± 0.15	94.86 ± 0.13	70%
WideResNet-28-2	95.01 ± 0.04	93.56 ± 0.14	94.17 ± 0.12	94.57 ± 0.13	94.76 ± 0.18	80%
WideResNet-28-2	95.01 ± 0.04	92.58 ± 0.22	93.41 ± 0.22	93.88 ± 0.08	94.36 ± 0.24	90%
WideResNet-28-2	95.01 ± 0.04	90.80 ± 0.04	92.24 ± 0.14	92.74 ± 0.17	93.62 ± 0.05	95%
WideResNet-28-2	95.01 ± 0.04	83.45 ± 0.38	85.36 ± 0.80	\star	88.92 ± 0.29	99%
WideResNet-28-4	95.69 ± 0.10	95.42 ± 0.05	95.57 ± 0.08	95.67 ± 0.07	95.58 ± 0.21	70%
WideResNet-28-4	95.69 ± 0.10	95.24 ± 0.07	95.27 ± 0.02	95.49 ± 0.04	95.60 ± 0.08	80%
WideResNet-28-4	95.69 ± 0.10	94.56 ± 0.11	95.01 ± 0.05	95.30 ± 0.12	95.65 ± 0.14	90%
WideResNet-28-4	95.69 ± 0.10	93.62 ± 0.17	94.45 ± 0.14	94.63 ± 0.08	95.38 ± 0.04	95%
WideResNet-28-4	95.69 ± 0.10	92.06 ± 0.38	93.80 ± 0.24	93.92 ± 0.16	94.98 ± 0.08	97.5%
WideResNet-28-4	95.69 ± 0.10	89.49 ± 0.20	92.18 ± 0.04	92.50 ± 0.07	93.86 ± 0.20	99%
WideResNet-28-8	96.06 ± 0.06	95.81 ± 0.05	95.92 ± 0.12	96.06 ± 0.09	-	70%
WideResNet-28-8	96.06 ± 0.06	95.86 ± 0.10	95.97 ± 0.05	96.05 ± 0.12	-	80%
WideResNet-28-8	96.06 ± 0.06	95.49 ± 0.21	95.67 ± 0.14	95.81 ± 0.10	96.08 ± 0.15	90%
WideResNet-28-8	96.06 ± 0.06	94.92 ± 0.13	95.64 ± 0.07	95.55 ± 0.12	95.98 ± 0.10	95%
WideResNet-28-8	96.06 ± 0.06	94.11 ± 0.19	95.31 ± 0.20	95.11 ± 0.07	95.84 ± 0.04	97.5%
WideResNet-28-8	96.06 ± 0.06	92.04 ± 0.11	94.38 ± 0.12	94.10 ± 0.12	95.63 ± 0.16	99%
WideResNet-28-8	96.06 ± 0.06	74.50 ± 2.23	\star	88.65 ± 0.36	91.76 ± 0.18	99.9%

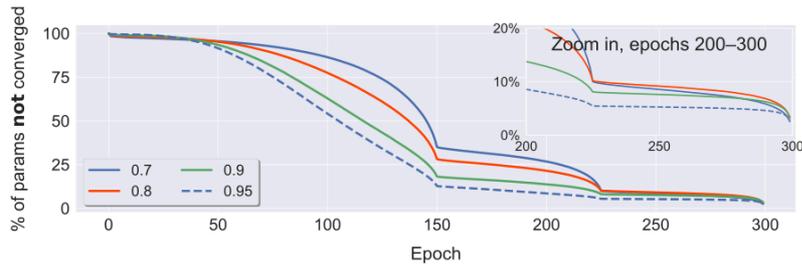


Figure 15.1 – Convergence of the pruning mask \mathbf{m}_t of DPF for various target sparsity levels (see legend). The y-axis represent the percentage of mask elements that still change **after** a certain epoch (x -axis). The illustrated example are from ResNet-20 on CIFAR-10. We decayed the learning rate at 150 and 225 epochs.

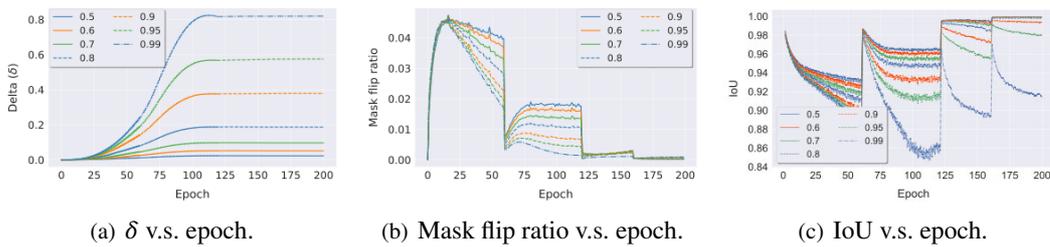


Figure 15.2 – Training dynamics of DPF (WideResNet-28-2 on CIFAR-10) for unstructured pruning with various sparsity ratios. IoU stands for Intersection over Union for the non-masked elements of two consecutive masks; the smaller value the more fraction of the masks will flip.

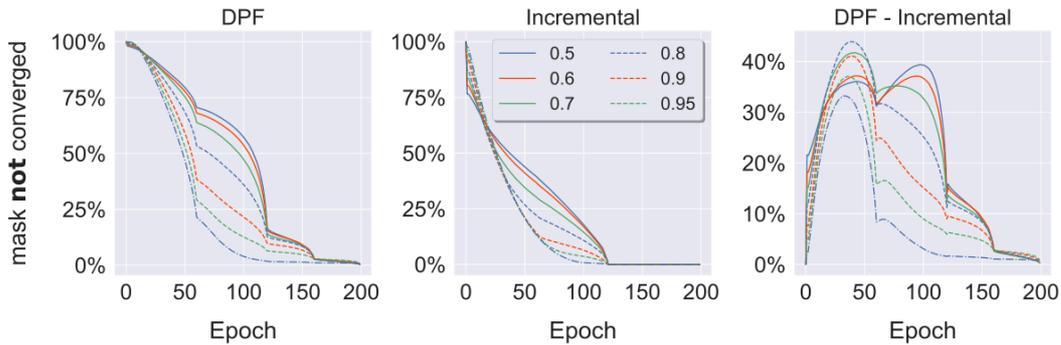


Figure 15.3 – Convergence of the pruning mask \mathbf{m}_t of DPF v.s. Incremental (Zhu and Gupta, 2017) for various target sparsity levels (see legend). The y-axis represent the percentage of mask elements that still change **after** a certain epoch (x -axis). The illustrated example are from WideResNet-28-2 on CIFAR-10. We decayed the learning rate at 60;120;160 epochs. These two schemes use the same gradual warmup schedule (and hyperparameters) for the pruning ratio during the training.

15.3.2 Understanding the Training Dynamics and Lottery Ticket Effect

Figure 15.1 and Figure 15.2 complements Figure 6.4, and details the training dynamics (e.g. the converge of δ and masks) of DPF from the other aspect. Figure 15.3 compares the training dynamics between DPF and Incremental (Zhu and Gupta, 2017), demonstrating the fact that our scheme enables a drastical reparameterization over the dense parameter space for better

15.3. Additional Results for Unstructured Pruning

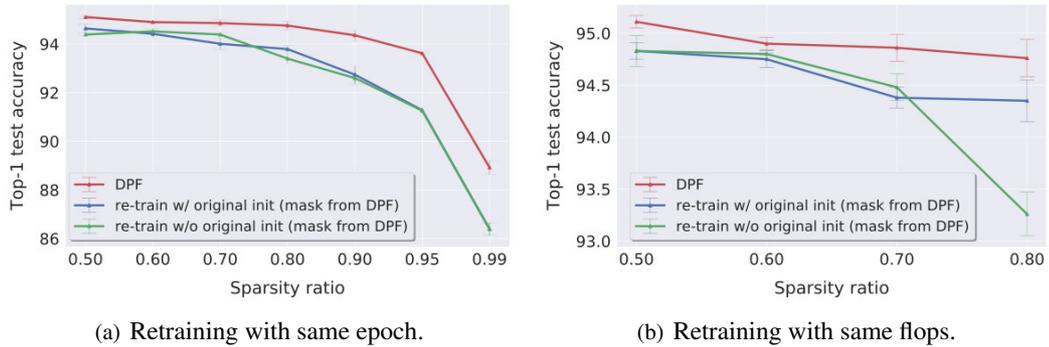


Figure 15.4 – Investigate the effect of lottery ticket for model compression (unstructured weight pruning for WideResNet28-2 on CIFAR-10). It complements the observations in Figure 6.5 by retraining the model for the same amount of computation budget (i.e. flops).

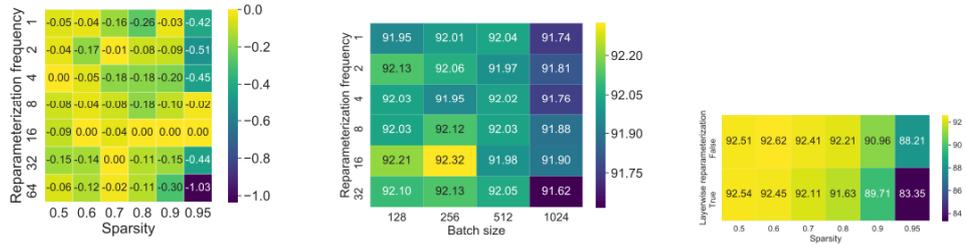
generalization performance.

Figure 15.4 in addition to the Figure 6.5 (in the main text) further studies the lottery ticket hypothesis under various training budgets (same epochs or same total flops). The results of DPF also demonstrate the importance of training-time structural exploration as well as the corresponding implicit regularization effects. Note that we do not want to question the importance of the weight initialization or the existence of the lottery ticket. Instead, our DPF can provide an alternative training scheme to compress the model to an extremely high compression ratio without sacrificing the test accuracy, where most of the existing methods still meet severe quality loss (including Frankle and Carbin (2019); Liu et al. (2019d); Frankle et al. (2019)).

15.3.3 Computational Overhead and the Impact of Hyperparameters

In Figure 15.5, we evaluated the top-1 test accuracy of a compressed model trained by DPF under various setups, e.g. diverse reparameterization periods p , several sparsity ratios, multiple mini-batch sizes, as well as whether layer-wise pruning or not. We can witness that the optimal reparameterization (i.e. $p = 16$) is quite consistent over the choices of sparsity ratios and mini-batch sizes, and we used it in all our experiments. The global-wise unstructured weight pruning (instead of layer-wise weight pruning) allows our DPF more flexible to perform dynamic parameter reallocation, and thus can provide better results especially for more aggressive pruning sparsity ratios. However, we also need to note that, for the same number of compressed parameters (layerwise or globalwise unstructured weight pruning), using global-wise pruning leads to a slight increase in the amount of MACs, as illustrated in Table 15.2.

Figure 15.6 demonstrates the trivial computational overhead of involving DPF to gradually train a compressed model (ResNet-50) from scratch (on ImageNet). Note that we evaluated the introduced reparameterization cost for dynamic pruning, which is independent of (potential) significant system speedup brought by the extreme high model sparsity. Even though our work did not estimate the practical speedup, we do believe we can have a similar training efficiency as



(a) Reparameterization period v.s. sparsity ratio. The heatmap value is the test accuracy minus the best accuracy of the corresponding sparsity. We use mini-batch size 128 w/o layerwise reparameterization. (b) Reparameterization period v.s. mini-batch size. The heatmap displays the test accuracy. We use sparsify ratio 0.80. (c) Reparameterization scheme (whether layerwise) v.s. sparsity ratio. The heatmap displays the test accuracy. We use mini-batch size 128 w/ reparameterization period $p = 16$.

Figure 15.5 – Investigate how the reparameterization period/scheme and mini-batch size impact the generalization performance (test top-1 accuracy), for dynamically training (and reparameterizing) a compressed model from scratch (ResNet-20 with CIFAR-10).

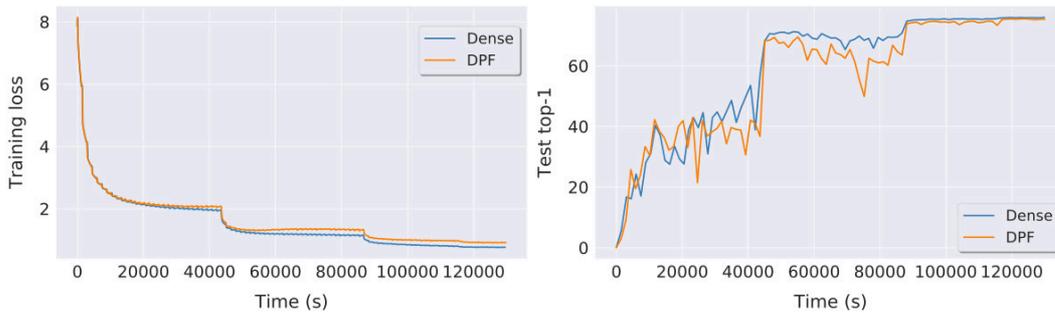


Figure 15.6 – The learning curve of our DPF (with unstructured magnitude pruning) and the standard mini-batch SGD for training ResNet50 on ImageNet. Our proposed DPF has trivial computational overhead. We trained ResNet50 on 4 NVIDIA V100 GPU’s with 1024 mini-batch size. The target sparsity ratio is 80%.

Table 15.2 – Investigate how the reparameterization scheme (layer-wise or not) impact the MACs (for the same number of compressed parameters), for using DPF on ResNet-20 with CIFAR-10.

Target sparsity	50%	60%	70%	80%	90%	95%
w/o layerwise reparameterization	22.80M	19.13M	15.18M	11.12M	6.54M	4.02M
w/ layerwise reparameterization	20.99M	16.91M	12.83M	8.75M	4.67M	2.63M

the values reported in [Dettmers and Zettlemoyer \(2019\)](#).

15.3.4 Implicit Neural Architecture Search

DPF can provide effective training-time structural exploration or even implicit neural network search. Figure 15.7 below demonstrates that for the same pruned model size (i.e. any point in the x -axis), we can always perform “architecture search” to get a better (in terms of generalization)

15.4. Additional Results for Structured Pruning

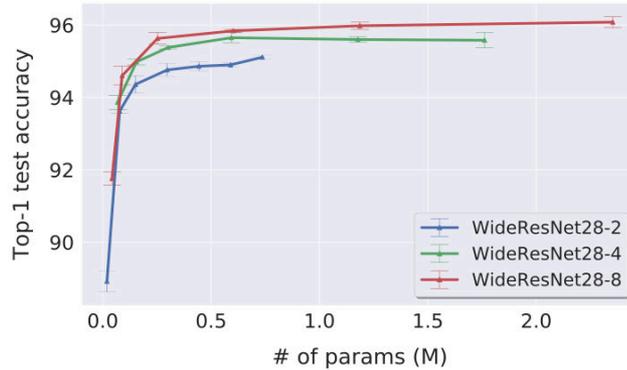


Figure 15.7 – Test top-1 accuracy v.s. the compressed model size, for training WideResNet-28 (with various widths) on CIFAR-10. The compressed model is searched from WideResNet-28 (fixed depth) with various width (number of filters per layer).

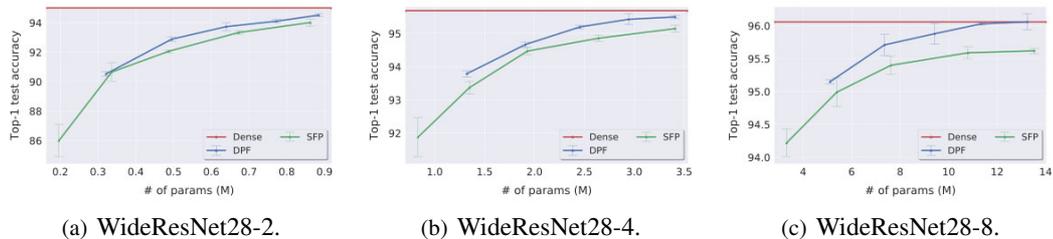


Figure 15.8 – # of params v.s. top-1 test accuracy, for training WideResNet28 (with various width) on CIFAR-10. Structured filter-wise pruning is used here. The reported results are averaged over three runs.

pruned model, from a larger network (e.g. WideResNet-28-8) rather than the one searched from a relatively small network (e.g. WideResNet-28-4).

15.4 Additional Results for Structured Pruning

15.4.1 Generalization Performance for CIFAR-10

Figure 15.8 complements the results of structured pruning in the main text (Figure 6.6), and Table 15.3 details the numerical results presented in both of Figure 6.6 and Figure 15.8.

15.4.2 Understanding the Lottery Ticket Effect

Similar to the observations in Section 6.7 (for unstructured pruning), Figure 15.9 instead considers structured pruning and again we found DPF does not find a lottery ticket. The superior generalization performance of DPF cannot be explained by the found mask or the weight initialization scheme.

Chapter 15. Appendix for DPF

Table 15.3 – Performance evaluation of DPF (and other baseline methods) for training (Wide)ResNet variants on CIFAR-10. We use the norm-based criteria for filter selection (as in SFP (He et al., 2018)) to estimate the output channel-wise pruning threshold. We follow the gradual pruning warmup scheme (as in Zhu and Gupta (2017)) from 0 epoch to the epoch when performing the second learning rate decay. Note that SFP prunes filters within the layer by a given ratio while our DPF prunes filters across layers. Due to the difference between filters for various layers, the # of parameters pruned by DPF might slight different from the one pruned by SFP. The pruning ratio refers to either prune filters within the layer or across the layers.

Model	Baseline on dense model	Methods		Target Pr. ratio
		SFP (H ⁺ , 2018)	DPF	
ResNet-20	92.48 ± 0.20	92.18 ± 0.31	92.54 ± 0.07	10%
ResNet-20	92.48 ± 0.20	91.12 ± 0.20	91.90 ± 0.06	20%
ResNet-20	92.48 ± 0.20	90.32 ± 0.25	91.07 ± 0.40	30%
ResNet-20	92.48 ± 0.20	89.60 ± 0.46	90.28 ± 0.26	40%
ResNet-32	93.52 ± 0.13	92.07 ± 0.22	92.18 ± 0.16	30%
ResNet-32	93.52 ± 0.13	91.14 ± 0.45	91.50 ± 0.21	40%
ResNet-56	94.51 ± 0.20	93.99 ± 0.27	94.53 ± 0.13	30%
ResNet-56	94.51 ± 0.20	93.57 ± 0.16	94.03 ± 0.38	40%
WideResNet-28-2	95.01 ± 0.04	94.02 ± 0.24	94.52 ± 0.08	40%
WideResNet-28-2	95.01 ± 0.04	93.34 ± 0.14	94.11 ± 0.12	50%
WideResNet-28-2	95.01 ± 0.04	92.07 ± 0.09	93.74 ± 0.25	60%
WideResNet-28-2	95.01 ± 0.04	90.66 ± 0.62	92.89 ± 0.16	70%
WideResNet-28-2	95.01 ± 0.04	86.00 ± 1.09	90.53 ± 0.17	80%
WideResNet-28-4	95.69 ± 0.10	95.15 ± 0.11	95.50 ± 0.05	40%
WideResNet-28-4	95.69 ± 0.10	94.86 ± 0.10	95.43 ± 0.16	50%
WideResNet-28-4	95.69 ± 0.10	94.47 ± 0.10	95.20 ± 0.05	60%
WideResNet-28-4	95.69 ± 0.10	93.37 ± 0.18	94.67 ± 0.08	70%
WideResNet-28-4	95.69 ± 0.10	91.88 ± 0.59	93.79 ± 0.09	80%
WideResNet-28-8	96.06 ± 0.06	95.62 ± 0.04	96.06 ± 0.12	40%
WideResNet-28-8	96.06 ± 0.06	95.59 ± 0.09	96.03 ± 0.02	50%
WideResNet-28-8	96.06 ± 0.06	95.40 ± 0.14	95.88 ± 0.16	60%
WideResNet-28-8	96.06 ± 0.06	94.99 ± 0.22	95.71 ± 0.16	70%
WideResNet-28-8	96.06 ± 0.06	94.22 ± 0.21	95.15 ± 0.03	80%

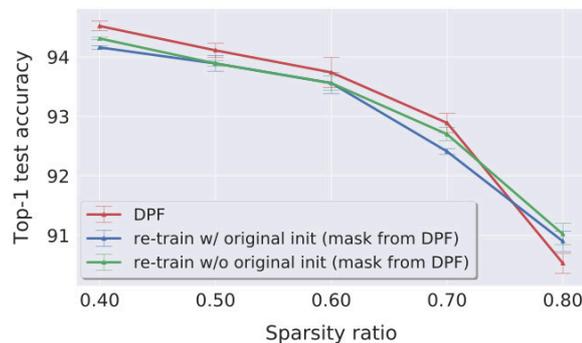


Figure 15.9 – Investigate the effect of lottery ticket for model compression (WideResNet28-2 with CIFAR-10) for structured pruning. We retrained the model with the mask from the model trained by DPF, by using the same epoch budget.

15.5 Missing Proofs

In this section we present the proofs for the claims in Section 6.5.

First, we give the proof for the strongly convex case. Here we follow [Lacoste-Julien et al. \(2012\)](#) for the general structure, combined with estimates from the error-feedback framework ([Stich et al., 2018](#); [Stich and Karimireddy, 2020](#)) to control the pruning errors.

Proof of Theorem 6.5.1. By definition of (DPF), $\mathbf{x}_{t+1} = \mathbf{x}_t - \gamma_t \mathbf{g}(\hat{\mathbf{x}}_t)$, hence,

$$\begin{aligned} \mathbb{E} \left[\|\mathbf{x}_{t+1} - \mathbf{x}^*\|^2 \mid \mathbf{x}_t \right] &= \|\mathbf{x}_t - \mathbf{x}^*\|^2 - 2\gamma_t \langle \mathbf{x}_t - \mathbf{x}^*, \mathbb{E} \mathbf{g}(\hat{\mathbf{x}}_t) \rangle + \gamma_t^2 \mathbb{E} \|\mathbf{g}(\hat{\mathbf{x}}_t)\|^2 \\ &\leq \|\mathbf{x}_t - \mathbf{x}^*\|^2 - 2\gamma_t \langle \mathbf{x}_t - \mathbf{x}^*, \nabla f(\hat{\mathbf{x}}_t) \rangle + \gamma_t^2 G^2 \\ &= \|\mathbf{x}_t - \mathbf{x}^*\|^2 - 2\gamma_t \langle \hat{\mathbf{x}}_t - \mathbf{x}^*, \nabla f(\hat{\mathbf{x}}_t) \rangle + \gamma_t^2 G^2 \\ &\quad + 2\gamma_t \langle \hat{\mathbf{x}}_t - \mathbf{x}_t, \nabla f(\hat{\mathbf{x}}_t) \rangle. \end{aligned}$$

By strong convexity,

$$-2 \langle \hat{\mathbf{x}}_t - \mathbf{x}^*, \nabla f(\hat{\mathbf{x}}_t) \rangle \leq -\mu \|\hat{\mathbf{x}}_t - \mathbf{x}^*\|^2 - 2(f(\hat{\mathbf{x}}_t) - f(\mathbf{x}^*)),$$

and with $\|\mathbf{a} + \mathbf{b}\|^2 \leq 2\|\mathbf{a}\|^2 + 2\|\mathbf{b}\|^2$ further

$$-\|\hat{\mathbf{x}}_t - \mathbf{x}^*\|^2 \leq -\frac{1}{2} \|\mathbf{x}_t - \mathbf{x}^*\|^2 + \|\mathbf{x}_t - \hat{\mathbf{x}}_t\|^2$$

and with $\langle \mathbf{a}, \mathbf{b} \rangle \leq \frac{1}{2\alpha} \|\mathbf{a}\|^2 + \frac{\alpha}{2} \|\mathbf{b}\|^2$ for $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$ and $\alpha > 0$,

$$\begin{aligned} 2 \langle \hat{\mathbf{x}}_t - \mathbf{x}_t, \nabla f(\hat{\mathbf{x}}_t) \rangle &\leq 2L \|\hat{\mathbf{x}}_t - \mathbf{x}_t\|^2 + \frac{1}{2L} \|\nabla f(\hat{\mathbf{x}}_t)\|^2 \\ &= 2L \|\hat{\mathbf{x}}_t - \mathbf{x}_t\|^2 + \frac{1}{2L} \|\nabla f(\hat{\mathbf{x}}_t) - \nabla f(\mathbf{x}^*)\|^2 \\ &\leq 2L \|\hat{\mathbf{x}}_t - \mathbf{x}_t\|^2 + f(\hat{\mathbf{x}}_t) - f(\mathbf{x}^*), \end{aligned}$$

where the last inequality is a consequence of L -smoothness. Combining all these inequalities yields

$$\begin{aligned} \mathbb{E} \left[\|\mathbf{x}_{t+1} - \mathbf{x}^*\|^2 \mid \mathbf{x}_t \right] &\leq \left(1 - \frac{\mu\gamma_t}{2}\right) \|\mathbf{x}_t - \mathbf{x}^*\|^2 - \gamma_t (f(\hat{\mathbf{x}}_t) - f(\mathbf{x}^*)) + \gamma_t^2 G^2 \\ &\quad + \gamma_t (2L + \mu) \|\hat{\mathbf{x}}_t - \mathbf{x}_t\|^2 \\ &\leq \left(1 - \frac{\mu\gamma_t}{2}\right) \|\mathbf{x}_t - \mathbf{x}^*\|^2 - \gamma_t (f(\hat{\mathbf{x}}_t) - f(\mathbf{x}^*)) + \gamma_t^2 G^2 \\ &\quad + 3\gamma_t L \|\hat{\mathbf{x}}_t - \mathbf{x}_t\|^2. \end{aligned}$$

as $\mu \leq L$. Hence, by rearranging and multiplying with a weight $\lambda_t > 0$:

$$\lambda_t \mathbb{E} (f(\hat{\mathbf{x}}_t) - f(\mathbf{x}^*)) \leq \frac{\lambda_t (1 - \mu\gamma_t/2)}{\gamma_t} \mathbb{E} \|\mathbf{x}_t - \mathbf{x}^*\|^2 - \frac{\lambda_t}{\gamma_t} \mathbb{E} \|\mathbf{x}_{t+1} - \mathbf{x}^*\|^2 + \gamma_t \lambda_t G^2 + 3\lambda_t L \mathbb{E} \|\hat{\mathbf{x}}_t - \mathbf{x}_t\|^2.$$

By plugging in the learning rate, $\gamma_t = \frac{4}{\mu(t+2)}$ and setting $\lambda_t = (t+1)$ we obtain

$$\begin{aligned} \lambda_t \mathbb{E}(f(\hat{\mathbf{x}}_t) - f(\mathbf{x}^*)) &\leq \frac{\mu}{4} \left[t(t+1) \mathbb{E} \|\mathbf{x}_t - \mathbf{x}^*\|^2 - (t+1)(t+2) \mathbb{E} \|\mathbf{x}_{t+1} - \mathbf{x}^*\|^2 \right] \\ &\quad + \frac{4(t+1)}{\mu(t+2)} G^2 + 3(t+1)L \mathbb{E} \|\hat{\mathbf{x}}_t - \mathbf{x}_t\|^2. \end{aligned}$$

By summing from $t=0$ to $t=T$ these λ_t -weighted inequalities, we obtain a telescoping sum:

$$\begin{aligned} \sum_{t=0}^T \lambda_t \mathbb{E}(f(\hat{\mathbf{x}}_t) - f(\mathbf{x}^*)) &\leq \frac{\mu}{4} \left[0 - (T+1)(T+2) \mathbb{E} \|\mathbf{x}_{T+1} - \mathbf{x}^*\|^2 \right] + \frac{4(T+1)}{\mu} G^2 \\ &\quad + 3L \sum_{t=0}^T \lambda_t \mathbb{E} \|\hat{\mathbf{x}}_t - \mathbf{x}_t\|^2 \\ &\leq \frac{4(T+1)}{\mu} G^2 + 3L \sum_{t=0}^T \lambda_t \mathbb{E} \|\hat{\mathbf{x}}_t - \mathbf{x}_t\|^2. \end{aligned}$$

Hence, for $\Lambda_T := \sum_{t=0}^T \lambda_t = \frac{(T+1)(T+2)}{2}$,

$$\begin{aligned} \frac{1}{\Lambda_T} \sum_{t=0}^T \lambda_t \mathbb{E}(f(\hat{\mathbf{x}}_t) - f(\mathbf{x}^*)) &\leq \frac{4(T+1)}{\mu \Lambda_T} G^2 + \frac{3L}{\Lambda_T} \sum_{t=0}^T \lambda_t \mathbb{E} \|\hat{\mathbf{x}}_t - \mathbf{x}_t\|^2 \\ &= \mathcal{O} \left(\frac{G^2}{\mu T} + \frac{L}{\Lambda_T} \sum_{t=0}^T \lambda_t \mathbb{E} \|\hat{\mathbf{x}}_t - \mathbf{x}_t\|^2 \right). \end{aligned}$$

Finally, using $\|\hat{\mathbf{x}}_t - \mathbf{x}_t\|^2 = \delta_t \|\mathbf{x}_t\|^2$ by (6.1), shows the theorem. \square

Before giving the proof of Theorem 6.5.2, we first give a justification for the remark just below Theorem 6.5.1 on the one-shot pruning of the final iterate.

We have by L -smoothness and $\langle \mathbf{a}, \mathbf{b} \rangle \leq \frac{1}{2\alpha} \|\mathbf{a}\|^2 + \frac{\alpha}{2} \|\mathbf{b}\|^2$ for $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$ and $\alpha > 0$ for any iterate \mathbf{x}_t :

$$\begin{aligned} f(\hat{\mathbf{x}}_t) - f(\mathbf{x}^*) &\leq f(\mathbf{x}_t) - f(\mathbf{x}^*) + \langle \nabla f(\mathbf{x}_t), \hat{\mathbf{x}}_t - \mathbf{x}_t \rangle + \frac{L}{2} \|\hat{\mathbf{x}}_t - \mathbf{x}_t\|^2 \\ &\leq f(\mathbf{x}_t) - f(\mathbf{x}^*) + \frac{1}{2L} \|\nabla f(\mathbf{x}_t)\|^2 + L \|\hat{\mathbf{x}}_t - \mathbf{x}_t\|^2 \\ &\leq 2(f(\mathbf{x}_t) - f(\mathbf{x}^*)) + \delta_t L \|\mathbf{x}_t\|^2. \end{aligned} \tag{15.1}$$

Furthermore, again by L -smoothness,

$$f(\mathbf{x}_T) - f(\mathbf{x}^*) \leq \frac{L}{2} \|\mathbf{x}_T - \mathbf{x}^*\|^2 = \mathcal{O} \left(\frac{LG^2}{\mu^2 T} \right)$$

as standard SGD analysis gives the estimate $\mathbb{E} \|\mathbf{x}_T - \mathbf{x}^*\|^2 = \mathcal{O} \left(\frac{G^2}{\mu^2 T} \right)$, see e.g. [Lacoste-Julien et al. \(2012\)](#). Combining these two estimates (with $\mathbf{x}_t = \mathbf{x}_T$) shows the claim.

Furthermore, we also claimed that also the dense model converges to a neighborhood of optimal solution. This follows by L -smoothness and (15.1): For any fixed model \mathbf{x}_t we have the

Chapter 15. Appendix for DPF

estimate (15.1), hence for a randomly chosen (dense) model \mathbf{u} (from the same distribution as the sparse model in Theorem 6.5.1) we have

$$\mathbb{E}f(\mathbf{u}) - f(\mathbf{x}^*) \stackrel{(15.1)}{\leq} 2\mathbb{E}[f(\hat{\mathbf{u}}) - f(\mathbf{x}^*)] + L\mathbb{E}[\delta_t \|\mathbf{w}_t\|^2] \stackrel{(\text{Thm 6.5.1})}{=} \mathcal{O}\left(\frac{G^2}{\mu T} + L\mathbb{E}[\delta_t \|\mathbf{w}_t\|^2]\right).$$

Lastly, we give the proof of Theorem 6.5.2, following [Karimireddy et al. \(2019\)](#).

Proof of Theorem 6.5.2. By smoothness, and $\langle \mathbf{a}, \mathbf{b} \rangle \leq \frac{1}{2}\|\mathbf{a}\|^2 + \frac{1}{2}\|\mathbf{b}\|^2$ for $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$,

$$\begin{aligned} \mathbb{E}[f(\mathbf{x}_{t+1}) | \mathbf{x}_t] &\leq f(\mathbf{x}_t) - \gamma \langle \nabla f(\mathbf{x}_t), \mathbb{E}\mathbf{g}(\hat{\mathbf{x}}_t) \rangle + \gamma^2 \frac{L}{2} \mathbb{E}\|\mathbf{g}(\hat{\mathbf{x}}_t)\|^2 \\ &\leq f(\mathbf{x}_t) - \gamma \langle \nabla f(\mathbf{x}_t), \nabla f(\hat{\mathbf{x}}_t) \rangle + \gamma^2 \frac{LG^2}{2} \\ &= f(\mathbf{x}_t) - \gamma \langle \nabla f(\hat{\mathbf{x}}_t), \nabla f(\hat{\mathbf{x}}_t) \rangle + \gamma^2 \frac{LG^2}{2} \\ &\quad + \gamma \langle \nabla f(\hat{\mathbf{x}}_t) - \nabla f(\mathbf{x}_t), \nabla f(\hat{\mathbf{x}}_t) \rangle \\ &\leq f(\mathbf{x}_t) - \gamma \|\nabla f(\hat{\mathbf{x}}_t)\|^2 + \gamma^2 \frac{LG^2}{2} \\ &\quad + \frac{\gamma}{2} \|\nabla f(\hat{\mathbf{x}}_t) - \nabla f(\mathbf{x}_t)\|^2 + \frac{\gamma}{2} \|\nabla f(\hat{\mathbf{x}}_t)\|^2 \\ &\leq f(\mathbf{x}_t) - \frac{\gamma}{2} \|\nabla f(\hat{\mathbf{x}}_t)\|^2 + \gamma^2 \frac{LG^2}{2} + \frac{\gamma L^2}{2} \|\mathbf{x}_t - \hat{\mathbf{x}}_t\|^2, \end{aligned}$$

and by rearranging

$$\mathbb{E}\|\nabla f(\hat{\mathbf{x}}_t)\|^2 \leq \frac{2}{\gamma} [\mathbb{E}f(\mathbf{x}_t) - \mathbb{E}f(\mathbf{x}_{t+1})] + \gamma LG^2 + L^2 \mathbb{E}\|\mathbf{x}_t - \hat{\mathbf{x}}_t\|^2.$$

Summing these inequalities from $t = 0$ to $t = T$ gives

$$\begin{aligned} \frac{1}{T+1} \sum_{t=0}^T \mathbb{E}\|\nabla f(\hat{\mathbf{x}}_t)\|^2 &\leq \frac{2}{\gamma(T+1)} \sum_{t=0}^T (\mathbb{E}[f(\mathbf{x}_t)] - \mathbb{E}[f(\mathbf{x}_{t+1})]) + \gamma LG^2 \\ &\quad + \frac{L^2}{T+1} \sum_{t=0}^T \mathbb{E}\|\mathbf{x}_t - \hat{\mathbf{x}}_t\|^2 \\ &\leq \frac{2(f(\mathbf{x}_0) - f(\mathbf{x}^*))}{\gamma(T+1)} + L\gamma G^2 + \frac{L^2}{T+1} \sum_{t=0}^T \mathbb{E}\|\mathbf{e}_t\|^2. \end{aligned}$$

Finally, using $\|\hat{\mathbf{x}}_t - \mathbf{x}_t\|^2 = \delta_t \|\mathbf{x}_t\|^2$ by (6.1), and plugging in the stepsize γ that minimizes the right hand side shows the claim. \square

16 Appendix for Masking as an Efficient Alternative to Fine-tuning

16.1 Reproducibility Checklist

16.1.1 Computing Infrastructure

All experiments are conducted on following GPU models: Tesla V100, GeForce GTX 1080 Ti, and GeForce GTX 1080. We use per-GPU batch size 32. Thus, experiments comparing masking and fine-tuning on QNLI and AG take 4 GPUs and all the other tasks use a single GPU.

16.1.2 Number of Parameters

In Section 7.6.3 we thoroughly compare the number of parameters and memory consumption of fine-tuning and masking. Numerical values are in Table 16.5.

16.1.3 Validation Performance

The dev set performance of Table 7.2 is covered in Table 7.1. We report Matthew's correlation coefficient (MCC) for CoLA, micro-F1 for NER, and accuracy for the other tasks. We use the evaluation functions in `scikit-learn` (Pedregosa et al., 2011) and `segeval` (<https://github.com/chakki-works/segeval>).

16.1.4 Hyperparameter Search

The only hyperparameter we searched is learning rate, for both masking and fine-tuning, according to the setup discussion in Section 7.5. The optimal values are in Table 16.1.

Chapter 16. Appendix for Masking as an Efficient Alternative to Fine-tuning

Table 16.1 – The optimal learning rate on various tasks for BERT/RobERTa/DistilBERT. We perform fine-tuning/masking on all tasks for 10 epochs with early stopping of 2 epochs.

		MRPC	SST2	CoLA	RTE	QNLI	POS	NER	SWAG	SEM	TREC	AG
BERT	Fine-tuning	5e-5	1e-5	3e-5	5e-5	3e-5	3e-5	3e-5	7e-5	1e-5	3e-5	3e-5
	Masking	1e-3	5e-4	9e-4	1e-3	7e-4	5e-4	7e-4	1e-4	7e-5	1e-4	5e-4
RoBERTa	Fine-tuning	3e-5	1e-5	1e-5	7e-6	1e-5	9e-6	3e-5	1e-5	7e-6	9e-6	3e-5
	Masking	3e-4	9e-5	3e-4	3e-4	1e-4	3e-4	3e-4	1e-4	3e-4	5e-4	5e-4
DistilBERT	Fine-tuning	3e-5	7e-5	3e-5	3e-5	3e-5	3e-5	1e-5	7e-6	1e-5	3e-5	3e-5
	Masking	9e-4	7e-4	9e-4	9e-4	1e-3	7e-4	7e-4	3e-4	3e-4	9e-4	1e-3

Table 16.2 – Number of examples in dev and test per task. For POS and NER, we report the number of words.

	Dev	Test
MRPC	408	n/a
SST2	872	n/a
CoLA	1,042	n/a
RTE	277	n/a
QNLI	5,732	n/a
SEM	1,325	10,551
TREC	548	500
AG	24,000	7,600
POS	135,105	133,082
NER	51,341	46,425
SWAG	20,006	n/a

16.1.5 Datasets

For GLUE tasks, we use the official datasets from the benchmark <https://gluebenchmark.com/>. For TREC and AG, we download the datasets developed by [Zhang et al. \(2015\)](#), which are available at here. Note that this link is provided by [Zhang et al. \(2015\)](#) and also used by [Sun et al. \(2019\)](#). For SEM, we obtain the dataset from the official SemEval website: <http://alt.qcri.org/semeval2016/task4/>. For NER, we use the official dataset: <https://www.clips.uantwerpen.be/conll2003/ner/>. We obtain our POS dataset from the linguistic data consortium (LDC). We use the official dataset of SWAG ([Zellers et al., 2018](#)): <https://github.com/rowanz/swagaf/tree/master/data>.

For POS, sections 0-18 of WSJ are train, sections 19-21 are dev, and sections 22-24 are test ([Collins, 2002](#)). We use the official train/dev/test splits of all the other datasets.

To preprocess the datasets, we use the tokenizers provided by the `Transformers` package ([Wolf et al., 2019](#)) to convert the raw dataset to the formats required by BERT/RobERTa/DistilBERT. Since wordpiece tokenization is used, there is no out-of-vocabulary words.

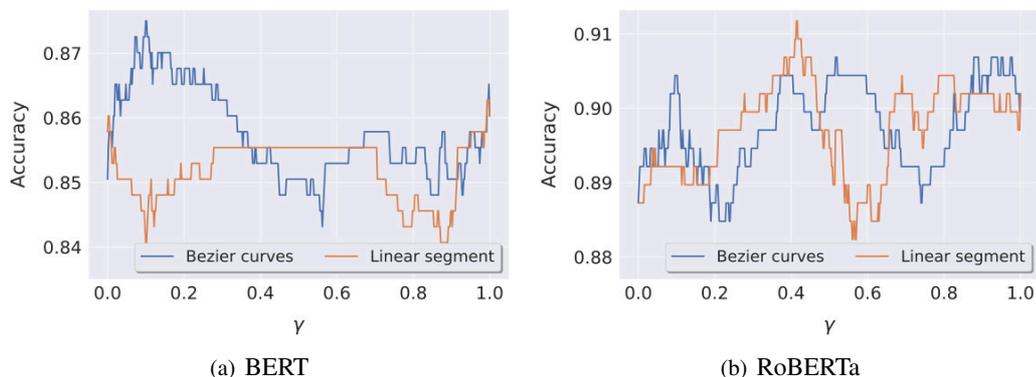


Figure 16.1 – The accuracy on MRPC dev set, as a function of the point on the curves $\phi_{\theta}(\gamma)$, connecting the two minima found by fine-tuning (left, $\gamma=0$) and masking (right, $\gamma=1$).

Since we use a maximum sequence length of 128, our preprocessing steps exclude some word-tag annotations in POS and NER. For POS, after wordpiece tokenization, we see 1 sentence in dev and 2 sentences in test have more than 126 (the [CLS] and [SEP] need to be considered) wordpieces. As a result, we exclude 5 annotated words in dev and 87 annotated words in test. Similarly, for NER (which is also formulated as a tagging task following [Devlin et al. \(2019\)](#)), we see 3 sentences in dev and 1 sentence in test have more than 126 wordpieces. As a result, we exclude 27 annotated words in dev and 8 annotated words in test.

The number of examples in dev and test per task is shown in following Table 16.2.

16.2 More on Mode Connectivity

Following the mode connectivity framework proposed in [Garipov et al. \(2018\)](#), we parameterize the path joining two minima using a Bézier curve. Let \mathbf{x}_0 and \mathbf{x}_{n+1} be the parameters of the models trained from fine-tuning and masking. Then, an n -bend Bézier curve connecting \mathbf{x}_0 and \mathbf{x}_{n+1} , with n trainable intermediate models $\theta = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$, can be represented by $\phi_{\theta}(t)$, such that $\phi_{\theta}(0) = \mathbf{x}_0$ and $\phi_{\theta}(1) = \mathbf{x}_{n+1}$, and

$$\phi_{\theta}(t) = \sum_{i=0}^{n+1} \binom{n+1}{i} (1-t)^{n+1-i} t^i \mathbf{x}_i.$$

We train a 3-bend Bézier curve by minimizing the loss $\mathbb{E}_{t \sim U[0,1]} \mathcal{L}(\phi_{\theta}(t))$, where $U[0,1]$ is the uniform distribution in the interval $[0,1]$. Monte Carlo method is used to estimate the gradient of this expectation-based function and gradient-based optimization is used for the minimization. The results are illustrated in Figure 16.1. Masking implicitly performs gradient descent, analogy to the weights update achieved by fine-tuning; the observations complement our arguments in the main text.

16.3 More Empirical Results

Ensemble results of RoBERTa and DistilBERT. Following Table 16.3 shows the single and ensemble results of RoBERTa and DistilBERT on the test set of SEM, TREC, AG, POS, and NER.

Table 16.3 – Error rate (%) on test set of tasks by RoBERTa and DistilBERT. Single: the averaged performance of four models with varied random seeds. Ensemble: ensemble of the four models.

			SEM	TREC	AG	POS	NER
RoBERTa	Masking	Single	11.12	3.15	5.06	2.11	11.03
		Ensemble	10.54	2.40	4.55	2.11	10.57
	Fine-tuning	Single	10.74	3.00	5.10	2.00	10.43
		Ensemble	10.74	2.60	4.50	1.96	9.54
DistilBERT	Masking	Single	11.89	3.70	5.71	2.39	10.40
		Ensemble	11.60	3.00	5.29	2.54	9.86
	Fine-tuning	Single	11.94	3.30	5.42	2.39	10.18
		Ensemble	11.48	3.00	4.84	2.29	9.74

Table 16.4 – Numerical value of the layer-wise behavior experiment. We train for 10 epochs with mini-batch size 32. The learning rate is finetuned using the mean results on four various random seeds.

	MRPC	RTE	CoLA
Fine-tuning (BERT + classifier)	0.861 ± 0.008	0.692 ± 0.027	0.596 ± 0.015
Masking (BERT 00-11 + classifier, initial sparsity 5%)	0.862 ± 0.015	0.673 ± 0.036	0.592 ± 0.004
Masking (BERT 00-11 + classifier, initial sparsity 15%)	0.825 ± 0.039	0.626 ± 0.040	0.522 ± 0.027
Masking (BERT 02-11 + classifier, initial sparsity 5%)	0.868 ± 0.011	0.695 ± 0.030	0.595 ± 0.010
Masking (BERT 02-11 + classifier, initial sparsity 15%)	0.844 ± 0.024	0.662 ± 0.021	0.556 ± 0.012
Masking (BERT 04-11 + classifier, initial sparsity 5%)	0.861 ± 0.004	0.705 ± 0.037	0.583 ± 0.005
Masking (BERT 04-11 + classifier, initial sparsity 15%)	0.861 ± 0.009	0.669 ± 0.014	0.553 ± 0.014
Masking (BERT 06-11 + classifier, initial sparsity 5%)	0.862 ± 0.004	0.696 ± 0.027	0.551 ± 0.006
Masking (BERT 06-11 + classifier, initial sparsity 15%)	0.868 ± 0.008	0.691 ± 0.033	0.534 ± 0.016
Masking (BERT 08-11 + classifier, initial sparsity 5%)	0.848 ± 0.016	0.675 ± 0.034	0.538 ± 0.014
Masking (BERT 08-11 + classifier, initial sparsity 15%)	0.851 ± 0.009	0.688 ± 0.022	0.545 ± 0.005
Masking (BERT 00-09 + classifier, initial sparsity 5%)	0.859 ± 0.012	0.683 ± 0.031	0.589 ± 0.011
Masking (BERT 00-09 + classifier, initial sparsity 15%)	0.820 ± 0.052	0.604 ± 0.021	0.514 ± 0.016
Masking (BERT 00-07 + classifier, initial sparsity 5%)	0.829 ± 0.032	0.649 ± 0.053	0.574 ± 0.012
Masking (BERT 00-07 + classifier, initial sparsity 15%)	0.807 ± 0.042	0.600 ± 0.027	0.509 ± 0.004
Masking (BERT 00-05 + classifier, initial sparsity 5%)	0.814 ± 0.033	0.632 ± 0.058	0.565 ± 0.027
Masking (BERT 00-05 + classifier, initial sparsity 15%)	0.781 ± 0.032	0.567 ± 0.030	0.510 ± 0.025
Masking (BERT 00-03 + classifier, initial sparsity 5%)	0.791 ± 0.026	0.606 ± 0.027	0.535 ± 0.034
Masking (BERT 00-03 + classifier, initial sparsity 15%)	0.776 ± 0.035	0.600 ± 0.019	0.527 ± 0.014

16.4 Numerical Values of Plots

16.4.1 Layer-wise Behaviors

Table 16.4 details the numerical values of Figure 7.2.

16.4.2 Memory Consumption

Table 16.5 details the numerical values of Figure 7.3.

	Number of Parameters		Memory Usage (Kilobytes)	
	Fine-tuning	Masking	Fine-tuning	Masking
Pretrained	109,482,240		437,928.96	
MRPC	+ 1,536	+ 1,536 + 71,368,704 + 1,536	+ 6.144	+ 6.144 + 8,921.088 + 0.192
SST2	+ 1,536 + 109,482,240	+ 71,368,704 + 1,536	+ 6.144 + 437,928.96	+ 8,921.088 + 0.192
CoLA	+ 1,536 + 109,482,240	+ 71,368,704 + 1,536	+ 6.144 + 437,928.96	+ 8,921.088 + 0.192
RTE	+ 1,536 + 109,482,240	+ 71,368,704 + 1,536	+ 6.144 + 437,928.96	+ 8,921.088 + 0.192
QNLI	+ 1,536 + 109,482,240	+ 71,368,704 + 1,536	+ 6.144 + 437,928.96	+ 8,921.088 + 0.192
SEM	+ 1,536 + 109,482,240	+ 71,368,704 + 1,536	+ 6.144 + 437,928.96	+ 8,921.088 + 0.192
TREC	+ 4,608 + 109,482,240	+ 4,608 + 71,368,704 + 4,608	+ 18.432 + 437,928.96	+ 18.432 + 8,921.088 + 0.576
AG	+ 3,072 + 109,482,240	+ 3,072 + 71,368,704 + 3,072	+ 12.288 + 437,928.96	+ 12.288 + 8,921.088 + 0.384
POS	+ 37,632 + 109,482,240	+ 37,632 + 71,368,704 + 37,632	+ 150.528 + 437,928.96	+ 150.528 + 8,921.088 + 4.704
NER	+ 6,912 + 109,482,240	+ 6,912 + 71,368,704 + 6,912	+ 27.648 + 437,928.96	+ 27.648 + 8,921.088 + 0.864
SWAG	+ 768 + 109,482,240	+ 768 + 71,368,704 + 768	+ 3.072 + 437,928.96	+ 3.072 + 8,921.088 + 0.096

Table 16.5 – Model size comparison when applying masking and fine-tuning. Numbers are based on BERT-base-uncased. Note that our masking scheme enables sharing parameters across tasks: tasks with the same number of output dimension can use the same classifier layer.

17 Appendix for FedDF

17.1 Detailed Related Work Discussion

Prior work. We first comment on the two close approaches (FedMD and Cronus), in order to address 1) Distinctions between FedDF and prior work, 2) Privacy/Communication traffic concerns, 3) Omitted experiments on FedMD and Cronus.

- Distinctions between FedDF and prior work. As discussed in the related work, most SOTA FL methods directly manipulate received model parameters (e.g. FedAvg/FedAvgM/FedMA). To our best knowledge, FedMD and Cronus are the only two that utilize logits information (of neural nets) for FL. The distinctions from them are made below.
- Various objectives and evaluation metrics. Cronus is designed for robust FL under poisoning attack, whereas FedMD is for personalized FL. In contrast, FedDF is intended for on-server model aggregation (evaluation on the aggregated model), whereas neither FedMD nor Cronus aggregates the model on the server.
- Various Operations.
 1. FedDF, like FedAvg, *only* exchanges models between the server and clients, without transmitting input data. In contrast, FedMD and Cronus rely on exchanging public data logits. As FedAvg, FedDF can include privacy/security extensions and has the same communication cost per round.
 2. FedDF performs ensemble distillation with unlabeled data *on the server*. In contrast, FedMD/Cronus use averaged logits received from the server for *local client training*.
- Omitted experiments with FedMD/Cronus.
 1. FedMD requires to locally pre-train on the *labeled* public data, thus the model classifier necessitates an output dimension of # of public classes *plus* # of private classes (c.f. the output dimension of # of private classes in other FL methods). We cannot compare FedMD with FedDF with the same architecture (classifier) to ensure fairness.
 2. Cronus is shown to be consistently worse than FedAvg in normal FL (i.e. no attack case) in their Table IV & VI.
 3. Various objectives/metrics argued above. We thoroughly evaluated SOTA baselines with

the same objective/metric.

Contemporaneous work. We then detail some contemporaneous work, e.g. Sun and Lyu (2021); Chen and Chao (2021); Zhou et al. (2020); He et al. (2020a). Sun and Lyu (2021) slightly extends FedMD by adding differential privacy. In Zhou et al. (2020), the server aggregates the synthetic data distilled from clients’ private dataset, which in turn uses for one-shot on-server learning. He et al. (2020a) improve FL for resource-constrained edge devices by combing FL with Split Learning (SL) and knowledge distillation: edge devices train compact feature extractor through local SGD and then synchronize extracted features and logits with the server, while the server (asynchronously) uses the latest received features and logits to train a much larger server-side CNN. The knowledge distillation is used on both the server and clients to improve the optimization quality.

FedBE (Chen and Chao, 2021) is very similar to us, where it resorts to stochastic weight average-Gaussian (SWAG) (Maddox et al., 2019) and the ensemble distillation is achieved via cyclical learning rate schedule with SWA (Izmailov et al., 2018). In Table 17.3 below, we empirically compare our FedDF with this contemporaneous work (i.e. FedBE).

17.2 Algorithmic Description

Algorithm 17 below details a general training procedure on local clients. The local update step of FEDPROX corresponds to adding a proximal term (i.e. $\eta \frac{\partial^2 \|\mathbf{x}_t^k - \mathbf{x}_{t-1}^k\|_2^2}{\partial \mathbf{x}_t^k}$) to line 5.

Algorithm 17 Illustration of local client update in FEDAVG. The K clients are indexed by k ; \mathcal{P}_k indicates the set of indexes of data points on client k , and $n_k = |\mathcal{P}_k|$. E is the number of local epochs, and η is the learning rate. ℓ evaluates the loss on model weights for a mini-batch of an arbitrary size.

```

1: procedure CLIENT-LOCALUPDATE( $k, \mathbf{x}_{t-1}^k$ )
2:   Client  $k$  receives  $\mathbf{x}_{t-1}^k$  from server and copies it as  $\mathbf{x}_t^k$ 
3:   for each local epoch  $i$  from 1 to  $E$  do
4:     for mini-batch  $b \subset \mathcal{P}_k$  do
5:        $\mathbf{x}_t^k \leftarrow \mathbf{x}_t^k - \eta \frac{\partial \ell(\mathbf{x}_t^k; b)}{\partial \mathbf{x}_t^k}$  ▷ can be arbitrary optimizers (e.g. Adam)
6:   return  $\mathbf{x}_t^k$  to server

```

Algorithm 18 illustrates the model fusion of FedDF for the FL system with heterogeneous model prototypes. The schematic diagram is presented in Figure 17.1. To perform model fusion in such heterogeneous scenarios, FedDF constructs several prototypical models on the server. Each prototype represents all clients with identical architecture/size/precision etc.

Algorithm 18 Illustration of FedDF for heterogeneous FL systems. The K clients are indexed by k , and n_k indicates the number of data points for the k -th client. The number of communication rounds is T , and C controls the client participation ratio per communication round. The number of total iterations used for model fusion is denoted as N . The distinct model prototype set \mathcal{P} has p model prototypes, with each initialized as \mathbf{x}_0^P .

```

1: procedure SERVER
2:   initialize HashMap  $\mathcal{M}$ : map each model prototype  $P$  to its weights  $\mathbf{x}_0^P$ .
3:   initialize HashMap  $\mathcal{C}$ : map each client to its model prototype.
4:   initialize HashMap  $\tilde{\mathcal{C}}$ : map each model prototype to the associated clients.
5:   for each communication round  $t = 1, \dots, T$  do
6:      $\mathcal{S}_t \leftarrow$  a random subset ( $C$  fraction) of the  $K$  clients
7:     for each client  $k \in \mathcal{S}_t$  in parallel do
8:        $\hat{\mathbf{x}}_t^k \leftarrow$  Client-LocalUpdate( $k, \mathcal{M}[\mathcal{C}[k]]$ ) ▷ detailed in Algorithm 17.
9:     for each prototype  $P \in \mathcal{P}$  in parallel do
10:      initialize the client set  $\mathcal{S}_t^P$  with model prototype  $P$ , where  $\mathcal{S}_t^P \leftarrow \tilde{\mathcal{C}}[P] \cap \mathcal{S}_t$ 
11:      initialize for model fusion  $\mathbf{x}_{t,0}^P \leftarrow \sum_{k \in \mathcal{S}_t^P} \frac{n_k}{\sum_{k \in \mathcal{S}_t^P} n_k} \hat{\mathbf{x}}_t^k$ 
12:      for  $j$  in  $\{1, \dots, N\}$  do
13:        sample  $\mathbf{d}$ , from e.g. (1) an unlabeled dataset, (2) a generator
14:        use ensemble of  $\{\hat{\mathbf{x}}_t^k\}_{k \in \mathcal{S}_t^P}$  to update server student  $\mathbf{x}_{t,j}^P$  through AVGLOGITS
15:       $\mathcal{M}[P] \leftarrow \mathbf{x}_{t,N}^P$ 
16:   return  $\mathcal{M}$ 

```

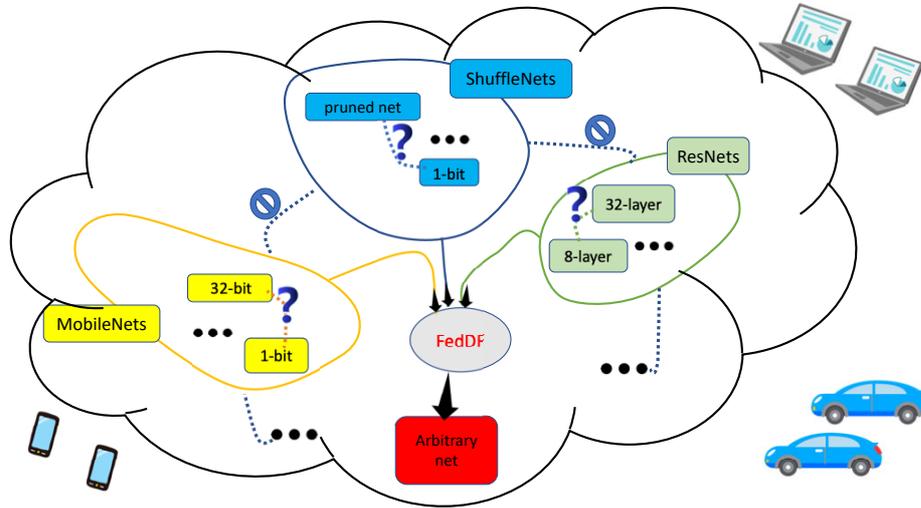


Figure 17.1 – **The schematic diagram for heterogeneous model fusion.** We use dotted lines to indicate model parameter averaging FL methods such as FEDAVG. We could notice the architectural/precision discrepancy invalidates these methods in heterogeneous FL systems. However, FedDF could aggregate knowledge from all available models without hindrance.

17.3 Additional Experimental Setup and Evaluations

17.3.1 Detailed Description for Toy Example (Figure 8.1)

Figure 17.2 provides a detailed illustration of the limitation in FEDAVG.

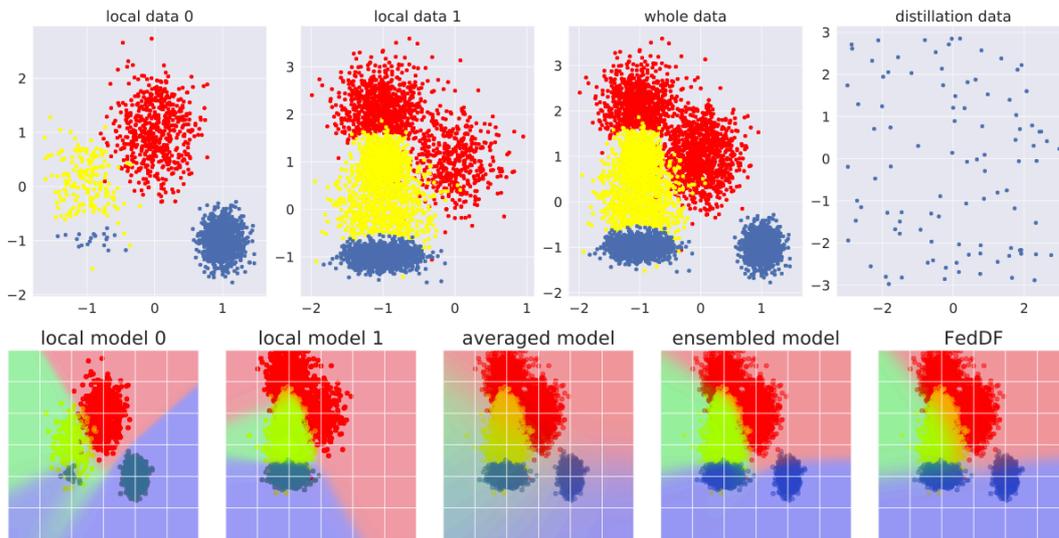


Figure 17.2 – **The limitation of FEDAVG.** We consider a toy example of a 3-class classification task with a 3-layer MLP, and display the decision boundaries (probabilities over RGB channels) on the input space. We illustrate the used datasets in the **top** row; the distillation dataset consists of 60 data points, with each uniformly sampled from the range of $(-3, 3)$. In the **bottom** row, the left two figures consider the individually trained local models. The right three figures evaluate aggregated models and the global data distribution; the averaged model (FEDAVG) results in much blurred decision boundaries.

17.3.2 Detailed Experiment Setup

The detailed hyperparameter tuning procedure. The tuning procedure of hyperparameters ensures that the best hyperparameter lies in the middle of our search grids; otherwise, we extend our search grid. The initial search grid of learning rate is $\{1.5, 1, 0.5, 0.1, 0.05, 0.01\}$. The initial search grid of proximal factor in FEDPROX is $\{0.001, 0.01, 0.1, 1\}$. The initial search grid of momentum factor β in FEDAVGM is $\{0.1, 0.2, 0.3, 0.4\}$; the update scheme of FEDAVGM follows $\Delta \mathbf{v} := \beta \mathbf{v} + \Delta \mathbf{x}; \mathbf{x} := \mathbf{x} - \Delta \mathbf{v}$, where $\Delta \mathbf{x}$ is the model difference between the updated local model and the sent global model, for previous communication round.

Unless mentioned (i.e. Table 8.1), otherwise the learning rate is set to 0.1 for ResNet like architectures (e.g. ResNet-8, ResNet-20, ResNet-32, ShuffleNetV2), 0.05 for VGG and $1e-5$ for DistilBERT. When comparing with other methods, e.g. FEDPROX, FEDAVGM, we always tune their corresponding hyperparameters (e.g. proximal factor in FEDPROX and momentum factor in FEDAVGM).

17.3. Additional Experimental Setup and Evaluations

Experiment details of FEDMA. We detail our attempts of reproducing FEDMA experiments on VGG-9 with CIFAR-10 in this section. We clone their codebase from GitHub and add functionality to sample clients after synchronizing the whole model.

Different from other methods evaluated in the paper, FEDMA uses a layer-wise local training scheme. For each round of the local training, the involved clients only update the model parameters from one specific layer onwards, while the already matched layers are frozen. The fusion (matching) is only performed on the chosen layer. Such a layer is gradually chosen from the bottom layer to the top layer, following a bottom-up fashion (Wang et al., 2020a). One complete model update cycle of FEDMA requires more frequent (but slightly cheaper) communication, which is equivalent to the number of layers in the neural network.

In our experiments of FEDMA, the number of local training epochs is 5 epochs per layer (45 epochs per model update), which is slightly larger than 40 epochs used by other methods. We ensure a similar¹ number of model updates in terms of the whole model. We consider global-wise learning rate, different from the layer-wise one in Wang et al. (2020a). We also turn off the momentum and weight decay during the local training for a consistent evaluation. The implementation of VGG-9 follows <https://github.com/kuangliu/pytorch-cifar/>.

The detailed experimental setup for FedDF (low-bit quantized models). FedDF increases the feasibility of robust model fusion in FL for binarized ResNet-8. As stated in Table 8.4 (Section 8.5.3), we employ the “Straight-through estimator” (Bengio et al., 2013; Hinton, 2012; Hubara et al., 2016, 2017) or the “error-feedback” (Lin et al., 2020c) to simulate the on-device local training of the binarized ResNet-8. For each communication round, the server of the FL system will receive locally trained and binarized ResNet-8 from activated clients. The server will then distill the knowledge of these low-precision models to a full-precision one² and broadcast to newly activated clients for the next communication round. For the sake of simplicity, the case study demonstrated in the paper only considers reducing the communication cost (from clients to the server), and the local computational cost; a thorough investigation on how to perform a communication-efficient and memory-efficient FL is left as future work.

The synthetic formulation of non-i.i.d. client data. Assume every client training example is drawn independently with class labels following a categorical distribution over M classes parameterized by a vector \mathbf{q} ($q_i \geq 0, i \in [1, M]$ and $\|\mathbf{q}\|_1 = 1$). Following the partition scheme introduced and used in Yurochkin et al. (2019); Hsu et al. (2019)³, to synthesize client non-i.i.d.

¹ The other methods use 40 local training epochs per whole model update. Given the fact of layer-wise training scheme in FEDMA, as well as the used 9-layer VGG (same as the one used in Wang et al. (2020a) and we are unable to adapt their code to other architectures due to their hard-coded architecture manipulations), we decide to slightly increase the number of local epochs per layer for FEDMA.

² The training of the binarized network requires to maintain a full-precision model (Hubara et al., 2016, 2017; Lin et al., 2020c) for model update (quantized/pruned model is used during the backward pass).

³ We heavily borrowed the partition description of Hsu et al. (2019) for the completeness of the paper.

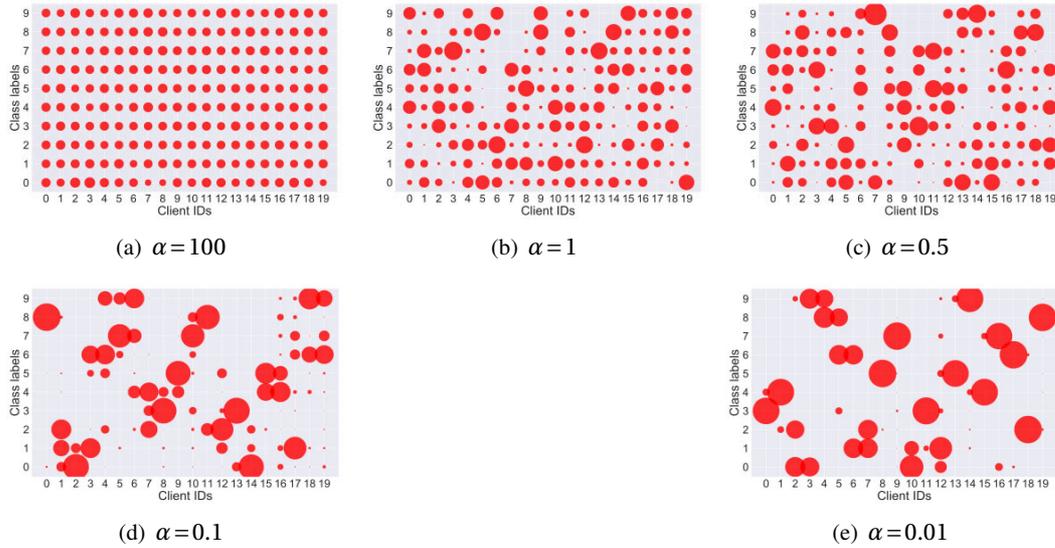


Figure 17.3 – Classes allocated to each client at various Dirichlet distribution alpha values, for CIFAR-10 with 20 clients. The size of each dot reflects the magnitude of the samples number.

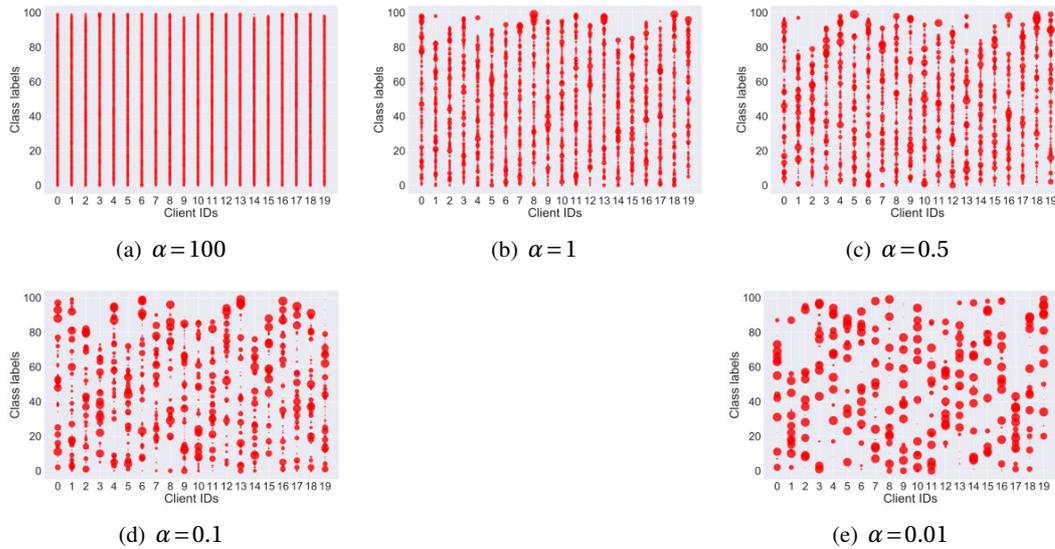


Figure 17.4 – Classes allocated to each client at various Dirichlet distribution alpha values, for CIFAR-100 with 20 clients. The size of each dot reflects the magnitude of the samples number.

local data distributions, we draw $\alpha \sim \text{Dir}(\alpha \mathbf{p})$ from a Dirichlet distribution, where \mathbf{p} characterizes a prior class distribution over M classes, and $\alpha > 0$ is a concentration parameter controlling the identicalness among clients. With $\alpha \rightarrow \infty$, all clients have identical distributions to the prior; with $\alpha \rightarrow 0$, each client holds examples from only one random class.

To better understand the local data distribution for the datasets we considered in the experiments, we visualize the partition results of CIFAR-10 and CIFAR-100 on $\alpha = \{0.01, 0.1, 0.5, 1, 100\}$ for

17.3. Additional Experimental Setup and Evaluations

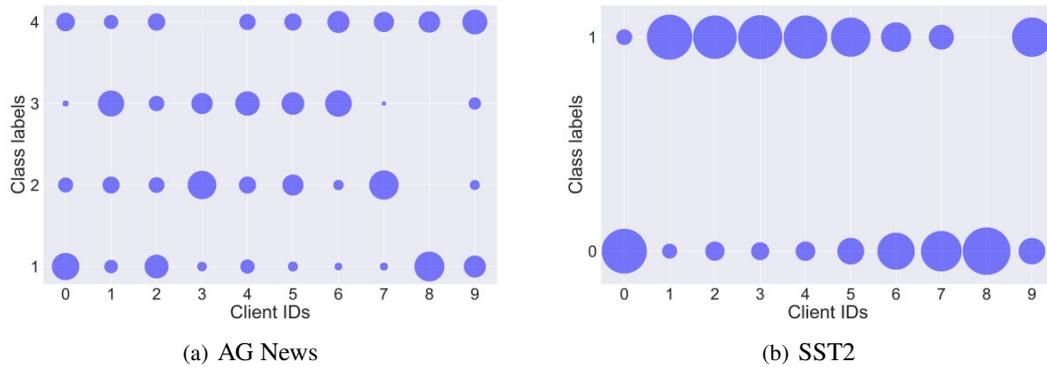


Figure 17.5 – Classes allocated to each client at Dirichlet distribution $\alpha = 1$, for AG News and SST2 datasets with 10 clients. The size of each dot reflects the magnitude of the samples number.

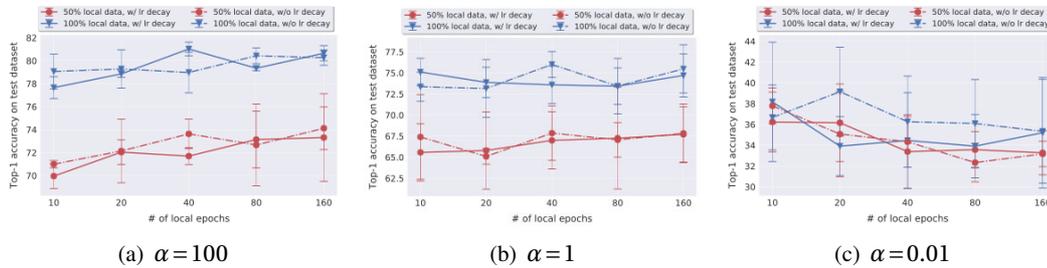


Figure 17.6 – **The ablation study of FEDAVG for various # of local epochs and learning rate schedules**, for standard federated learning on CIFAR-10 with ResNet-8. For each communication round (100 in total), 40% of the total 20 clients are randomly selected. We use α to synthetically control the non-iid degree of the local data, as in [Yurochkin et al. \(2019\)](#); [Hsu et al. \(2019\)](#). The smaller α , the larger discrepancy between local data distributions ($\alpha = 100$ mimics identical local data distributions). We report the top-1 accuracy (on three seeds) on the test dataset.

20 clients, in Figure 17.3 and Figure 17.4, respectively.

In Figure 17.5 we visualize the partitioned local data on 10 clients with $\alpha = 1$, for AG News and SST-2.

17.3.3 Some Empirical Understanding of FEDAVG

Figure 17.6 reviews the general behaviors of FEDAVG under various non-iid degrees of local data, various local data sizes, various numbers of local epochs per communication round, as well as the learning rate schedule during the local training. Since we cannot observe the benefits of decaying the learning rate during the local training phase, we turn off the learning rate decay for the experiments in the main text.

In Figure 17.7, we visualize the learning curves of training ResNet-8 on CIFAR-10 with various normalization techniques. The numerical results correspond to Table 8.2 in the main text.

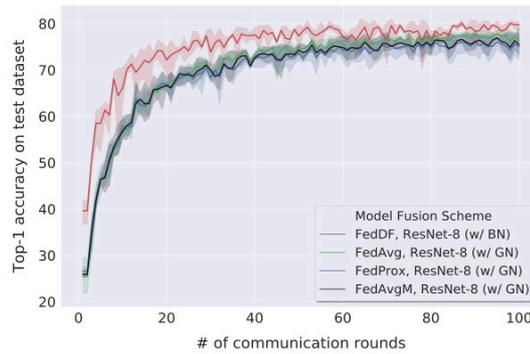


Figure 17.7 – The impact of various normalization techniques, i.e. Batch Normalization (BN), Group Normalization (GN), for federated learning on CIFAR-10 with ResNet-8 with $\alpha = 1$. For each communication round (100 in total), 40% of the total 20 clients are randomly selected for 40 local epochs.

Table 17.1 – **Understanding the importance of model initialization in FedDF**, on CIFAR-10 with ResNet-8. For each communication round (100 in total), 40% of the total 20 clients are randomly selected. The scheme “from average” indicates initializing the model for ensemble distillation from the uniformly averaged model of this communication round; while the scheme “from previous” instead uses the fused model from the previous communication round as the starting point. We report the top-1 accuracy (on three seeds) on the test dataset.

local training epochs	$\alpha = 1$		$\alpha = 0.1$	
	from average	from previous	from average	from previous
40	80.43 \pm 0.37	74.13 \pm 0.91	71.84 \pm 0.86	62.94 \pm 1.12
80	81.17 \pm 0.53	76.37 \pm 0.60	74.73 \pm 0.65	67.88 \pm 0.90

17.3.4 The Advantages of FedDF

Ablation Study

The Importance of the Model Initialization in FedDF. We empirically study the importance of the initialization (before performing ensemble distillation) in FedDF. Table 17.1 demonstrates the performance difference of FedDF for two different model initialization schemes: 1) “from average”, where the uniformly averaged model from this communication round is used as the initial model (i.e. the default design choice of FedDF as illustrated in Algorithm 5 and Algorithm 18); and 2) “from previous”, where we initialize the model for ensemble distillation by utilizing the fusion result of FedDF from the previous communication round. The noticeable performance differences illustrated in Table 17.1 identify the importance of using the uniformly averaged model⁴ (from the current communication round) as a starting model for better ensemble distillation.

⁴ The related preprints (Li and Wang, 2019; Chang et al., 2019) are closer to the second initialization scheme. They do not or cannot introduce the uniformly averaged model (on the server) into the federated learning pipeline; instead, they only utilize the averaged logits (on the same data) for each client’s local training.

17.3. Additional Experimental Setup and Evaluations

Table 17.2 – **Understanding the impact of local training quality**, on CIFAR-10 with ResNet-8. For each communication round (100 in total), 40% of the total 20 clients are randomly selected for 40 local epochs. We report the top-1 accuracy (on three seeds) on the test dataset.

local client training scheme	$\alpha = 1$		$\alpha = 0.1$	
	FedDF	FEDAVG	FedDF	FEDAVG
SGD	80.27	72.73	71.52	62.44
Adam	83.32	78.13	72.58	62.53

Table 17.3 – **On the impact of using various optimizers for ensemble distillation** in FedDF, on CIFAR-10 with ResNet-8. For each communication round (100 in total), 40% of the total 20 clients are randomly selected for 40 local epochs. We report the top-1 accuracy (on three seeds) on the test dataset. “SGD” uses the same learning rate scheduler as our “Adam” choice (i.e. cosine annealing), and with fine-tuned initial learning rate. “SWAG” refers to the mechanism to form an approximated posterior distribution (Maddox et al., 2019) where more models can be sampled from, and Chen and Chao (2021) further propose to use SWAG on the received client models for better ensemble distillation; our default design resorts to directly averaged logits from received local clients with Adam optimizer. To ensure a fair comparison, we use the same distillation dataset as in FedDF (i.e. CIFAR-100) for “SWAG” (Chen and Chao, 2021). We fine-tune other hyperparameters in “SWAG”: we use all received client models and 10 sampled models from Gaussian distribution (as suggested in Chen and Chao (2021)) for the ensemble distillation.

optimizer used on the server	$\alpha = 1$		$\alpha = 0.1$	
	FedDF	FEDAVG	FedDF	FEDAVG
SGD	76.68	72.73	57.33	62.44
Adam (our default design)	80.27	72.73	71.52	62.44
SWAG (Maddox et al., 2019; Chen and Chao, 2021)	80.84	72.73	72.40	62.44

The performance gain in FedDF. To distinguish the benefits of FedDF from the small learning rate (during the local training) or Adam optimizer (used for ensemble distillation in FedDF), we report the results of using Adam (lr=1e-3) for both local training and model fusion (over three seeds), on CIFAR-10 with ResNet-8, in Table 17.2. Improving the local training through Adam might help Federated Learning but the benefit vanishes with higher data heterogeneity (e.g. $\alpha = 0.1$). Performance gain from FedDF is robust to data heterogeneity and also orthogonal to effects of learning rates and Adam.

Table 17.3 examines the effect of various optimization schemes on the quality of ensemble distillation. We can witness that with two extra hyperparameters (sampling scale for SWAG and the number of models to be sampled), SWAG can slightly improve the distillation performance. In contrast, we use Adam with default hyperparameters as our design choice in FedDF: it demonstrates similar performance (compared to the choice of SWAG) with trivial tuning overhead.

The compatibility of FedDF with other methods. Table 17.4 justifies the compatibility of FedDF. Our empirical results demonstrate a significant performance gain of FedDF over the FEDAVG, even in the case of using local proximal regularizer to avoid catastrophically over-fitting the heterogeneous local data, which reduces the diversity of local models that FedDF benefits from.

Chapter 17. Appendix for FedDF

Table 17.4 – **The compatibility of FedDF with other training schemes**, on CIFAR-10 with ResNet-8. For each communication round (100 in total), 40% of the total 20 clients are randomly selected for 40 local epochs. We consider the fine-tuned proximal penalty from FedDF. We report the top-1 accuracy (on three seeds) on the test dataset.

local client training scheme	$\alpha = 1$		$\alpha = 0.1$	
	FedDF	FEDAVG	FedDF	FEDAVG
SGD	80.27	72.73	71.52	62.44
SGD + proximal penalty	80.56	76.11	71.64	62.53

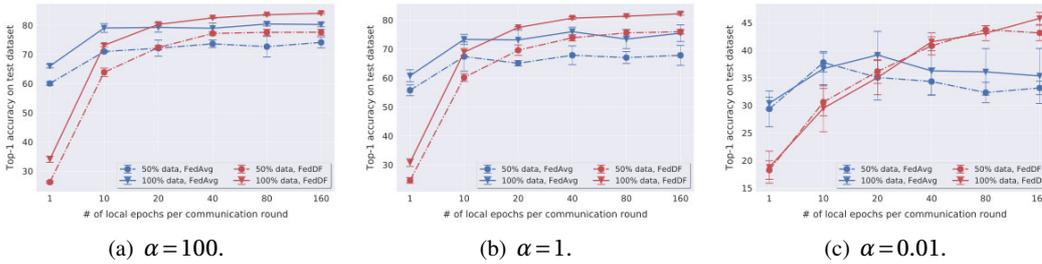
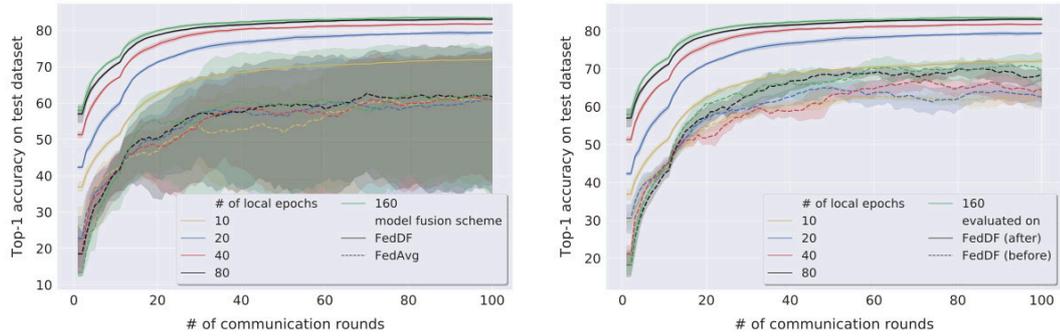


Figure 17.8 – The **test performance** of **FedDF** and **FEDAVG** on **CIFAR-10** with **ResNet-8**, for various local data non-iid degrees α , data fractions, and # of local epochs per communication round. For each communication round (100 in total), 40% of the total 20 clients are randomly selected. We report the top-1 accuracy (on three seeds) on the test dataset. This Figure complements Figure 8.2.

Comparison with FEDAVG

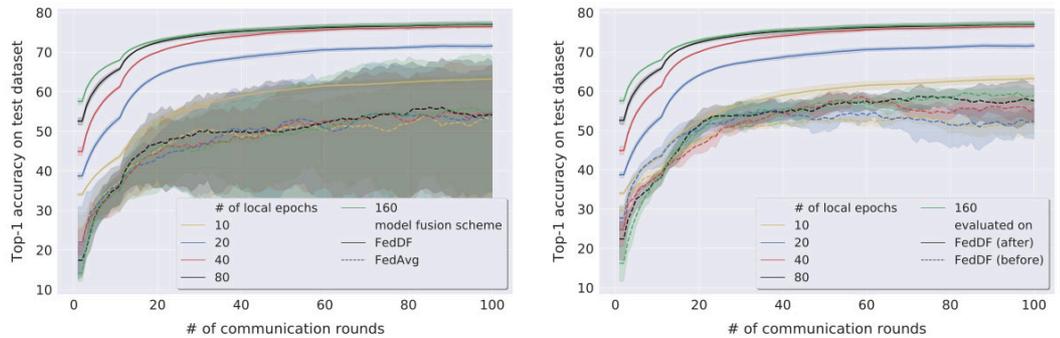
Figure 17.8 complements Figure 8.2 in the main text and presents a thorough comparison between FEDAVG and FedDF, for a variety of local training epochs, data fractions, non-i.i.d. degrees. The detailed learning curves of the cases in this figure are visualized in Figure 17.9, Figure 17.10, and Figure 17.11.

17.3. Additional Experimental Setup and Evaluations



(a) The learning behaviors of FedDF and FEDAVG. We evaluate various # of local epochs on 100% local data.

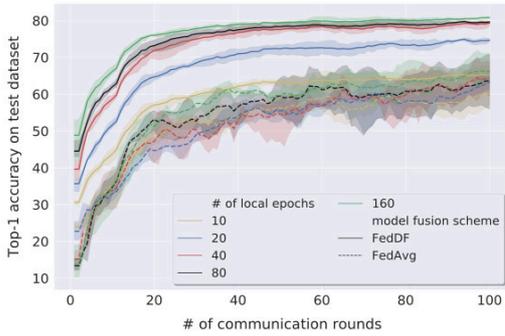
(b) The fused model performance before (i.e. line 6 in Algorithm 5) and after FedDF (i.e. line 10 in Algorithm 5). We evaluate various # of local epochs on 100% local data.



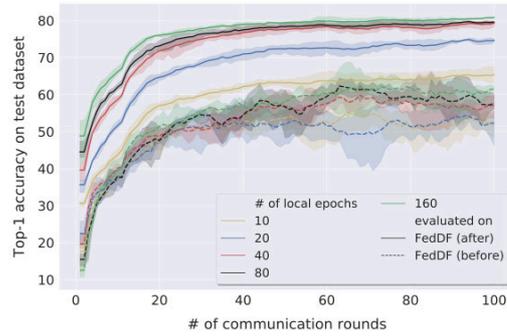
(c) The learning behaviors of FedDF and FEDAVG. We evaluate various # of local epochs on 50% local data.

(d) The fused model performance before (i.e. line 6 in Algorithm 5) and after FedDF (i.e. line 10 in Algorithm 5). We evaluate various # of local epochs on 50% local data.

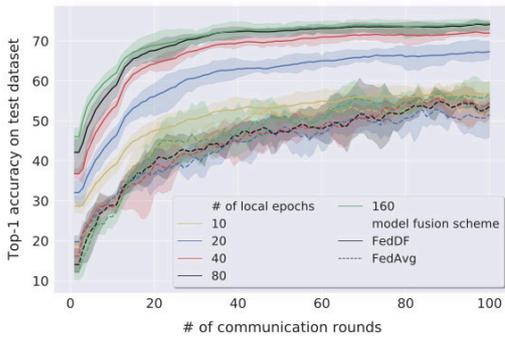
Figure 17.9 – Understanding the learning behaviors of FedDF on CIFAR-10 with ResNet-8 for $\alpha=100$. For each communication round (100 in total), 40% of the total 20 clients are randomly selected. We report the top-1 accuracy (on three seeds) on the test dataset.



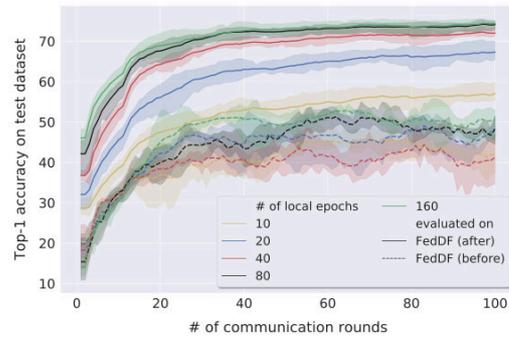
(a) The learning behaviors of FedDF and FEDAVG. We evaluate various # of local epochs on 100% local data.



(b) The fused model performance before (i.e. line 6 in Algorithm 5) and after FedDF (i.e. line 10 in Algorithm 5). We evaluate various # of local epochs on 100% local data.



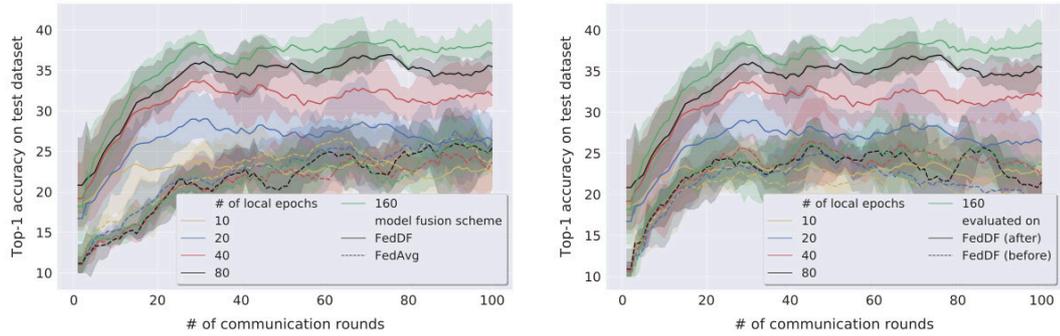
(c) The learning behaviors of FedDF and FEDAVG. We evaluate various # of local epochs on 50% local data.



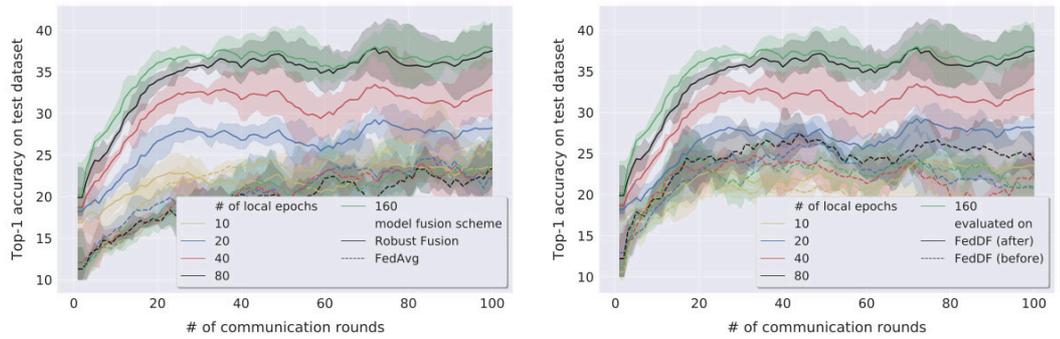
(d) The fused model performance before (i.e. line 6 in Algorithm 5) and after FedDF (i.e. line 10 in Algorithm 5). We evaluate various # of local epochs on 50% local data.

Figure 17.10 – Understanding the learning behaviors of FedDF on CIFAR-10 with ResNet-8 for $\alpha = 1$. For each communication round (100 in total), 40% of the total 20 clients are randomly selected. We report the top-1 accuracy (on three seeds) on the test dataset.

17.3. Additional Experimental Setup and Evaluations



(a) The learning behaviors of FedDF and FEDAVG. We evaluate various # of local epochs on 100% local data. (b) The fused model performance before (i.e. line 6 in Algorithm 5) and after FedDF (i.e. line 10 in Algorithm 5). We evaluate various # of local epochs on 100% local data.



(c) The learning behaviors of FedDF and FEDAVG. We evaluate various # of local epochs on 50% local data. (d) The fused model performance before (i.e. line 6 in Algorithm 5) and after FedDF (i.e. line 10 in Algorithm 5). We evaluate various # of local epochs on 50% local data.

Figure 17.11 – Understanding the learning behaviors of FedDF on CIFAR-10 with ResNet-8 for $\alpha=0.01$. For each communication round (100 in total), 40% of the total 20 clients are randomly selected. We report the top-1 accuracy (on three seeds) on the test dataset.

17.4 Details on Generalization Bounds

The derivation of the generalization bound starts from the following notations. In FL, each client has access to its own data distribution \mathcal{D}_i over domain $\Xi := \mathcal{X} \times \mathcal{Y}$, where $\mathcal{X} \in \mathbb{R}^d$ is the input space and \mathcal{Y} is the output space. The global distribution on the server is denoted as \mathcal{D} . For the empirical distribution by the given dataset, we assume that each local model has access to an equal amount (m) of local data. Thus, each local empirical distribution has equal contribution to the global empirical distribution: $\hat{\mathcal{D}} = \frac{1}{K} \sum_{k=1}^K \hat{\mathcal{D}}_k$, where $\hat{\mathcal{D}}_k$ denotes the empirical distribution from client k .

For our analysis we assume a binary classification task, with hypothesis h as a function $h: \mathcal{X} \rightarrow \{0, 1\}$. The loss function of the task is defined as $\ell(h(\mathbf{x}), y) = |\hat{y} - y|$, where $\hat{y} := h(\mathbf{x})$. Note that $\ell(\hat{y}, y)$ is convex with respect to \hat{y} . We denote $\operatorname{argmin}_{h \in \mathcal{H}} L_{\hat{\mathcal{D}}}(h)$ by $h_{\hat{\mathcal{D}}}$.

The theorem below leverages the domain measurement tools developed in multi-domain learning theory (Ben-David et al., 2010) and provides insights for the generalization bound of the ensemble⁵ of local models (trained on local empirical distribution $\hat{\mathcal{D}}_i$).

Theorem 17.4.1. *Let \mathcal{H} be a hypothesis class with $\operatorname{VCdim}(\mathcal{H}) \leq d < \infty$. The difference between $L_{\mathcal{D}}(\frac{1}{K} \sum_k h_{\hat{\mathcal{D}}_k})$ and $L_{\hat{\mathcal{D}}}(h_{\hat{\mathcal{D}}})$, i.e. the distance between the risk of our “ensembled” model in FedDF and the empirical risk of the “virtual ERM” with access to all local data, can be bounded with probability at least $1 - \delta$:*

$$L_{\mathcal{D}}\left(\frac{1}{K} \sum_k h_{\hat{\mathcal{D}}_k}\right) \leq L_{\hat{\mathcal{D}}}(h_{\hat{\mathcal{D}}}) + \frac{4 + \sqrt{\log(\tau_{\mathcal{H}}(2m))}}{\delta / K \sqrt{2m}} + \frac{1}{K} \sum_k \left(\frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_k, \mathcal{D}) + \lambda_k \right),$$

where $\hat{\mathcal{D}} = \frac{1}{K} \sum_k \hat{\mathcal{D}}_k$, $d_{\mathcal{H}\Delta\mathcal{H}}$ measures the domain discrepancy between two distributions (Ben-David et al., 2010), and $\lambda_k = \inf_{h \in \mathcal{H}} (\mathcal{L}_{\mathcal{D}}(h) + \mathcal{L}_{\mathcal{D}_k}(h))$. $\tau_{\mathcal{H}}$ is the growth function of \mathcal{H} —it satisfies for all m , $\tau_{\mathcal{H}}(m) \leq \sum_{i=0}^d \binom{m}{i}$, and if $m > d + 1$ then $\tau_{\mathcal{H}}(m) \leq (\frac{em}{d})^d$.

Remark 17.4.2. *Theorem 17.4.1 shows that, the upper bound on the risk of the ensemble of K local models on \mathcal{D} mainly consists of 1) the empirical risk of a model trained on the global empirical distribution $\hat{\mathcal{D}} = \frac{1}{K} \sum_k \hat{\mathcal{D}}_k$, and 2) terms dependent on the distribution discrepancy between \mathcal{D}_k and \mathcal{D} .*

The ensemble of the local models sets the performance upper bound for the later distilled model on the test domain as shown in Figure 8.4. Theorem 8.6.1 shows that compared to a model trained on aggregated local data (ideal case), the performance of an ensemble model on the test distribution is affected by the domain discrepancy between local distributions \mathcal{D}_k 's and the test distribution \mathcal{D} . The shift between the distillation and the test distribution determines the knowledge transfer quality between these two distributions and hence the test performance of the fused model. Through the lens of the domain adaptation theory (Ben-David et al., 2010), we can better spot the potential influence/limiting factors on our ensemble distillation procedure.

⁵ The uniformly weighted hypothesis average in multi-source adaptation is equivalent to the ensemble of a list of models, by considering the output of each hypothesis/model.

Remark 17.4.3. *In the area of multiple-source adaptation, Mansour et al. (2009); Hoffman et al. (2018) point out that the standard convex combinations of the source hypotheses may perform poorly on the test distribution. They propose combinations with weights derived from source distributions. However, FL scenarios require the server only access local models without any further local information. Thus we choose to uniformly average over local hypotheses as our global hypothesis. A privacy-preserved local distribution estimation is left for future work.*

17.4.1 Proof for Generalization Bounds

Theorem 17.4.4 (Uniform Convergence (Shalev-Shwartz and Ben-David, 2014)). *Let \mathcal{H} be a class and let $\tau_{\mathcal{H}}$ be its growth function. Then, for every \mathcal{D} and every $\delta \in (0, 1)$, with probability of at least $1 - \delta$ over the choice of $S \sim \mathcal{D}^m$, we have*

$$|L_{\mathcal{D}}(h) - L_S(h)| \leq \frac{4 + \sqrt{\log(\tau_{\mathcal{H}}(2m))}}{\delta\sqrt{2m}}.$$

Lemma 17.4.5 (Sauer’s Lemma (Shalev-Shwartz and Ben-David, 2014)). *Let \mathcal{H} be a hypothesis class with $\text{VCdim}(\mathcal{H}) \leq d < \infty$. Then for all m , $\tau_{\mathcal{H}}(m) \leq \sum_{i=0}^d \binom{m}{i}$. In particular, if $m > d + 1$ then $\tau_{\mathcal{H}}(m) \leq (\frac{em}{d})^d$.*

Theorem 17.4.6 (Domain adaptation (Ben-David et al., 2010)). *Considering the distributions \mathcal{D}_S and \mathcal{D}_T , for every $h \in \mathcal{H}$ and any $\delta \in (0, 1)$, with probability at least $1 - \delta$ (over the choice of the samples), there exists:*

$$L_{\mathcal{D}_T}(h) \leq L_{\mathcal{D}_S}(h) + \frac{1}{2}d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T) + \lambda, \quad (17.1)$$

where $\lambda = L_{\mathcal{D}_S}(h^*) + L_{\mathcal{D}_T}(h^*)$. $h^* := \arg\min_{h \in \mathcal{H}} L_{\mathcal{D}_S}(h) + L_{\mathcal{D}_T}(h)$ corresponds to ideal joint hypothesis that minimizes the combined error.

Proof of Theorem 17.4.1. We start from the risk of our “ensembled” model $L_{\mathcal{D}}(\frac{1}{K} \sum_k h_{\hat{\phi}_k})$ and derive a series of upper bounds.

Considering the distance between $L_{\mathcal{D}}(\frac{1}{K} \sum_k h_{\hat{\phi}_k})$ and $L_{\hat{\phi}}(h_{\hat{\phi}})$. By convexity of ℓ and Jensen inequality, we have

$$L_{\mathcal{D}}(\frac{1}{K} \sum_k h_{\hat{\phi}_k}) \leq \frac{1}{K} \sum_k L_{\mathcal{D}}(h_{\hat{\phi}_k}). \quad (17.2)$$

Using the domain adaptation theory in Theorem 17.4.6, we transfer from domain \mathcal{D} to \mathcal{D}_k ,

$$L_{\mathcal{D}}(h_{\hat{\phi}_k}) \leq L_{\mathcal{D}_k}(h_{\hat{\phi}_k}) + \frac{1}{2}d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_k, \mathcal{D}) + \lambda_k, \quad (17.3)$$

where $\lambda_k := L_{\mathcal{D}}(h^*) + L_{\mathcal{D}_k}(h^*)$ and $h^* := \arg\min_{h \in \mathcal{H}} L_{\mathcal{D}}(h) + L_{\mathcal{D}_k}(h)$.

We can bound the risk with its empirical counterpart. Applying Theorem 17.4.4 yields

$$L_{\mathcal{D}_k}(h_{\hat{\mathcal{D}}_k}) \leq L_{\hat{\mathcal{D}}_k}(h_{\hat{\mathcal{D}}_k}) + \frac{4 + \sqrt{\log(\tau_{\mathcal{H}}(2m))}}{\delta / K \sqrt{2m}}, \quad (17.4)$$

where $\tau_{\mathcal{H}}$ denotes the growth function of \mathcal{H} .⁶

Thus for K sources, we have

$$\begin{aligned} & \Pr_{S_1 \sim \mathcal{D}_1^m, \dots, S_K \sim \mathcal{D}_K^m} \left[\bigcap_{k=1}^K \left\{ L_{\mathcal{D}_k}(h_{\hat{\mathcal{D}}_k}) \leq L_{\hat{\mathcal{D}}_k}(h_{\hat{\mathcal{D}}_k}) + \frac{4 + \sqrt{\log(\tau_{\mathcal{H}}(2m))}}{\delta / K \sqrt{2m}} \right\} \right] \\ &= 1 - \Pr_{S_1 \sim \mathcal{D}_1^m, \dots, S_K \sim \mathcal{D}_K^m} \left[\bigcup_{k=1}^K \left\{ L_{\mathcal{D}_k}(h_{\hat{\mathcal{D}}_k}) \geq L_{\hat{\mathcal{D}}_k}(h_{\hat{\mathcal{D}}_k}) + \frac{4 + \sqrt{\log(\tau_{\mathcal{H}}(2m))}}{\delta / K \sqrt{2m}} \right\} \right] \\ &\geq 1 - \sum_{k=1}^K \Pr_{S_1 \sim \mathcal{D}_1^m, \dots, S_K \sim \mathcal{D}_K^m} \left[\left\{ L_{\mathcal{D}_k}(h_{\hat{\mathcal{D}}_k}) \geq L_{\hat{\mathcal{D}}_k}(h_{\hat{\mathcal{D}}_k}) + \frac{4 + \sqrt{\log(\tau_{\mathcal{H}}(2m))}}{\delta / K \sqrt{2m}} \right\} \right] \\ &\geq 1 - \delta. \end{aligned} \quad (17.5)$$

Based on the definition of ERM, we have $L_{\hat{\mathcal{D}}_k}(h_{\hat{\mathcal{D}}_k}) \leq L_{\hat{\mathcal{D}}_k}(h_{\hat{\mathcal{D}}})$, where $h_{\hat{\mathcal{D}}}$ corresponds to the classifier trained with data from all workers. By using the definition of $\hat{\mathcal{D}}$ ($\hat{\mathcal{D}} = \frac{1}{K} \sum_k \hat{\mathcal{D}}_k$) and the linearity of expectation, we have

$$\frac{1}{K} \sum_k L_{\hat{\mathcal{D}}_k}(h_{\hat{\mathcal{D}}_k}) \leq \frac{1}{K} \sum_k L_{\hat{\mathcal{D}}_k}(h_{\hat{\mathcal{D}}}) = L_{\hat{\mathcal{D}}}(h_{\hat{\mathcal{D}}}). \quad (17.6)$$

Putting these equations together, we have with probability of at least $1 - \delta$ over $S_1 \sim \mathcal{D}_1^m, \dots, S_K \sim \mathcal{D}_K^m$ that

$$\begin{aligned} L_{\mathcal{D}}\left(\frac{1}{K} \sum_k h_{\hat{\mathcal{D}}_k}\right) &\leq \frac{1}{K} \sum_k L_{\mathcal{D}}(h_{\hat{\mathcal{D}}_k}) \\ &\leq \frac{1}{K} \sum_k \left(L_{\mathcal{D}_k}(h_{\hat{\mathcal{D}}_k}) + \frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_k, \mathcal{D}) + \lambda_k \right) \\ &\leq \frac{1}{K} \sum_k \left(L_{\hat{\mathcal{D}}_k}(h_{\hat{\mathcal{D}}_k}) + \frac{4 + \sqrt{\log(\tau_{\mathcal{H}}(2m))}}{\delta / K \sqrt{2m}} + \frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_k, \mathcal{D}) + \lambda_k \right) \\ &\leq \frac{1}{K} \sum_k L_{\hat{\mathcal{D}}_k}(h_{\hat{\mathcal{D}}_k}) + \frac{4 + \sqrt{\log(\tau_{\mathcal{H}}(2m))}}{\delta / K \sqrt{2m}} + \frac{1}{K} \sum_k \left(\frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_k, \mathcal{D}) + \lambda_k \right) \\ &\leq L_{\hat{\mathcal{D}}}(h_{\hat{\mathcal{D}}}) + \frac{4 + \sqrt{\log(\tau_{\mathcal{H}}(2m))}}{\delta / K \sqrt{2m}} + \frac{1}{K} \sum_k \left(\frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_k, \mathcal{D}) + \lambda_k \right), \end{aligned}$$

where $\lambda_k = \inf_{h \in \mathcal{H}} (\mathcal{L}_{\mathcal{D}}(h) + \mathcal{L}_{\mathcal{D}_k}(h))$. □

⁶ Note that with Lemma 17.4.5, we could bound $\tau_{\mathcal{H}}$ by a polynomial function of m and d .

18 Appendix for Quasi-Global Momentum

18.1 Detailed Experimental Setup

18.1.1 Visualization for Communication Topologies

Figure 18.1 visualizes the Social Network topology we evaluated in the main paper.

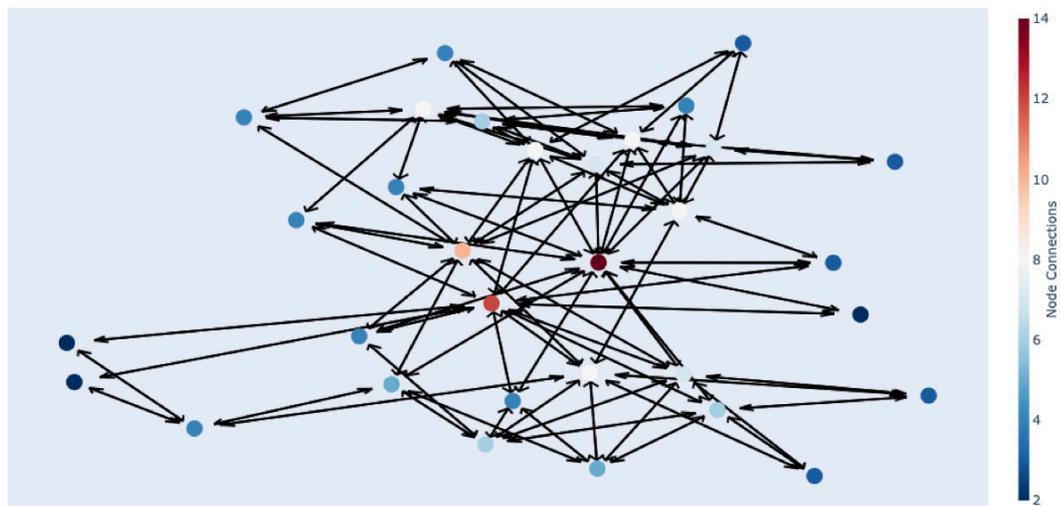


Figure 18.1 – The visualization of the examined social topology (generated from “networkx.generators.social.davis_southern_women_graph”).

18.1.2 Visualization of Non-IID Local Data

The synthetic formulation of non-i.i.d. client data. We re-iterate the partition scheme introduced and stated in (Yurochkin et al., 2019; Hsu et al., 2019) for completeness reasons.

Assume every client training example is drawn independently with class labels following a categorical distribution over M classes parameterized by a vector \mathbf{q} ($q_i \geq 0, i \in [1, M]$ and $\|\mathbf{q}\|_1 = 1$). To synthesize client non-i.i.d. local data distributions, we draw $\alpha \sim \text{Dir}(\alpha\mathbf{p})$ from a Dirichlet distribution, where \mathbf{p} characterizes a prior class distribution over M classes, and $\alpha > 0$ is a concentration parameter controlling the identicalness among clients. With $\alpha \rightarrow \infty$, all clients have identical distributions to the prior; with $\alpha \rightarrow 0$, each client holds examples from only one random class.

To better understand the local data distribution for the datasets we considered in the experiments, in Figure 18.2 we visualize the partition results of CIFAR-10 and ImageNet(-32) for various degrees of non-i.i.d.-ness and network scales; in Figure 18.3, we visualize the partitioned local data on 16 clients with $\alpha = \{10, 1, 0.1\}$ for AG News and SST-2.

18.2 Detailed Algorithm Description and Connections

18.2.1 Detailed Algorithm Description

The variant of Adam with the idea of quasi-global momentum is detailed in Algorithm 19.

Algorithm 19 QG-DAdam. $\hat{\mathbf{m}}_i^{(0)}, \hat{\mathbf{v}}_i^{(0)}$ are initialized as $\mathbf{0}$ for all workers.

```

1: procedure
2:   for  $t \in \{1, \dots, T\}$  do
3:     sample  $\xi_i^{(t)}$  and compute  $\mathbf{g}_i^{(t)} = \nabla F_i(\mathbf{x}_i^{(t)}, \xi_i^{(t)})$ 
4:      $\mathbf{m}_i^{(t)} = \beta_1 \hat{\mathbf{m}}_i^{(t-1)} + (1 - \beta_1) \hat{\mathbf{g}}_i^{(t)}$ 
5:      $\mathbf{v}_i^{(t)} = \beta_2 \hat{\mathbf{v}}_i^{(t-1)} + (1 - \beta_2) \hat{\mathbf{g}}_i^{(t)} \odot \hat{\mathbf{g}}_i^{(t)}$ 
6:      $\mathbf{x}_i^{(t+\frac{1}{2})} = \mathbf{x}_i^{(t)} - \eta \frac{\mathbf{m}_i^{(t)}}{\sqrt{\mathbf{v}_i^{(t)} + \epsilon}}$ 
7:      $\mathbf{x}_i^{(t+1)} = \sum_{j \in \mathcal{N}_i^{(t)}} w_{ij} \mathbf{x}_j^{(t+\frac{1}{2})}$ 
8:      $\mathbf{d}_i^{(t)} = \mathbf{x}_i^{(t)} - \mathbf{x}_i^{(t+1)}$ 
9:      $\hat{\mathbf{d}}_i^{(t)} = \frac{\mathbf{d}_i^{(t)}}{\|\mathbf{d}_i^{(t)}\|_2}$ 
10:     $\hat{\mathbf{m}}_i^{(t)} = \beta_1 \hat{\mathbf{m}}_i^{(t-1)} + (1 - \beta_1) \hat{\mathbf{d}}_i^{(t)}$ 
11:     $\hat{\mathbf{v}}_i^{(t)} = \beta_2 \hat{\mathbf{v}}_i^{(t-1)} + (1 - \beta_2) \hat{\mathbf{d}}_i^{(t)} \odot \hat{\mathbf{d}}_i^{(t)}$ 
12:  return  $\mathbf{x}_i^{(T)}$ 

```

Algorithm 21 depicts the general procedure of MimeLite in (Karimireddy et al., 2021). For SGDm, the update step \mathcal{U} and the tracking step \mathcal{V} follow

$$\mathcal{U}(\nabla F_i(\mathbf{y}_i, \xi), \mathbf{s}) := (1 - \beta) \nabla F_i(\mathbf{y}_i, \xi) + \beta \mathbf{s}$$

$$\mathcal{V}\left(\frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \nabla f_i(\mathbf{x}), \mathbf{s}\right) := (1 - \beta) \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \nabla f_i(\mathbf{x}) + \beta \mathbf{s}.$$

18.2. Detailed Algorithm Description and Connections

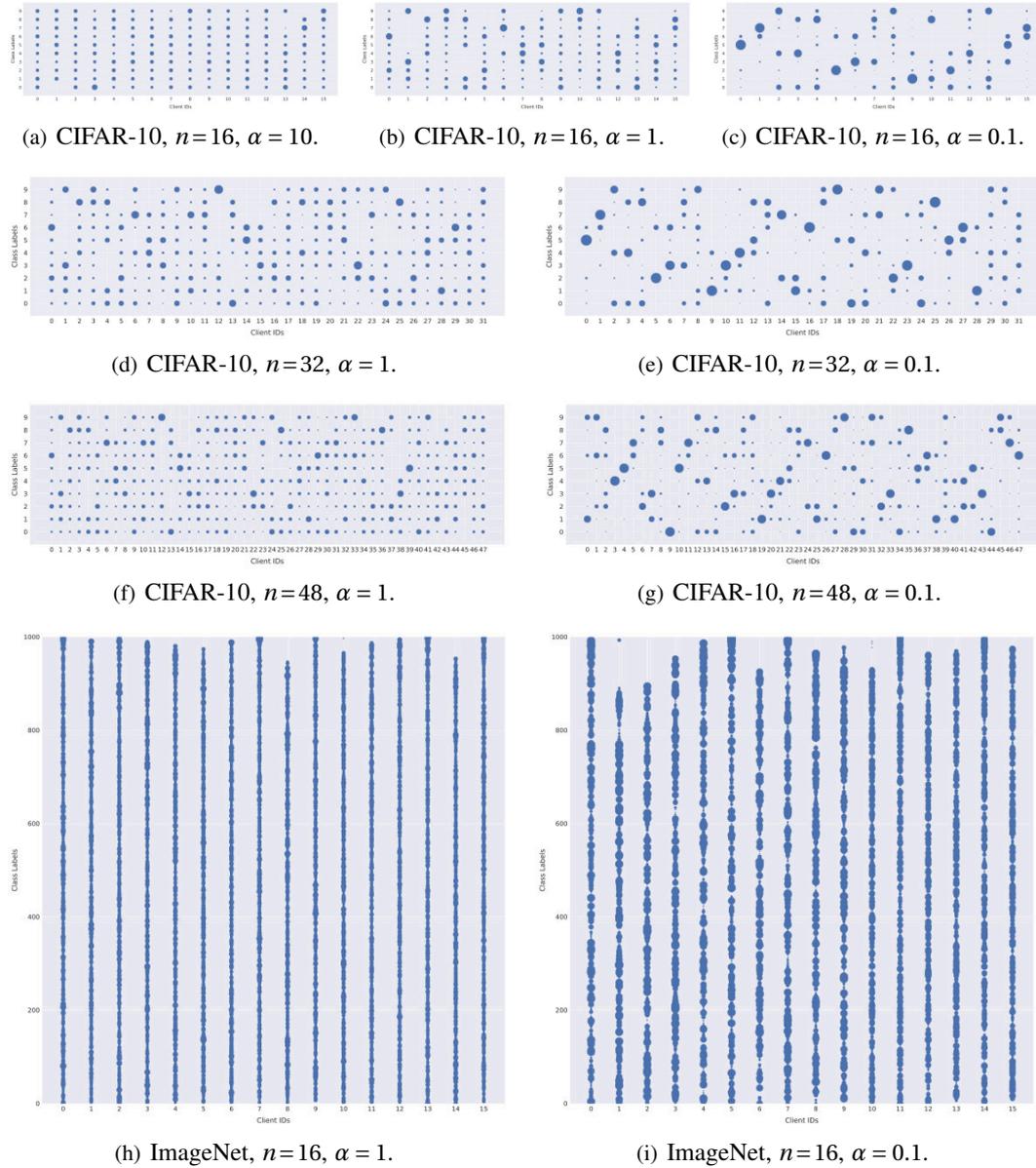


Figure 18.2 – Illustration of # of samples per class allocated to each client (indicated by dot sizes), for various Dirichlet distribution α values on CV datasets.

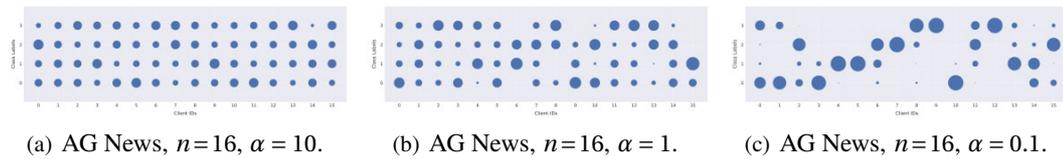


Figure 18.3 – Illustration of # of samples per class allocated to each client (indicated by dot sizes), for various Dirichlet distribution α values on NLP datasets.

Chapter 18. Appendix for Quasi-Global Momentum

Algorithm 20 Multiple-step variant of QG-DSGDm. $\mathbf{m}_i^{(0)} = \hat{\mathbf{m}}_i^{(0)} := \mathbf{0}$. τ is the number of local steps.

```

1: procedure WORKER- $i$ 
2:   for  $t \in \{1, \dots, T\}$  do
3:     sample  $\xi_i^{(t)}$  and compute  $\mathbf{g}_i^{(t)} = \nabla F_i(\mathbf{x}_i^{(t)}, \xi_i^{(t)})$ 
4:      $\mathbf{m}_i^{(t)} = \beta \hat{\mathbf{m}}_i^{(t-1)} + \mathbf{g}_i^{(t)}$ 
5:      $\mathbf{x}_i^{(t+\frac{1}{2})} = \mathbf{x}_i^{(t)} - \eta \mathbf{m}_i^{(t)}$ 
6:      $\mathbf{x}_i^{(t+1)} = \sum_{j \in \mathcal{N}_i^{(t)}} w_{ij} \mathbf{x}_j^{(t+\frac{1}{2})}$ 
7:     if  $\text{mod}(t, \tau) \neq 0$  then
8:        $\hat{\mathbf{m}}_i^{(t)} = \hat{\mathbf{m}}_i^{(t-1)}$ 
9:     else
10:       $\mathbf{d}_i^{(t)} = \frac{\mathbf{x}_i^{(t)} - \mathbf{x}_i^{(t+1)}}{\eta}$ 
11:       $\hat{\mathbf{m}}_i^{(t)} = \mu \hat{\mathbf{m}}_i^{(t-1)} + (1 - \mu) \mathbf{d}_i^{(t)}$ 
12:   return  $\mathbf{x}_i^{(T)}$ 

```

Algorithm 21 MimeLite (Karimireddy et al., 2021).

```

1: procedure
2:   for each round  $t \in [T]$  do
3:     sample subset  $\mathcal{S}$  of clients
4:     communicate  $(\mathbf{x}, \mathbf{s})$  to all clients  $i \in \mathcal{S}$ 
5:     for client  $i \in \mathcal{S}$  in parallel do
6:       initialize local model  $\mathbf{y}_i \leftarrow \mathbf{x}$ 
7:       for client  $i \in \mathcal{S}$  in parallel do
8:         for  $k \in [\tau]$  do
9:           sample mini-batch  $\xi$  from local data
10:           $\mathbf{y}_i \leftarrow \mathbf{y}_i - \eta \mathcal{U}(\nabla F_i(\mathbf{y}_i, \xi), \mathbf{s})$ 
11:          compute full local-batch gradient  $\nabla f_i(\mathbf{x})$ 
12:          communicate  $(\mathbf{y}_i, \nabla f_i(\mathbf{x}))$ 
13:         $\mathbf{s} \leftarrow \mathcal{V}\left(\frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \nabla f_i(\mathbf{x}, \mathbf{s})\right)$  ▷ update optimization statistics
14:         $\mathbf{x} \leftarrow \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \mathbf{y}_i$  ▷ update server parameters
15:   return  $\mathbf{x}_T$ 

```

Algorithm 22 shows the pseudocode of SlowMo (Wang et al., 2020d). For our evaluation in Table 9.6, we follow the hyperparameter suggestion mentioned in (Wang et al., 2020d): for CIFAR-10 dataset, we set $\alpha = 1$, $\tau = 12$, $\beta = 0.7$.

18.2.2 Difference between DMSGD and QG-DSGDm

We first re-iterate DMSGD (Balu et al., 2021) in Algorithm 23 with slightly adjusted notations.

By re-organizing, we can further simplify Algorithm 23 to Algorithm 24.

18.2. Detailed Algorithm Description and Connections

Algorithm 22 SlowMo (Wang et al., 2020d). $\mathbf{d}_{i,k}^{(t)}$ indicates the local update direction for communication round t at local update steps k .

```

1: procedure
2:   for  $t \in [T]$  at worker- $i$  in parallel do
3:     Maintain/Average base optimizer buffers
4:     for  $k \in [\tau]$  do
5:       Base optimizer step:  $\mathbf{x}_{i,k+1}^{(t)} = \mathbf{x}_{i,k}^{(t)} - \gamma^{(t)} \mathbf{d}_{i,k}^{(t)}$ 
6:       Exact-Average:  $\mathbf{x}_\tau^{(t)} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_{i,\tau}^{(t)}$ 
7:       Update slow momentum:  $\mathbf{m}^{(t+1)} = \beta \mathbf{m}^{(t)} + \frac{1}{\gamma^{(t)}} (\mathbf{x}_{i,0}^{(t)} - \mathbf{x}_\tau^{(t)})$ 
8:       Update outer iterates:  $\mathbf{x}_{i,0}^{(t+1)} = \mathbf{x}_{i,0}^{(t)} - \alpha \gamma^{(t)} \mathbf{m}^{(t+1)}$ 
9:   return  $\mathbf{x}_T$ 

```

Algorithm 23 Original formulation of DMSGD. $\hat{\mathbf{m}}_i^{(0)}$ are initialized as $\mathbf{0}$ for all workers.

```

1: procedure WORKER- $i$ 
2:   for  $t \in \{1, \dots, T\}$  do
3:      $\mathbf{v}_i^{(t)} = \sum_{j \in \mathcal{N}_i^{(t)}} w_{ij} \mathbf{x}_j^{(t)}$  ▷ Consensus step
4:      $\hat{\mathbf{m}}_i^{(t)} = \mu (\mathbf{x}_i^{(t)} - \mathbf{x}_i^{(t-1)}) + (1 - \mu) (\mathbf{v}_i^{(t)} - \mathbf{v}_i^{(t-1)})$  ▷ Momentum step
5:     sample  $\xi_i^{(t)}$  and compute  $\mathbf{g}_i^{(t)} = \nabla F_i(\mathbf{x}_i^{(t)}, \xi_i^{(t)})$ 
6:      $\mathbf{x}_i^{(t+1)} = \mathbf{v}_i^{(t)} - \eta \mathbf{g}_i^{(t)} + \beta \hat{\mathbf{m}}_i^{(t)}$  ▷ Option I of local gradient step
7:      $\mathbf{x}_i^{(t+1)} = \mathbf{x}_i^{(t)} - \eta \mathbf{g}_i^{(t)} + \beta \hat{\mathbf{m}}_i^{(t)}$  ▷ Option II of local gradient step

```

Algorithm 24 Re-organized formulation of DMSGD. $\hat{\mathbf{m}}_i^{(0)}$ are initialized as $\mathbf{0}$ for all workers.

```

1: procedure WORKER- $i$ 
2:   for  $t \in \{1, \dots, T\}$  do
3:     sample  $\xi_i^{(t)}$  and compute  $\mathbf{g}_i^{(t)} = \nabla F_i(\mathbf{x}_i^{(t-\frac{1}{2})}, \xi_i^{(t)})$ 
4:      $\mathbf{x}_i^{(t+\frac{1}{2})} = \mathbf{x}_i^{(t)} - \eta (\beta \hat{\mathbf{m}}_i^{(t-1)} + \mathbf{g}_i^{(t)})$  ▷ Option I of local gradient step
5:      $\mathbf{x}_i^{(t+\frac{1}{2})} = \mathbf{x}_i^{(t-\frac{1}{2})} - \eta (\beta \hat{\mathbf{m}}_i^{(t-1)} + \mathbf{g}_i^{(t)})$  ▷ Option II of local gradient step
6:      $\mathbf{x}_i^{(t+1)} = \sum_{j \in \mathcal{N}_i^{(t)}} w_{ij} \mathbf{x}_j^{(t+\frac{1}{2})}$  ▷ Consensus step
7:      $\hat{\mathbf{m}}_i^{(t)} = \mu (\mathbf{x}_i^{(t-\frac{1}{2})} - \mathbf{x}_i^{(t+\frac{1}{2})}) + (1 - \mu) (\mathbf{x}_i^{(t)} - \mathbf{x}_i^{(t+1)})$  ▷ Momentum step

```

For a fair comparison, we unify Algorithm 24 with QG-DSGDm, as in Algorithm 25 (we slightly abuse the notations for comparison purpose).

Note that in Algorithm 25 (slightly different from the $\hat{\mathbf{m}}^{(t)}$ in Algorithm 24), $\hat{\mathbf{m}}_i^{(t)}$ in DMSGD is defined as

$$\hat{\mathbf{m}}_i^{(t)} = \frac{\mu (\mathbf{x}_i^{(t-\frac{1}{2})} - \mathbf{x}_i^{(t+\frac{1}{2})}) + (1 - \mu) (\mathbf{x}_i^{(t)} - \mathbf{x}_i^{(t+1)})}{\eta},$$

Chapter 18. Appendix for Quasi-Global Momentum

Algorithm 25 DMSGD v.s. QG-DSGDm. $\hat{\mathbf{m}}_i^{(0)}$ are initialized as $\mathbf{0}$ for all workers.

```

1: procedure WORKER- $i$ 
2:   for  $t \in \{1, \dots, T\}$  do
3:     sample  $\xi_i^{(t)}$  and compute  $\mathbf{g}_i^{(t)} = \nabla F_i(\mathbf{x}_i^{(t)}, \xi_i^{(t)})$ 
4:      $\mathbf{x}_i^{(t+\frac{1}{2})} = \mathbf{x}_i^{(t)} - \eta(\beta\hat{\mathbf{m}}_i^{(t-1)} + \mathbf{g}_i^{(t)})$ 
5:      $\mathbf{x}_i^{(t+1)} = \sum_{j \in \mathcal{N}_i^{(t)}} w_{ij} \mathbf{x}_j^{(t+\frac{1}{2})}$ 
6:      $\hat{\mathbf{m}}_i^{(t)}$  is determined by the algorithm.
7:   return  $\mathbf{x}_i^{(T)}$ 

```

while for QG-DSGDm, we have

$$\hat{\mathbf{m}}_i^{(t)} = \mu\hat{\mathbf{m}}_i^{(t-1)} + (1-\mu)\frac{\mathbf{x}_i^{(t)} - \mathbf{x}_i^{(t+1)}}{\eta}.$$

Thus, for option I of DMSGD, we have

$$\begin{aligned} \hat{\mathbf{m}}_i^{(t)} &= \frac{\mu(\mathbf{x}_i^{(t-\frac{1}{2})} - \mathbf{x}_i^{(t+\frac{1}{2})}) + (1-\omega)(\mathbf{x}_i^{(t)} - \mathbf{x}_i^{(t+1)})}{\eta} \\ &= \frac{\mu}{\eta} \left((\mathbf{x}_i^{(t-1)} - \eta(\beta\hat{\mathbf{m}}_i^{(t-2)} + \mathbf{g}_i^{(t-1)})) - (\mathbf{x}_i^{(t)} - \eta(\beta\hat{\mathbf{m}}_i^{(t-1)} + \mathbf{g}_i^{(t)})) \right) + (1-\mu)\frac{\mathbf{x}_i^{(t)} - \mathbf{x}_i^{(t+1)}}{\eta} \\ &= \mu \left(\frac{\mathbf{x}_i^{(t-1)} - \mathbf{x}_i^{(t)}}{\eta} - \left((\beta\hat{\mathbf{m}}_i^{(t-2)} + \mathbf{g}_i^{(t-1)}) - (\beta\hat{\mathbf{m}}_i^{(t-1)} + \mathbf{g}_i^{(t)}) \right) \right) + (1-\mu)\frac{\mathbf{x}_i^{(t)} - \mathbf{x}_i^{(t+1)}}{\eta} \\ &= \mu \left(\frac{\mathbf{x}_i^{(t-1)} - \mathbf{x}_i^{(t)}}{\eta} - \beta(\hat{\mathbf{m}}_i^{(t-2)} - \hat{\mathbf{m}}_i^{(t-1)}) - (\mathbf{g}_i^{(t-1)} - \mathbf{g}_i^{(t)}) \right) + (1-\mu)\frac{\mathbf{x}_i^{(t)} - \mathbf{x}_i^{(t+1)}}{\eta} \\ &= \mu \left(\beta\hat{\mathbf{m}}_i^{(t-1)} + \mathbf{g}_i^{(t)} + \frac{\mathbf{x}_i^{(t-1)} - \mathbf{x}_i^{(t)}}{\eta} - \beta\hat{\mathbf{m}}_i^{(t-2)} - \mathbf{g}_i^{(t-1)} \right) + (1-\mu)\frac{\mathbf{x}_i^{(t)} - \mathbf{x}_i^{(t+1)}}{\eta}, \end{aligned}$$

for option II of DMSGD, we have

$$\begin{aligned} \hat{\mathbf{m}}_i^{(t)} &= \frac{\mu(\mathbf{x}_i^{(t-\frac{1}{2})} - \mathbf{x}_i^{(t+\frac{1}{2})}) + (1-\omega)(\mathbf{x}_i^{(t)} - \mathbf{x}_i^{(t+1)})}{\eta} \\ &= \mu \left(\beta\hat{\mathbf{m}}_i^{(t-1)} + \mathbf{g}_i^{(t)} \right) + (1-\mu)\frac{\mathbf{x}_i^{(t)} - \mathbf{x}_i^{(t+1)}}{\eta}. \end{aligned}$$

It is obvious that the design of DMSGD is different from our QG-DSGDm:

- The update scheme on the momentum buffer $\hat{\mathbf{m}}_i$ is different, as illustrate above.
- DMSGD is based on the heavy-ball momentum, while our scheme can generalize to heavy ball momentum SGD, Nesterov momentum SGD, and even Adam variants.

18.2.3 Connections Between SGDM and QG-SGDM

Connections Between SGDM and QG-DSGDM

Note our scheme QG-DSGDM on the single worker case (i.e. QG-SGDM) has the following equation:

$$\begin{aligned}\mathbf{x}^{(t+1)} &= \mathbf{x}^{(t)} - \eta (\beta \mathbf{m}^{(t-1)} + \mathbf{g}^{(t)}) \\ \mathbf{m}^t &= \mu \mathbf{m}^{(t-1)} + (1 - \mu) \frac{\mathbf{x}^{(t)} - \mathbf{x}^{(t+1)}}{\eta} = (\mu + (1 - \mu)\beta) \mathbf{m}^{(t-1)} + (1 - \mu)\mathbf{g}^{(t)}.\end{aligned}$$

By letting $\hat{\mathbf{m}}^{(t)} := \frac{\mathbf{m}^{(t)}}{1 - \mu}$, we have

$$\begin{aligned}\mathbf{x}^{(t+1)} &= \mathbf{x}^{(t)} - \eta (\beta(1 - \mu)\hat{\mathbf{m}}^{(t-1)} + \mathbf{g}^{(t)}) \\ \hat{\mathbf{m}}^{(t)} &= (\mu + (1 - \mu)\beta) \hat{\mathbf{m}}^{(t-1)} + \mathbf{g}^{(t)}.\end{aligned}$$

We further let $\hat{\beta} := \mu + (1 - \mu)\beta$, then we have

$$\begin{aligned}\mathbf{x}^{(t+1)} &= \mathbf{x}^{(t)} - \eta (\hat{\beta}\hat{\mathbf{m}}^{(t-1)} + \mathbf{g}^{(t)} + (\beta(1 - \mu) - \hat{\beta})\hat{\mathbf{m}}^{(t-1)}) = \mathbf{x}^{(t)} - \eta (\hat{\beta}\hat{\mathbf{m}}^{(t-1)} + \mathbf{g}^{(t)} - \mu\hat{\mathbf{m}}^{(t-1)}) \\ \hat{\mathbf{m}}^{(t)} &= \hat{\beta}\hat{\mathbf{m}}^{(t-1)} + \mathbf{g}^{(t)}.\end{aligned}$$

By re-organizing, we have

$$\begin{aligned}\hat{\mathbf{m}}^{(t)} &= \hat{\beta}\hat{\mathbf{m}}^{(t-1)} + \mathbf{g}^{(t)} \\ \mathbf{x}^{(t+1)} &= \mathbf{x}^{(t)} - \eta (\hat{\mathbf{m}}^{(t)} - \mu\hat{\mathbf{m}}^{(t-1)}) = \mathbf{x}^{(t)} - \eta (\hat{\mathbf{m}}^{(t)} - \mu\hat{\mathbf{m}}^{(t-1)}) = \mathbf{x}^{(t)} - \eta \left(\left(1 - \frac{\mu}{\hat{\beta}}\right)\hat{\mathbf{m}}^{(t)} + \frac{\mu}{\hat{\beta}}\mathbf{g}^{(t)} \right),\end{aligned}\tag{18.1}$$

which recovers the QHM (Gitman et al., 2019).

Comparing to the case of SGD with Heavy-ball Momentum (SGDM), where

$$\begin{aligned}\hat{\mathbf{m}}^{(t)} &= \beta\hat{\mathbf{m}}^{(t-1)} + \mathbf{g}^{(t)} \\ \mathbf{x}^{(t+1)} &= \mathbf{x}^{(t)} - \eta\hat{\mathbf{m}}^{(t)},\end{aligned}$$

we can witness that SGDM is only a special case of (18.1) (when $\mu=0$).

Connections Between SGDM-N and QG-SGDM

First note that our simplified version of QG-DSGDM (i.e. QG-SGDM) can recover the QHM (Gitman et al., 2019), as illustrated in Appendix 18.2.3. Furthermore, as pointed out in (Gitman et al.,

Chapter 18. Appendix for Quasi-Global Momentum

2019) that, the QHM is indeed equivalent to the original SGDm-N with re-scaling of $\eta \rightarrow \eta/(1-\beta)$. Therefore, we can argue that our simplified version of QG-DSGDm (i.e. QG-SGDm or QHM) is equivalent to the original SGDm-N with re-scaling of $\eta \rightarrow \eta/(1-\beta)$.

For the reason of completeness, we include the derivatives below.

First of all, SGD with Nesterov Momentum (SGDm-N) can be rewritten as

$$\begin{aligned}\mathbf{x}^{(t+\frac{1}{2})} &= \mathbf{x}^{(t)} + \beta\mathbf{m}^{(t-1)} \\ \mathbf{m}^{(t)} &= \beta\mathbf{m}^{(t-1)} - \eta\nabla f(\mathbf{x}^{(t+\frac{1}{2})}) \\ \mathbf{x}^{(t+1)} &= \mathbf{x}^{(t+\frac{1}{2})} - \eta\nabla f(\mathbf{x}^{(t+\frac{1}{2})}),\end{aligned}$$

and in PyTorch, we instead have

$$\begin{aligned}\mathbf{x}^{(t+\frac{1}{2})} &= \mathbf{x}^{(t)} + \beta\mathbf{m}^{(t-1)} \\ \mathbf{m}^{(t)} &= \beta\mathbf{m}^{(t-1)} + \nabla f(\mathbf{x}^{(t+\frac{1}{2})}) \\ \mathbf{x}^{(t+1)} &= \mathbf{x}^{(t)} - \eta\mathbf{m}^{(t)},\end{aligned}\tag{18.2}$$

where the above equations are equivalent for the constant β .

Then we reiterate the derivatives from [Gitman et al. \(2019\)](#) below, from SGDm-N to QHM (which is equivalently Equation (18.1)). The SGDm-N in [Gitman et al. \(2019\)](#) follows

$$\begin{aligned}\mathbf{m}^{(t)} &= \beta\mathbf{m}^{(t-1)} - \eta\nabla f(\mathbf{x}^{(t)} + \beta\mathbf{m}^{(t-1)}) \\ \mathbf{x}^{(t+1)} &= \mathbf{x}^{(t)} + \mathbf{m}^{(t)},\end{aligned}$$

where we can move the learning rate out of the momentum into the iterates update:

$$\begin{aligned}\mathbf{m}^{(t)} &= \beta\mathbf{m}^{(t-1)} + \nabla f(\mathbf{x}^{(t)} - \eta\beta\mathbf{m}^{(t-1)}) \\ \mathbf{x}^{(t+1)} &= \mathbf{x}^{(t)} - \eta\mathbf{m}^{(t)},\end{aligned}\tag{18.3}$$

where the above two methods produce the same sequence of iterates $\mathbf{x}^{(t)}$ if $\mathbf{m}^{(0)}$ is initialized at $\mathbf{0}$. The second equation (18.3) is equivalent to the Pytorch implementation in Equation (18.2).

Let's normalize the momentum update by $1-\beta$:

$$\begin{aligned}\mathbf{m}^{(t)} &= \beta\mathbf{m}^{(t-1)} + (1-\beta)\nabla f(\mathbf{x}^{(t)} - \eta\beta\mathbf{m}^{(t-1)}) \\ \mathbf{x}^{(t+1)} &= \mathbf{x}^{(t)} - \eta\mathbf{m}^{(t)},\end{aligned}$$

which is equivalent to the un-normalized one by re-scaling $\eta \rightarrow \frac{\eta}{1-\beta}$ for constant parameters. We

18.2. Detailed Algorithm Description and Connections

make a change of variables $\mathbf{y}^{(t)} = \mathbf{x}^{(t)} - \eta\beta\mathbf{m}^{(t-1)}$,

$$\begin{aligned}\mathbf{m}^{(t)} &= \beta\mathbf{m}^{(t-1)} + (1-\beta)\nabla f(\mathbf{y}^{(t)}) \\ \mathbf{y}^{(t+1)} &= \mathbf{x}^{(t+1)} - \eta\beta\mathbf{m}^{(t)} = \mathbf{x}^{(t)} - \eta\mathbf{m}^{(t)} - \eta\beta\mathbf{m}^{(t)} \\ &= \mathbf{y}^{(t)} + \eta\beta\mathbf{m}^{(t-1)} - \eta\mathbf{m}^{(t)} - \eta\beta\mathbf{m}^{(t)} \\ &= \mathbf{y}^{(t)} + \eta(\mathbf{m}^{(t)} - (1-\beta)\nabla f(\mathbf{y}^{(t)})) - \eta\mathbf{m}^{(t)} - \eta\beta\mathbf{m}^{(t)} \\ &= \mathbf{y}^{(t)} - \eta((1-\beta)\nabla f(\mathbf{y}^{(t)}) + \beta\mathbf{m}^{(t)}),\end{aligned}$$

where by renaming $\mathbf{y}^{(t)}$ back to $\mathbf{x}^{(t)}$, we obtain the exact formula used in QHM update.

The Simplification of QG-DSGDm-N on Single Worker Case

We can further simplify our scheme QG-DSGDm-N on the single worker case (and obtain QG-SGDm-N):

$$\begin{aligned}\mathbf{x}^{(t+\frac{1}{2})} &= \mathbf{x}^{(t)} + \beta\mathbf{m}^{(t-1)} \\ \hat{\mathbf{m}}^{(t)} &= \beta\mathbf{m}^{(t-1)} + \nabla f(\mathbf{x}^{(t+\frac{1}{2})}) \\ \mathbf{x}^{(t+1)} &= \mathbf{x}^{(t)} - \eta\hat{\mathbf{m}}^{(t)} \\ \mathbf{m}^{(t)} &= \mu\mathbf{m}^{(t-1)} + (1-\mu)\frac{\mathbf{x}^{(t)} - \mathbf{x}^{(t+1)}}{\eta}.\end{aligned}$$

We rewrite $\mathbf{m}^{(t)}$ as

$$\begin{aligned}\mathbf{m}^{(t)} &= \mu\mathbf{m}^{(t-1)} + (1-\mu)\frac{\mathbf{x}^{(t)} - \mathbf{x}^{(t+1)}}{\eta} = \mu\mathbf{m}^{(t-1)} + (1-\mu)\hat{\mathbf{m}}^{(t)} \\ &= (\mu + \beta - \mu\beta)\mathbf{m}^{(t-1)} + (1-\mu)\nabla f(\mathbf{x}^{(t+\frac{1}{2})}).\end{aligned}$$

Similar to the treatment in Appendix 18.2.3, by letting $\hat{\mathbf{m}}^{(t)} := \frac{\mathbf{m}^{(t)}}{1-\mu}$, we have

$$\begin{aligned}\mathbf{x}^{(t+1)} &= \mathbf{x}^{(t)} - \eta\hat{\mathbf{m}}^{(t)} = \mathbf{x}^{(t)} - \eta\left(\beta(1-\mu)\hat{\mathbf{m}}^{(t-1)} + \nabla f(\mathbf{x}^{(t+\frac{1}{2})})\right) \\ \hat{\mathbf{m}}^{(t)} &= (\mu + \beta - \mu\beta)\hat{\mathbf{m}}^{(t-1)} + \nabla f(\mathbf{x}^{(t+\frac{1}{2})}),\end{aligned}$$

and thus, in the end we have the following equations for QG-SGDm-N

$$\begin{aligned}\mathbf{x}^{(t+\frac{1}{2})} &= \mathbf{x}^{(t)} + \beta(1-\mu)\hat{\mathbf{m}}^{(t-1)} \\ \hat{\mathbf{m}}^{(t)} &= \hat{\beta}\hat{\mathbf{m}}^{(t-1)} + \nabla f(\mathbf{x}^{(t+\frac{1}{2})}) \\ \mathbf{x}^{(t+1)} &= \mathbf{x}^{(t)} - \eta\left(\left(1 - \frac{\mu}{\hat{\beta}}\right)\hat{\mathbf{m}}^{(t)} + \frac{\mu}{\hat{\beta}}\nabla f(\mathbf{x}^{(t+\frac{1}{2})})\right),\end{aligned}$$

where $\hat{\beta} := \mu + (1 - \mu)\beta$ and we can recover SGDM-N by setting $\mu = 0$.

18.2.4 Comparison with D^2 and Gradient Tracking (GT) methods

We comment on GT methods below (including D^2 (Tang et al., 2018b)), in order to 1) highlight the distinctions between different algorithms, and 2) justify the comparison with existing GT methods.

- Distinctions between algorithms: 1) Both D^2 and GT do not consider momentum in their algorithm design and theoretical analysis, while one of our main contributions is the design of quasi-global momentum—a simple yet effective approach for the SOTA decentralized deep learning training; 2) It is unclear how to integrate D^2 with momentum, given the original design intuition of D^2 ; 3) QG-DSGDm is different from D^2 , where the updates of QG-DSGDm and D^2 follow $\mathbf{w}((1 + \beta \frac{\eta^{(t)}}{\eta^{(t-1)}})\mathbf{x}^{(t)} - \beta \frac{\eta^{(t)}}{\eta^{(t-1)}}\mathbf{x}^{(t-1)} - \eta^{(t)}\nabla f(\mathbf{x}^{(t)}))$ and $\mathbf{w}(2\mathbf{x}^{(t)} - \mathbf{x}^{(t-1)} - \eta(\nabla f(\mathbf{x}^{(t)}) - \nabla f(\mathbf{x}^{(t-1)})))$ respectively (we simplify the comparison by letting $\mu = 0$ in QG-DSGDm); 4) Compared to QG-DSGDm, GT requires extra one communication step per update to approximate the global average of local gradients.
- D^2 cannot achieve comparable test performance on the standard deep learning benchmark.
 - D^2 requires a constant learning rate, which does not fit the SOTA learning rate schedule (e.g. stage-wise) in deep learning. Note that D^2 can be rewritten as $\mathbf{w}(\mathbf{x}^{(t)} - \eta((\mathbf{x}^{(t-1)} - \mathbf{x}^{(t)})/\eta + \nabla f(\mathbf{x}^{(t)}) - \nabla f(\mathbf{x}^{(t-1)})))$, and the update would break if the magnitude of $\mathbf{x}^{(t-1)} - \mathbf{x}^{(t)}$ is a factor of 10η (i.e. performing learning rate decay at step t).
 - It is non-trivial to improve D^2 for the SOTA deep learning training. To support our argument, Table 9.5 compares to an improved D^2 variant (noted as D_+^2) to address the issue of learning rate decay in D^2 (though it breaks the design intuition of D^2); the performance of D_+^2 is far behind our scheme. The update of D_+^2 follows $\mathbf{w}(\mathbf{x}^{(t)} - \eta^{(t)}((\mathbf{x}^{(t-1)} - \mathbf{x}^{(t)})/\eta^{(t-1)} + \nabla f(\mathbf{x}^{(t)}) - \nabla f(\mathbf{x}^{(t-1)})))$.
 - The numerical results of D^2 in Tang et al. (2018b); Pan et al. (2020); Lu et al. (2019) cannot support the practicability of D^2 : 1) The experiments of Tang et al. (2018b); Pan et al. (2020) only consider a very small scale setup (# of nodes $n=5$ or 8) for CIFAR-10, while Lu et al. (2019) only evaluates on the toy MNIST dataset—these setups are much less challenging than ours; 2) Only training loss curves are reported in Tang et al. (2018b); Pan et al. (2020); Lu et al. (2019), and the final training loss values of Tang et al. (2018b); Pan et al. (2020) are much higher than 0 (i.e. not converge to a local minimum).
- As suggested by anonymous reviewers, we compare with GT methods in Table 9.5: QG-DSGDm-N outperforms GT by a large margin. We would like to point out that 1) the observations of the marginal performance gain in GT are aligned with prior works, e.g. similar numerical results in Figure 3 & 4 & 5 of Xin et al. (2020); 2) GT may have more benefits in the extreme low training loss regime (e.g. less than $1e-4$) where gains might increase when combining with variance reduction techniques (Xin et al., 2020)—however, we focus on the test performance for deep learning (Defazio and Bottou, 2019); 3) recent work (Yuan et al., 2020b) also proves that gradient tracking methods are in general much more sensitive than diffusion-based methods.

18.3 Global Convergence Rate Proofs

We reiterate the update scheme of QG-DSGDm in a matrix form:

$$\begin{aligned}\mathbf{X}^{(t+1)} &= \mathbf{W}(\mathbf{X}^{(t)} - \eta(\beta\mathbf{M}^{(t)} + \mathbf{G}^{(t)})) \\ \mathbf{M}^{(t+1)} &= \mu\mathbf{M}^{(t)} + (1-\mu)\frac{\mathbf{X}^{(t)} - \mathbf{X}^{(t+1)}}{\eta} \\ &= (\mu + (1-\mu)\beta\mathbf{W})\mathbf{M}^{(t)} + (1-\mu)\mathbf{W}\mathbf{G}^{(t)} + \frac{1-\mu}{\eta}(\mathbf{I} - \mathbf{W})\mathbf{X}^{(t)},\end{aligned}\tag{18.4}$$

where our numerical experiments by default use $\mu = \beta$. For each matrix \mathbf{Z} , we define an averaged vector $\bar{\mathbf{z}} = \mathbf{Z}\frac{1}{n}\mathbf{1}$ and matrix $\bar{\mathbf{Z}} = \mathbf{Z}\frac{1}{n}\mathbf{1}\mathbf{1}^\top$. Note that we use bold lower-case to indicate vectors and bold upper-case to denote matrices.

First, we state some standard definitions and regularity conditions.

Assumption 11. *We assume that the following hold:*

1. The function $f(\mathbf{x})$ we are minimizing is lower bounded from below by f^* , and each node's loss f_i is smooth satisfying $\|\nabla f_i(\mathbf{y}) - \nabla f_i(\mathbf{x})\| \leq L\|\mathbf{y} - \mathbf{x}\|$.

The L -smoothness implies the following quadratic upper bound on f_i

$$f_i(\mathbf{y}) \leq f_i(\mathbf{x}) + \langle \nabla f_i(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \frac{L}{2}\|\mathbf{y} - \mathbf{x}\|^2.$$

If additionally the function f is convex and \mathbf{x}^* is an optimum of f , then it also implies

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{x}^*)\| \leq 2L(f(\mathbf{x}) - f^*).$$

2. The stochastic gradients within each node satisfies $\mathbb{E}[\mathbf{g}_i(\mathbf{x})] = \nabla f_i(\mathbf{x})$ and $\mathbb{E}\|\mathbf{g}_i(\mathbf{x}) - \nabla f_i(\mathbf{x})\|^2 \leq \sigma^2$. The variance across the workers is also bounded as $\frac{1}{n}\sum_{i=1}^n \|\nabla f_i(\mathbf{x}) - \nabla f(\mathbf{x})\|^2 \leq \zeta^2$.
3. The mixing matrix is doubly stochastic where for the all ones vector $\mathbf{1}$, we have $\mathbf{W}\mathbf{1} = \mathbf{1}$ and $\mathbf{W}^\top\mathbf{1} = \mathbf{1}$. Further, define $\bar{\mathbf{Z}} = \mathbf{Z}\frac{1}{n}\mathbf{1}\mathbf{1}^\top$ for any matrix $\mathbf{Z} \in \mathbb{R}^{d \times n}$. Then, the mixing matrix satisfies $\mathbb{E}_{\mathbf{W}}\|\mathbf{Z}\mathbf{W} - \bar{\mathbf{Z}}\|_F^2 \leq (1-\rho)\|\mathbf{Z} - \bar{\mathbf{Z}}\|_F^2$.

Average parameters. Let us examine the effect of updates in (18.4) on $\bar{\mathbf{x}}^{(t)}$ which is the parameters averaged across the nodes. Note that since \mathbf{W} is doubly stochastic, we can simplify the updates as follows:

$$\begin{aligned}\bar{\mathbf{x}}^{(t+1)} &= \bar{\mathbf{x}}^{(t)} - \eta(\beta\bar{\mathbf{m}}^{(t)} + \bar{\mathbf{g}}^{(t)}), \text{ and} \\ \bar{\mathbf{m}}^{(t+1)} &= \mu\bar{\mathbf{m}}^{(t)} + (1-\mu)\frac{\bar{\mathbf{x}}^{(t)} - \bar{\mathbf{x}}^{(t+1)}}{\eta} \\ &= (1 - (1-\mu)(1-\beta))\bar{\mathbf{m}}^{(t)} + (1-\mu)\bar{\mathbf{g}}^{(t)}.\end{aligned}\tag{18.5}$$

Chapter 18. Appendix for Quasi-Global Momentum

Here, $\bar{\mathbf{g}}^{(t)} := \frac{1}{n} \sum_{i=1}^n \nabla F_i(\mathbf{x}_i^{(t)}, \xi_i^{(t)})$ is the average of the stochastic gradients across the nodes.

Virtual Sequence. Now we define a virtual sequence of parameters $\{\hat{\mathbf{x}}^{(t)}\}$ which has a simple SGD style update, and an error sequence which will be easy to analyze:

$$\begin{aligned}\hat{\mathbf{x}}^{(t+1)} &= \hat{\mathbf{x}}^{(t)} - \frac{\eta}{1-\beta} \bar{\mathbf{g}}^{(t)}, \text{ and} \\ \bar{\mathbf{e}}^{(t)} &:= \hat{\mathbf{x}}^{(t)} - \bar{\mathbf{x}}^{(t)}.\end{aligned}\tag{18.6}$$

Our strategy for the analysis will be to analyze the virtual sequence $\{\hat{\mathbf{x}}^{(t)}\}$ and prove that the real sequence of iterates remains close i.e. that the $\mathbf{e}^{(t)}$ remains small.

Single-step progress of virtual update. We will show that every step we make some progress, but have to balance three sources of error: i) the stochastic error which depends on σ^2 due to using stochastic gradients, ii) consensus error which depends on $\mathbf{X}^t - \bar{\mathbf{X}}^t$, and finally iii) momentum error due to using momentum which depends on $\mathbf{e}^{(t)}$.

Lemma 18.3.1 (Non-convex one step progress). *Given Assumption 11, the sequence of iterates generated by (18.4) using $\eta \leq \frac{1-\beta}{4L}$ satisfy*

$$\mathbb{E}f(\hat{\mathbf{x}}^{(t+1)}) \leq \mathbb{E}f(\hat{\mathbf{x}}^{(t)}) - \frac{\tilde{\eta}}{4} \|\nabla f(\bar{\mathbf{x}}^{(t)})\|^2 - \frac{\tilde{\eta}}{4} \mathbb{E} \left\| \frac{1}{n} \sum_{i=1}^n \nabla f_i(\mathbf{x}_i^{(t)}) \right\|^2 + \frac{L\tilde{\eta}^2\sigma^2}{n} + \frac{3L^2\tilde{\eta}}{2} \|\mathbf{e}^{(t)}\|^2 + \frac{3L^2\tilde{\eta}}{n} \|\mathbf{X}^{(t)} - \bar{\mathbf{X}}^{(t)}\|_F^2,$$

where we define $\tilde{\eta} := \frac{\eta}{1-\beta}$.

Proof. Starting from the smoothness of f , we have

$$\begin{aligned}
 \mathbb{E}f(\hat{\mathbf{x}}^{(t+1)}) &\leq \mathbb{E}f(\hat{\mathbf{x}}^{(t)}) + \mathbb{E}\langle \nabla f(\hat{\mathbf{x}}^{(t)}), \hat{\mathbf{x}}^{(t+1)} - \hat{\mathbf{x}}^{(t)} \rangle + \frac{L}{2} \mathbb{E} \|\hat{\mathbf{x}}^{(t+1)} - \hat{\mathbf{x}}^{(t)}\|^2 \\
 &= \mathbb{E}f(\hat{\mathbf{x}}^{(t)}) - \frac{\eta}{1-\beta} \frac{1}{n} \sum_{i=1}^n \mathbb{E} \langle \nabla f(\hat{\mathbf{x}}^{(t)}), \nabla f_i(\mathbf{x}_i^{(t)}) \rangle + \frac{L\eta^2}{(1-\beta)^2} \mathbb{E} \|\bar{\mathbf{g}}^{(t)}\|^2 \\
 &\leq \mathbb{E}f(\hat{\mathbf{x}}^{(t)}) - \tilde{\eta} \frac{1}{n} \sum_{i=1}^n \mathbb{E} \langle \nabla f(\hat{\mathbf{x}}^{(t)}), \nabla f_i(\mathbf{x}_i^{(t)}) \rangle + L\tilde{\eta}^2 \mathbb{E} \left\| \frac{1}{n} \sum_{i=1}^n \nabla f_i(\mathbf{x}_i^{(t)}) \right\|^2 + \frac{L\tilde{\eta}^2 \sigma^2}{n} \\
 &= \mathbb{E}f(\hat{\mathbf{x}}^{(t)}) + L\tilde{\eta}^2 \mathbb{E} \left\| \frac{1}{n} \sum_{i=1}^n \nabla f_i(\mathbf{x}_i^{(t)}) \right\|^2 + \frac{L\tilde{\eta}^2 \sigma^2}{n} \\
 &\quad - \frac{\tilde{\eta}}{2} \mathbb{E} \left\| \frac{1}{n} \sum_{i=1}^n \nabla f_i(\mathbf{x}_i^{(t)}) \right\|^2 - \frac{\tilde{\eta}}{2} \mathbb{E} \|\nabla f(\hat{\mathbf{x}}^{(t)})\|^2 + \frac{\tilde{\eta}}{2} \mathbb{E} \left\| \frac{1}{n} \sum_{i=1}^n \nabla f_i(\mathbf{x}_i^{(t)}) - \nabla f(\hat{\mathbf{x}}^{(t)}) \right\|^2 \\
 &\leq \mathbb{E}f(\hat{\mathbf{x}}^{(t)}) + (L\tilde{\eta}^2 - \tilde{\eta}/2) \mathbb{E} \left\| \frac{1}{n} \sum_{i=1}^n \nabla f_i(\mathbf{x}_i^{(t)}) \right\|^2 + \frac{L\tilde{\eta}^2 \sigma^2}{n} \\
 &\quad - \frac{\tilde{\eta}}{4} \mathbb{E} \|\nabla f(\hat{\mathbf{x}}^{(t)})\|^2 + \frac{\tilde{\eta}}{2} \mathbb{E} \|\nabla f(\hat{\mathbf{x}}^{(t)}) - \nabla f(\bar{\mathbf{x}}^{(t)})\|^2 + \frac{\tilde{\eta}}{2} \mathbb{E} \left\| \frac{1}{n} \sum_{i=1}^n \nabla f_i(\mathbf{x}_i^{(t)}) - \nabla f(\bar{\mathbf{x}}^{(t)}) \right\|^2 \\
 &\leq \mathbb{E}f(\hat{\mathbf{x}}^{(t)}) - \frac{\tilde{\eta}}{4} \mathbb{E} \|\nabla f(\bar{\mathbf{x}}^{(t)})\|^2 + \frac{L\tilde{\eta}^2 \sigma^2}{n} - \tilde{\eta}/4 \mathbb{E} \left\| \frac{1}{n} \sum_{i=1}^n \nabla f_i(\mathbf{x}_i^{(t)}) \right\|^2 \\
 &\quad + \frac{3\tilde{\eta}}{2} \mathbb{E} \|\nabla f(\hat{\mathbf{x}}^{(t)}) - \nabla f(\bar{\mathbf{x}}^{(t)})\|^2 + \frac{\tilde{\eta}}{n} \sum_{i=1}^n \mathbb{E} \|\nabla f_i(\mathbf{x}_i^{(t)}) - \nabla f_i(\bar{\mathbf{x}}^{(t)})\|^2.
 \end{aligned}$$

In the last inequality we used our bound on the step-size that $\eta \leq \frac{1-\beta}{4L}$. In the rest of the inequalities, we repeatedly use the identity that $2ab = -a^2 - b^2 + (a-b)^2$. Finally, using the smoothness of the function f and the definition $\tilde{\eta} := \frac{\eta}{1-\beta}$, we get

$$\mathbb{E}f(\hat{\mathbf{x}}^{(t+1)}) \leq \mathbb{E}f(\hat{\mathbf{x}}^{(t)}) - \frac{\tilde{\eta}}{4} \mathbb{E} \|\nabla f(\bar{\mathbf{x}}^{(t)})\|^2 + \frac{L\tilde{\eta}^2 \sigma^2}{n} + \frac{3L^2 \tilde{\eta}}{2} \|\bar{\mathbf{x}}^{(t)} - \hat{\mathbf{x}}^{(t)}\|^2 + \frac{L^2 \tilde{\eta}}{n} \sum_{i=1}^n \|\mathbf{x}_i^{(t)} - \bar{\mathbf{x}}^{(t)}\|^2.$$

Recalling the definition of $\mathbf{e}^{(t)}$ and $\mathbf{X}^{(t)}$ yields the lemma. \square

Lemma 18.3.2 (Strongly-convex one step progress). *Suppose that the set of functions $\{f_i\}$ are μ -strongly convex in addition to Assumption 11. Then the sequence of iterates generated by (18.4) using $\eta \leq \frac{1-\beta}{4L}$ satisfy for $\tilde{\eta} := \frac{\eta}{1-\beta}$,*

$$\mathbb{E} \|\hat{\mathbf{x}}^{(t+1)} - \mathbf{x}^*\|^2 \leq (1-\mu\tilde{\eta}/2) \mathbb{E} \|\hat{\mathbf{x}}^{(t)} - \mathbf{x}^*\|^2 + \frac{\tilde{\eta}^2 \sigma^2}{n} - 3\tilde{\eta}/4 (f(\bar{\mathbf{x}}^{(t)}) - f^*) + 8L\tilde{\eta} \|\mathbf{e}^{(t)}\|^2 + \frac{5L\tilde{\eta}}{n} \|\mathbf{X}^t - \bar{\mathbf{X}}^t\|_F^2.$$

Proof. Starting from the update rule for $\hat{\mathbf{x}}^{(t+1)}$, and expanding very similar to the steps performed

in the previous lemma we get,

$$\begin{aligned}
 \mathbb{E} \|\hat{\mathbf{x}}^{(t+1)} - \mathbf{x}^*\|^2 &= \mathbb{E} \|\hat{\mathbf{x}}^{(t)} - \mathbf{x}^*\|^2 + 2\mathbb{E} \langle \hat{\mathbf{x}}^{(t+1)} - \hat{\mathbf{x}}^{(t)}, \hat{\mathbf{x}}^{(t)} - \mathbf{x}^* \rangle + \mathbb{E} \|\hat{\mathbf{x}}^{(t+1)} - \hat{\mathbf{x}}^{(t)}\|^2 \\
 &\leq \mathbb{E} \|\hat{\mathbf{x}}^{(t)} - \mathbf{x}^*\|^2 - \frac{2\tilde{\eta}}{n} \sum_{i=1}^n \langle \nabla f_i(\mathbf{x}_i^{(t)}), \hat{\mathbf{x}}^{(t)} - \mathbf{x}^* \rangle \\
 &\quad + \frac{\tilde{\eta}^2}{n} \sum_{i=1}^n \mathbb{E} \|\nabla f_i(\mathbf{x}_i^{(t)}) - \nabla f_i(\bar{\mathbf{x}}^{(t)})\|^2 + \tilde{\eta}^2 \mathbb{E} \|\nabla f(\bar{\mathbf{x}}^{(t)})\|^2 + \frac{\tilde{\eta}^2 \sigma^2}{n} \\
 &\leq \mathbb{E} \|\hat{\mathbf{x}}^{(t)} - \mathbf{x}^*\|^2 - 2\tilde{\eta} \underbrace{\frac{1}{n} \sum_{i=1}^n \langle \nabla f_i(\mathbf{x}_i^{(t)}), \hat{\mathbf{x}}^{(t)} - \mathbf{x}^* \rangle}_{\mathcal{F}_1} \\
 &\quad + \frac{\tilde{\eta}^2 L^2}{n} \sum_{i=1}^n \mathbb{E} \|\mathbf{x}_i^{(t)} - \bar{\mathbf{x}}^{(t)}\|^2 + 2L\tilde{\eta}^2 \mathbb{E}(f(\bar{\mathbf{x}}^{(t)}) - f^*) + \frac{\tilde{\eta}^2 \sigma^2}{n}.
 \end{aligned}$$

We will examine the term \mathcal{F}_1 now. Using strong convexity and smoothness of each of the functions $\{f_i\}$, we have

$$\begin{aligned}
 \frac{1}{n} \sum_{i=1}^n \langle \nabla f_i(\mathbf{x}_i), \mathbf{x}_i - \mathbf{x}^* \rangle &\geq \frac{1}{n} \sum_{i=1}^n f_i(\mathbf{x}_i) - f(\mathbf{x}^*) + \frac{\mu}{2} \frac{1}{n} \sum_{i=1}^n \|\mathbf{x}_i - \mathbf{x}^*\|^2 \\
 &\geq \frac{1}{n} \sum_{i=1}^n f_i(\mathbf{x}_i) - f(\mathbf{x}^*) + \frac{\mu}{4} \|\hat{\mathbf{x}} - \mathbf{x}^*\|^2 - \frac{\mu}{2n} \sum_{i=1}^n \|\mathbf{x}_i - \hat{\mathbf{x}}\|^2 \\
 &\geq \frac{1}{n} \sum_{i=1}^n f_i(\mathbf{x}_i) - f(\mathbf{x}^*) + \frac{\mu}{4} \|\hat{\mathbf{x}} - \mathbf{x}^*\|^2 - \frac{\mu}{n} \sum_{i=1}^n \|\mathbf{x}_i - \bar{\mathbf{x}}\|^2 - \mu \|\hat{\mathbf{x}} - \bar{\mathbf{x}}\|^2 \\
 &\geq \frac{1}{n} \sum_{i=1}^n f_i(\mathbf{x}_i) - f(\mathbf{x}^*) + \frac{\mu}{4} \|\hat{\mathbf{x}} - \mathbf{x}^*\|^2 - \frac{L}{n} \sum_{i=1}^n \|\mathbf{x}_i - \bar{\mathbf{x}}\|^2 - L \|\hat{\mathbf{x}} - \bar{\mathbf{x}}\|^2,
 \end{aligned}$$

and

$$\frac{1}{n} \sum_{i=1}^n \langle \nabla f_i(\mathbf{x}_i), \bar{\mathbf{x}} - \mathbf{x}_i \rangle \geq f(\bar{\mathbf{x}}) - \frac{1}{n} \sum_{i=1}^n f_i(\mathbf{x}_i) - \frac{L}{2n} \sum_{i=1}^n \|\mathbf{x}_i - \bar{\mathbf{x}}\|^2,$$

and finally,

$$\begin{aligned}
 \frac{1}{n} \sum_{i=1}^n \langle \nabla f_i(\mathbf{x}_i), \hat{\mathbf{x}} - \bar{\mathbf{x}} \rangle &= \left\langle \frac{1}{n} \sum_{i=1}^n (\nabla f_i(\mathbf{x}_i) \pm \nabla f_i(\bar{\mathbf{x}})), \hat{\mathbf{x}} - \bar{\mathbf{x}} \right\rangle \\
 &\geq -\frac{1}{8L} \left\| \frac{1}{n} \sum_{i=1}^n (\nabla f_i(\mathbf{x}_i) \pm \nabla f_i(\bar{\mathbf{x}})) \right\|^2 - 2L \|\hat{\mathbf{x}} - \bar{\mathbf{x}}\|^2 \\
 &\geq -\frac{1}{4Ln} \sum_{i=1}^n \|\nabla f_i(\mathbf{x}_i) - \nabla f_i(\bar{\mathbf{x}})\|^2 - \frac{1}{4L} \|\nabla f(\bar{\mathbf{x}})\|^2 - 2L \|\hat{\mathbf{x}} - \bar{\mathbf{x}}\|^2 \\
 &\geq -\frac{L}{2n} \sum_{i=1}^n \|\mathbf{x}_i - \bar{\mathbf{x}}\|^2 - \frac{1}{2} (f(\bar{\mathbf{x}}) - f(\mathbf{x}^*)) - 3L \|\hat{\mathbf{x}} - \bar{\mathbf{x}}\|^2,
 \end{aligned}$$

where the first inequality uses the basic inequality $\langle \mathbf{a}, \mathbf{b} \rangle \leq \frac{\beta}{2} \|\mathbf{a}\|^2 + \frac{1}{2\beta} \|\mathbf{b}\|^2, \forall \beta > 0$, and the second inequality uses the relaxed triangle inequality $\|\mathbf{a} + \mathbf{b}\|^2 \leq (1 + \alpha) \|\mathbf{a}\|^2 + (1 + \alpha^{-1}) \|\mathbf{b}\|^2, \forall \alpha > 0$.

Adding up the three inequalities together yields the following expression for the term \mathcal{T}_1

$$\frac{1}{n} \sum_{i=1}^n \langle \nabla f_i(\mathbf{x}_i^{(t)}), \hat{\mathbf{x}}^{(t)} - \mathbf{x}^* \rangle \geq \frac{1}{2} (f(\bar{\mathbf{x}}^{(t)}) - f(\mathbf{x}^*)) + \frac{\mu}{4} \|\hat{\mathbf{x}}^{(t)} - \mathbf{x}^*\|^2 - \frac{2L}{n} \sum_{i=1}^n \|\mathbf{x}_i^{(t)} - \bar{\mathbf{x}}^{(t)}\|^2 - 4L \|\hat{\mathbf{x}}^{(t)} - \bar{\mathbf{x}}^{(t)}\|^2.$$

Plugging this back into the previous inequality and using $\tilde{\eta} \leq \frac{1}{4L}$ finishes the proof of the lemma.

$$\begin{aligned} \mathbb{E} \|\hat{\mathbf{x}}^{(t+1)} - \mathbf{x}^*\|^2 &\leq (1 - \tilde{\eta}\mu/2) \mathbb{E} \|\hat{\mathbf{x}}^{(t)} - \mathbf{x}^*\|^2 - \tilde{\eta}(1 - 2L\tilde{\eta}) \mathbb{E} (f(\bar{\mathbf{x}}^{(t)}) - f^*) + \frac{\tilde{\eta}^2 \sigma^2}{n} \\ &\quad + \frac{\tilde{\eta}^2 L^2 + 4\tilde{\eta}L}{n} \sum_{i=1}^n \mathbb{E} \|\mathbf{x}_i^{(t)} - \bar{\mathbf{x}}^{(t)}\|^2 + 8L\tilde{\eta} \|\hat{\mathbf{x}}^{(t)} - \bar{\mathbf{x}}^{(t)}\|^2. \quad \square \end{aligned}$$

Bounding the consensus error. We will now try to bound the consensus error $(\mathbf{X}^t - \bar{\mathbf{X}}^t)$ between the node's parameters and its average. During each step, we perform a diffusion step (communication with neighbors) which brings the parameters of the nodes closer to each other. However we also perform additional gradient/momentum steps which moves the distance away from each other.

Lemma 18.3.3 (One step consensus change). *Given Assumption 11, the sequence of iterates generated by (18.4) using $\eta \leq \frac{\rho}{7L}$ satisfy for $\tilde{\eta} := \frac{\eta}{1-\rho}$,*

$$\frac{1}{n} \mathbb{E} \|\mathbf{X}^{t+1} - \bar{\mathbf{X}}^{t+1}\|_F^2 \leq \frac{(1-\rho/4)}{n} \mathbb{E} \|\mathbf{X}^t - \bar{\mathbf{X}}^t\|_F^2 + \frac{12\eta^2 \zeta^2}{\rho} + 4(1-\rho)\eta^2 \sigma^2 + \frac{6\eta^2 \beta^2}{\rho n} \mathbb{E} \|\mathbf{M}^{(t)} - \bar{\mathbf{M}}^{(t)}\|_F^2.$$

Proof. Starting from the update step (18.4),

$$\begin{aligned} \frac{1}{n} \mathbb{E} \|\mathbf{X}^{t+1} - \bar{\mathbf{X}}^{t+1}\|_F^2 &= \frac{1}{n} \mathbb{E} \|\mathbf{W}(\mathbf{X}^{(t)} - \eta(\beta \mathbf{M}^{(t)} + \mathbf{G}^{(t)})) - (\bar{\mathbf{X}}^{(t)} - \eta(\beta \bar{\mathbf{M}}^{(t)} + \bar{\mathbf{G}}^{(t)}))\|_F^2 \\ &\leq \frac{1-\rho}{n} \mathbb{E} \|(\mathbf{X}^{(t)} - \eta(\beta \mathbf{M}^{(t)} + \mathbf{G}^{(t)})) - (\bar{\mathbf{X}}^{(t)} - \eta(\beta \bar{\mathbf{M}}^{(t)} + \bar{\mathbf{G}}^{(t)}))\|_F^2 \\ &\leq \frac{1-\rho}{n} \mathbb{E} \|(\mathbf{X}^{(t)} - \eta(\beta \mathbf{M}^{(t)} + \mathbb{E}_t[\mathbf{G}^{(t)}])) - (\bar{\mathbf{X}}^{(t)} - \eta(\beta \bar{\mathbf{M}}^{(t)} + \mathbb{E}_t[\bar{\mathbf{G}}^{(t)}]))\|_F^2 \\ &\quad + 4(1-\rho)\eta^2 \sigma^2 \\ &\leq \frac{(1-\rho)(1+\rho/2)}{n} \mathbb{E} \|\mathbf{X}^{(t)} - \bar{\mathbf{X}}^{(t)}\|_F^2 + \frac{6\eta^2 \beta^2}{\rho n} \mathbb{E} \|\mathbf{M}^{(t)} - \bar{\mathbf{M}}^{(t)}\|_F^2 \\ &\quad + \frac{6\eta^2}{\rho n} \mathbb{E} \|\mathbb{E}_t[\mathbf{G}^{(t)}] - \mathbb{E}_t[\bar{\mathbf{G}}^{(t)}]\|_F^2 + 4(1-\rho)\eta^2 \sigma^2. \end{aligned}$$

Here we used the contractivity of the mixing matrix and Young's inequality. We can proceed as

$$\begin{aligned}
 \frac{1}{n} \mathbb{E} \|\mathbf{X}^{t+1} - \bar{\mathbf{X}}^{t+1}\|_F^2 &\leq \frac{(1-\rho/2)}{n} \mathbb{E} \|\mathbf{X}^{(t)} - \bar{\mathbf{X}}^{(t)}\|_F^2 + \frac{6\eta^2 \beta^2}{\rho n} \mathbb{E} \|\mathbf{M}^{(t)} - \bar{\mathbf{M}}^{(t)}\|_F^2 + 4(1-\rho)\eta^2 \sigma^2 \\
 &\quad + \frac{6\eta^2}{\rho n} \mathbb{E} \|\mathbb{E}_t [\mathbf{G}^{(t)}] - \nabla f(\bar{\mathbf{x}}^{(t)})\|_F^2 \\
 &= \frac{(1-\rho/2)}{n} \mathbb{E} \|\mathbf{X}^{(t)} - \bar{\mathbf{X}}^{(t)}\|_F^2 + \frac{6\eta^2 \beta^2}{\rho n} \mathbb{E} \|\mathbf{M}^{(t)} - \bar{\mathbf{M}}^{(t)}\|_F^2 + 4(1-\rho)\eta^2 \sigma^2 \\
 &\quad + \frac{6\eta^2}{\rho n} \sum_{i=1}^n \mathbb{E} \left\| \nabla f_i(\mathbf{x}_i^{(t)}) \pm \nabla f_i(\bar{\mathbf{x}}^{(t)}) - \nabla f(\bar{\mathbf{x}}^{(t)}) \right\|^2 \\
 &\leq \frac{(1-\rho/2)}{n} \mathbb{E} \|\mathbf{X}^{(t)} - \bar{\mathbf{X}}^{(t)}\|_F^2 + \frac{6\eta^2 \beta^2}{\rho n} \mathbb{E} \|\mathbf{M}^{(t)} - \bar{\mathbf{M}}^{(t)}\|_F^2 + 4(1-\rho)\eta^2 \sigma^2 \\
 &\quad + \frac{12\eta^2}{\rho n} \sum_{i=1}^n \mathbb{E} \left\| \nabla f_i(\mathbf{x}_i^{(t)}) - \nabla f_i(\bar{\mathbf{x}}^{(t)}) \right\|^2 + \frac{12\eta^2}{\rho n} \sum_{i=1}^n \mathbb{E} \left\| \nabla f_i(\bar{\mathbf{x}}^{(t)}) - \nabla f(\bar{\mathbf{x}}^{(t)}) \right\|^2 \\
 &\leq \frac{(1-\rho/2)}{n} \mathbb{E} \|\mathbf{X}^{(t)} - \bar{\mathbf{X}}^{(t)}\|_F^2 + \frac{6\eta^2 \beta^2}{\rho n} \mathbb{E} \|\mathbf{M}^{(t)} - \bar{\mathbf{M}}^{(t)}\|_F^2 + 4(1-\rho)\eta^2 \sigma^2 \\
 &\quad + \frac{12\eta^2 L^2}{\rho n} \sum_{i=1}^n \mathbb{E} \left\| \mathbf{x}_i^{(t)} - \bar{\mathbf{x}}^{(t)} \right\|^2 + \frac{12\eta^2 \zeta^2}{\rho}.
 \end{aligned}$$

Our assumption that the step size $\eta \leq \frac{\rho}{7L}$ ensures that $12\eta^2 L^2 \leq \rho^2/4$, finishing the proof. \square

We will now try to bound the momentum error $(\mathbf{X}^t - \bar{\mathbf{X}}^t)$ between the momentum on each node and its average across nodes.

Lemma 18.3.4 (One step momentum change). *Given Assumption 11, the sequence of iterates generated by (18.4) using momentum satisfying $\frac{\beta}{1-\beta} \leq \frac{\rho}{21}$,*

$$\begin{aligned}
 \frac{6\eta^2 \beta^2}{n\rho(1-\mu)(1-\beta)} \mathbb{E} \|\mathbf{M}^{t+1} - \bar{\mathbf{M}}^{t+1}\|_F^2 &\leq \left(\frac{6\eta^2 \beta^2}{n\rho(1-\mu)(1-\beta)} - \frac{6\eta^2 \beta^2}{n\rho} \right) \mathbb{E} \|\mathbf{M}^{(t)} - \bar{\mathbf{M}}^{(t)}\|_F^2 \\
 &\quad + \frac{\rho}{8n} \mathbb{E} \|\mathbf{X}^{(t)} - \bar{\mathbf{X}}^{(t)}\|_F^2 + \frac{\eta^2 \rho \zeta^2}{8} + \frac{\eta^2 \rho \sigma^2 (1-\beta)}{8(1-\mu)}.
 \end{aligned}$$

Proof. Starting from the update step (18.4) and proceeding similar to the previous lemma, we

have

$$\begin{aligned}
 & \frac{1}{n} \mathbb{E} \left\| \mathbf{M}^{(t+1)} - \bar{\mathbf{M}}^{(t+1)} \right\|_F^2 \\
 &= \frac{1}{n} \mathbb{E} \left\| (\mu \mathbf{I} + (1-\mu)\beta \mathbf{W})(\mathbf{M}^{(t)} - \bar{\mathbf{M}}^{(t)}) + (1-\mu)\mathbf{W}(\mathbf{G}^{(t)} - \bar{\mathbf{G}}^{(t)}) + \frac{1-\mu}{\eta}(\mathbf{I} - \mathbf{W})\mathbf{X}^{(t)} \right\|_F^2 \\
 &= \frac{1}{n} \mathbb{E} \left\| (\mu \mathbf{I} + (1-\mu)\beta \mathbf{W})(\mathbf{M}^{(t)} - \bar{\mathbf{M}}^{(t)}) + \frac{1-\mu}{\eta}(\mathbf{I} - \mathbf{W})\mathbf{X}^{(t)} + (1-\mu)\mathbf{W}(\mathbb{E}[\mathbf{G}^{(t)} - \bar{\mathbf{G}}^{(t)}]) \right\|_F^2 \\
 &\quad + \frac{1}{n} \mathbb{E} \left\| (1-\mu)\mathbf{W}(\mathbf{G}^{(t)} - \mathbb{E}[\mathbf{G}^{(t)}]) - (\bar{\mathbf{G}}^{(t)} - \mathbb{E}[\bar{\mathbf{G}}^{(t)}]) \right\|_F^2 \\
 &= \frac{1}{n} \mathbb{E} \left\| (\mu \mathbf{I} + (1-\mu)\beta \mathbf{W})(\mathbf{M}^{(t)} - \bar{\mathbf{M}}^{(t)}) + \frac{1-\mu}{\eta}(\mathbf{I} - \mathbf{W})\mathbf{X}^{(t)} + (1-\mu)\mathbf{W}(\mathbb{E}[\mathbf{G}^{(t)} - \bar{\mathbf{G}}^{(t)}]) \right\|_F^2 + 4\sigma^2 \\
 &\leq \frac{1}{n} \left(1 + \frac{(1-\mu)(1-\beta)}{1-(1-\mu)(1-\beta)} \right) \mathbb{E} \left\| (\mu \mathbf{I} + (1-\mu)\beta \mathbf{W})(\mathbf{M}^{(t)} - \bar{\mathbf{M}}^{(t)}) \right\|_F^2 + 4\sigma^2 \\
 &\quad + \frac{1}{n} \left(1 + \frac{1-(1-\mu)(1-\beta)}{(1-\mu)(1-\beta)} \right) \mathbb{E} \left\| \frac{1-\mu}{\eta}(\mathbf{I} - \mathbf{W})\mathbf{X}^{(t)} + (1-\mu)\mathbf{W}(\mathbb{E}[\mathbf{G}^{(t)} - \bar{\mathbf{G}}^{(t)}]) \right\|_F^2.
 \end{aligned}$$

Note that since $\mathbf{W} < \mathbf{I}$, we have $(\mu \mathbf{I} + (1-\mu)\beta \mathbf{W}) < (\mu + (1-\mu)\beta)\mathbf{I} = (1-(1-\beta)(1-\mu))\mathbf{I}$. Further, since $-\mathbf{I} < \mathbf{W}$, we have $\mathbf{I} - \mathbf{W} < 2\mathbf{I}$. With these observations, we can continue

$$\begin{aligned}
 & \frac{1}{n} \mathbb{E} \left\| \mathbf{M}^{(t+1)} - \bar{\mathbf{M}}^{(t+1)} \right\|_F^2 \\
 &\leq \frac{1}{n} \left(1 + \frac{(1-\mu)(1-\beta)}{1-(1-\mu)(1-\beta)} \right) \mathbb{E} \left\| (1-(1-\mu)(1-\beta))(\mathbf{M}^{(t)} - \bar{\mathbf{M}}^{(t)}) \right\|_F^2 + 4\sigma^2 \\
 &\quad + \frac{1}{n} \left(1 + \frac{1-(1-\mu)(1-\beta)}{(1-\mu)(1-\beta)} \right) \mathbb{E} \left\| \frac{1-\mu}{\eta}(\mathbf{I} - \mathbf{W})\mathbf{X}^{(t)} + (1-\mu)\mathbf{W}(\mathbb{E}[\mathbf{G}^{(t)} - \bar{\mathbf{G}}^{(t)}]) \right\|_F^2 \\
 &\leq \frac{1}{n} (1-(1-\mu)(1-\beta)) \mathbb{E} \left\| \mathbf{M}^{(t)} - \bar{\mathbf{M}}^{(t)} \right\|_F^2 + 4\sigma^2 \\
 &\quad + \frac{1}{(1-\mu)(1-\beta)n} \mathbb{E} \left\| \frac{1-\mu}{\eta}(\mathbf{I} - \mathbf{W})\mathbf{X}^{(t)} + (1-\mu)\mathbf{W}(\mathbb{E}[\mathbf{G}^{(t)} - \bar{\mathbf{G}}^{(t)}]) \right\|_F^2 \\
 &\leq \frac{1}{n} (1-(1-\mu)(1-\beta)) \mathbb{E} \left\| \mathbf{M}^{(t)} - \bar{\mathbf{M}}^{(t)} \right\|_F^2 + 4\sigma^2 \\
 &\quad + \frac{4(1-\mu)}{(1-\beta)n\eta^2} \mathbb{E} \left\| \mathbf{X}^{(t)} - \bar{\mathbf{X}}^{(t)} \right\|_F^2 + \frac{2(1-\mu)}{(1-\beta)n} \mathbb{E} \left\| \mathbb{E}[\mathbf{G}^{(t)}] - \bar{\mathbf{G}}^{(t)} \right\|_F^2.
 \end{aligned}$$

From the proof of the previous lemma, we can simplify the last term as

$$\begin{aligned}
\frac{1}{n} \mathbb{E} \|\mathbf{M}^{(t+1)} - \bar{\mathbf{M}}^{(t+1)}\|_F^2 &\leq \frac{1}{n} (1 - (1 - \mu)(1 - \beta)) \mathbb{E} \|\mathbf{M}^{(t)} - \bar{\mathbf{M}}^{(t)}\|_F^2 + 4\sigma^2 \\
&\quad + \frac{4(1 - \mu)}{n\eta^2(1 - \beta)} \mathbb{E} \|\mathbf{X}^{(t)} - \bar{\mathbf{X}}^{(t)}\|_F^2 + \frac{2(1 - \mu)}{n(1 - \beta)} \mathbb{E} \|\mathbb{E}[\mathbf{G}^{(t)}] \pm \nabla f(\bar{\mathbf{X}}^{(t)}) - \bar{\mathbf{G}}^{(t)}\|_F^2 \\
&\leq \frac{1}{n} (1 - (1 - \mu)(1 - \beta)) \mathbb{E} \|\mathbf{M}^{(t)} - \bar{\mathbf{M}}^{(t)}\|_F^2 + 4\sigma^2 \\
&\quad + \frac{4(1 - \mu)}{n\eta^2(1 - \beta)} \mathbb{E} \|\mathbf{X}^{(t)} - \bar{\mathbf{X}}^{(t)}\|_F^2 + \frac{8(1 - \mu)\zeta^2}{(1 - \beta)} + \frac{4(1 - \mu)L^2}{n(1 - \beta)} \mathbb{E} \|\mathbf{X}^{(t)} - \bar{\mathbf{X}}^{(t)}\|_F^2 \\
&= \frac{1}{n} (1 - (1 - \mu)(1 - \beta)) \mathbb{E} \|\mathbf{M}^{(t)} - \bar{\mathbf{M}}^{(t)}\|_F^2 + 4\sigma^2 \\
&\quad + \frac{4(1 - \mu)(1 + \eta^2 L^2)}{n\eta^2(1 - \beta)} \mathbb{E} \|\mathbf{X}^{(t)} - \bar{\mathbf{X}}^{(t)}\|_F^2 + \frac{8(1 - \mu)\zeta^2}{(1 - \beta)}
\end{aligned}$$

Multiplying both sides by $\frac{6\eta^2\beta^2}{\rho(1-\mu)(1-\beta)}$ yields

$$\begin{aligned}
\frac{6\eta^2\beta^2}{n\rho(1-\mu)(1-\beta)} \mathbb{E} \|\mathbf{M}^{t+1} - \bar{\mathbf{M}}^{t+1}\|_F^2 &\leq \frac{6\eta^2\beta^2}{n\rho(1-\mu)(1-\beta)} \mathbb{E} \|\mathbf{M}^{(t)} - \bar{\mathbf{M}}^{(t)}\|_F^2 - \frac{6\eta^2\beta^2}{n\rho} \mathbb{E} \|\mathbf{M}^{(t)} - \bar{\mathbf{M}}^{(t)}\|_F^2 \\
&\quad + \frac{48\beta^2(1+L^2\eta^2)}{n\rho(1-\beta)^2} \mathbb{E} \|\mathbf{X}^{(t)} - \bar{\mathbf{X}}^{(t)}\|_F^2 + \frac{24\eta^2\beta^2\sigma^2}{\rho(1-\mu)(1-\beta)} + \frac{48\eta^2\beta^2\zeta^2}{\rho(1-\beta)^2} \\
&\leq \frac{6\eta^2\beta^2}{n\rho(1-\mu)(1-\beta)} \mathbb{E} \|\mathbf{M}^{(t)} - \bar{\mathbf{M}}^{(t)}\|_F^2 - \frac{6\eta^2\beta^2}{n\rho} \mathbb{E} \|\mathbf{M}^{(t)} - \bar{\mathbf{M}}^{(t)}\|_F^2 \\
&\quad + \frac{\rho}{8n} \mathbb{E} \|\mathbf{X}^{(t)} - \bar{\mathbf{X}}^{(t)}\|_F^2 + \frac{\eta^2\rho\zeta^2}{8} + \frac{\eta^2\rho\sigma^2(1-\beta)}{8(1-\mu)}.
\end{aligned}$$

The last step follows from our assumption that the momentum parameter that $\frac{\beta}{1-\beta} \leq \frac{\rho}{21}$ and $\eta \leq \frac{1}{7L}$.

This ensures that $\frac{48\beta^2(1+L^2\eta^2)}{\rho(1-\beta)^2} \leq \frac{49\beta^2}{\rho(1-\beta)^2} \leq \frac{\rho}{8}$. \square

We can now exactly describe the progress in consensus made each round.

Lemma 18.3.5 (One step consensus improvement). *Given Assumption 11, the sequence of iterates generated by (18.4) using step-size $\eta \leq \frac{\rho}{7L}$ and momentum $\frac{\beta}{1-\beta} \leq \frac{\rho}{21}$, satisfy*

$$\begin{aligned}
\frac{1}{n} \mathbb{E} \|\mathbf{X}^{t+1} - \bar{\mathbf{X}}^{t+1}\|_F^2 + \frac{6\eta^2\beta^2}{n\rho(1-\mu)(1-\beta)} \mathbb{E} \|\mathbf{M}^{t+1} - \bar{\mathbf{M}}^{t+1}\|_F^2 \\
\leq \frac{1-\rho/8}{n} \mathbb{E} \|\mathbf{X}^t - \bar{\mathbf{X}}^t\|_F^2 + \frac{6\eta^2\beta^2}{n\rho(1-\mu)(1-\beta)} \mathbb{E} \|\mathbf{M}^t - \bar{\mathbf{M}}^t\|_F^2 + \frac{13\eta^2\zeta^2}{\rho} + \frac{13\eta^2\sigma^2(2-\beta-\mu)}{(1-\mu)\rho}
\end{aligned}$$

Proof. Simply adding the results of Lemmas 18.3.3 and 18.3.4 gives the result. \square

Average and virtual sequences. We will now bound the difference between the average and the virtual sequences $\mathbf{e}^{(t)}$ which recall was defined to be $\mathbf{e}^{(t)} = \hat{\mathbf{x}}^{(t)} - \bar{\mathbf{x}}^{(t)}$. The latter runs SGD with momentum whereas the former only runs SGD. We view the momentum terms as accumulating the gradient terms, delayed over time and hence the proof views SGDM as simply SGD run with a larger step-size.

Lemma 18.3.6 (One step error contraction). *Given Assumption 11, the sequence of iterates generated by (18.4) satisfy*

$$\mathbb{E} \|\mathbf{e}^{(t+1)}\|^2 \leq (1 - (1 - \mu)(1 - \beta)) \mathbb{E} \|\mathbf{e}^{(t)}\|^2 + \frac{2\tilde{\eta}^2 \beta^2}{(1 - \beta)(1 - \mu)} \mathbb{E} \|\mathbb{E}_t[\bar{\mathbf{g}}^t]\|^2 + \tilde{\eta}^2 \beta^2 \sigma^2.$$

Proof. By definition, $\hat{\mathbf{x}}^{(0)} = \bar{\mathbf{x}}^{(0)}$ and hence we have $\mathbf{e}^{(0)} = 0$. For $t \geq 0$, starting from the definition of the error term we have

$$\begin{aligned} \mathbf{e}^{(t+1)} &= \hat{\mathbf{x}}^{(t+1)} - \bar{\mathbf{x}}^{(t+1)} \\ &= \left(\hat{\mathbf{x}}^{(t)} - \frac{\eta}{1 - \beta} \bar{\mathbf{g}}^{(t)} \right) - (\bar{\mathbf{x}}^{(t)} - \eta(\beta \bar{\mathbf{m}}^{(t)} + \bar{\mathbf{g}}^{(t)})) \\ &= \mathbf{e}^{(t)} - \eta\beta \left(\frac{1}{1 - \beta} \bar{\mathbf{g}}^{(t)} - \bar{\mathbf{m}}^{(t)} \right) \\ &= \sum_{k=0}^t -\eta\beta \left(\frac{1}{1 - \beta} \bar{\mathbf{g}}^{(k)} - \bar{\mathbf{m}}^{(k)} \right). \end{aligned}$$

Using the update of the average momentum (18.5), we can write

$$\begin{aligned} \mathbf{e}^{t+1} &= \sum_{k=0}^{(t)} -\eta\beta \left(\frac{1}{1 - \beta} \bar{\mathbf{g}}^{(k)} - \bar{\mathbf{m}}^{(k)} \right) \\ &= \sum_{k=0}^{(t)} -\eta\beta \left(\frac{1}{1 - \beta} \bar{\mathbf{g}}^{(k)} - \left((1 - (1 - \mu)(1 - \beta)) \bar{\mathbf{m}}^{(k-1)} + (1 - \mu) \bar{\mathbf{g}}^{(k-1)} \right) \right) \\ &= (1 - (1 - \mu)(1 - \beta)) \sum_{k=0}^t -\eta\beta \left(\frac{1}{1 - \beta} \bar{\mathbf{g}}^{(k-1)} - \bar{\mathbf{m}}^{(k-1)} \right) + \sum_{k=0}^{(t)} -\frac{\eta\beta}{1 - \beta} (\bar{\mathbf{g}}^{(k)} - \bar{\mathbf{g}}^{(k-1)}) \\ &= (1 - (1 - \mu)(1 - \beta)) \mathbf{e}^{(t)} - \frac{\eta\beta}{1 - \beta} \bar{\mathbf{g}}^{(t)}. \end{aligned}$$

By convention, we assume that vectors with negative indices are 0. Taking norms and expectations gives

$$\begin{aligned} \mathbb{E} \|\mathbf{e}^{(t+1)}\|^2 &= \mathbb{E} \left\| (1 - (1 - \mu)(1 - \beta)) \mathbf{e}^{(t)} - \frac{\eta\beta}{1 - \beta} \bar{\mathbf{g}}^{(t)} \right\|^2 \\ &\leq \mathbb{E} \left\| (1 - (1 - \mu)(1 - \beta)) \mathbf{e}^{(t)} - \frac{\eta\beta}{1 - \beta} \mathbb{E}_t[\bar{\mathbf{g}}^{(t)}] \right\|^2 + \frac{\eta^2 \beta^2 \sigma^2}{(1 - \beta)^2} \\ &\leq (1 - (1 - \mu)(1 - \beta)) \mathbb{E} \|\mathbf{e}^{(t)}\|^2 + \frac{2\eta^2 \beta^2}{(1 - \beta)^3 (1 - \mu)} \mathbb{E} \|\mathbb{E}_t[\bar{\mathbf{g}}^{(t)}]\|^2 + \frac{\eta^2 \beta^2 \sigma^2}{(1 - \beta)^2}. \quad \square \end{aligned}$$

Convergence rate for non-convex case.

Theorem 18.3.7. *Given Assumption 11, the sequence of iterates generated by (18.4) for step size $\eta = \min\left(\frac{\rho}{7L}, \frac{1-\beta}{4L}, \frac{(1-\mu)(1-\beta)^2}{4\beta L}, \sqrt{\frac{4n(f(\bar{\mathbf{x}}^0) - f^*)}{L\sigma^2 T}}\right)$ and momentum parameter $\frac{\beta}{1-\beta} \leq \frac{\rho}{21}$ satisfies*

$$\begin{aligned} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E} \|\nabla f(\bar{\mathbf{x}}^t)\| &\leq \mathcal{O} \left(\sqrt{\frac{L\sigma^2(f(\bar{\mathbf{x}}^0) - f^*)}{nT}} + \right. \\ &\quad \left. \sqrt[3]{L^2(f(\bar{\mathbf{x}}^0) - f^*)^2 \frac{\tilde{\zeta}^2}{\rho^2 T^2}} + \right. \\ &\quad \left. \left(\frac{1}{\rho} + \frac{1}{1-\beta} + \frac{\beta}{(1-\mu)(1-\beta)^2} \right) \frac{L(f(\bar{\mathbf{x}}^0) - f^*)}{T} \right). \end{aligned}$$

Proof. Define $\tilde{\zeta}^2 := \zeta^2 + \sigma^2 \left(1 + \frac{1-\beta}{1-\mu}\right)$. Scaling Lemma 18.3.5 by $\frac{24L^2\tilde{\eta}}{\rho}$ gives

$$\begin{aligned} \frac{24L^2\tilde{\eta}}{\rho n} \mathbb{E} \|\mathbf{X}^{t+1} - \bar{\mathbf{X}}^{t+1}\|_F^2 &+ \frac{144L^2\tilde{\eta}^3\beta^2(1-\beta)}{n\rho^2(1-\mu)} \mathbb{E} \|\mathbf{M}^{t+1} - \bar{\mathbf{M}}^{t+1}\|_F^2 \\ &\leq \frac{24L^2\tilde{\eta}}{\rho n} \mathbb{E} \|\mathbf{X}^t - \bar{\mathbf{X}}^t\|_F^2 + \frac{144L^2\tilde{\eta}^3\beta^2(1-\beta)}{n\rho^2(1-\mu)} \mathbb{E} \|\mathbf{M}^t - \bar{\mathbf{M}}^t\|_F^2 \\ &\quad - \frac{3L^2\tilde{\eta}}{n} \mathbb{E} \|\mathbf{X}^t - \bar{\mathbf{X}}^t\|_F^2 + \frac{312L^2\tilde{\eta}^3(1-\beta)^2\tilde{\zeta}^2}{\rho^2}. \end{aligned}$$

Scaling Lemma 18.3.6 by $\frac{3L^2\tilde{\eta}}{2(1-\mu)(1-\beta)}$ gives

$$\begin{aligned} \frac{3L^2\tilde{\eta}}{2(1-\mu)(1-\beta)} \mathbb{E} \|\mathbf{e}^{(t+1)}\|^2 &\leq \frac{3L^2\tilde{\eta}}{2(1-\mu)(1-\beta)} \mathbb{E} \|\mathbf{e}^{(t)}\|^2 - \frac{3L^2\tilde{\eta}}{2} \mathbb{E} \|\mathbf{e}^{(t)}\|^2 \\ &\quad + \frac{3L^2\tilde{\eta}^3\beta^2}{(1-\beta)^2(1-\mu)^2} \mathbb{E} \|\mathbb{E}_t[\bar{\mathbf{g}}^t]\|^2 + \frac{3L^2\tilde{\eta}^3\beta^2\sigma^2}{2(1-\mu)(1-\beta)}. \end{aligned}$$

Finally Lemma 18.3.1 gives

$$\mathbb{E} f(\hat{\mathbf{x}}^{(t+1)}) \leq \mathbb{E} f(\hat{\mathbf{x}}^{(t)}) - \frac{\tilde{\eta}}{4} \|\nabla f(\bar{\mathbf{x}}^{(t)})\|^2 - \frac{\tilde{\eta}}{4} \mathbb{E} \left\| \frac{1}{n} \sum_{i=1}^n \nabla f_i(\mathbf{x}_i^{(t)}) \right\|^2 + \frac{L\tilde{\eta}^2\sigma^2}{n} + \frac{3L^2\tilde{\eta}}{2} \mathbb{E} \|\mathbf{e}^{(t)}\|^2 + \frac{3L^2\tilde{\eta}}{n} \|\mathbf{X}^t - \bar{\mathbf{X}}^t\|_F^2.$$

Define

$$\Phi^t := \frac{24L^2\tilde{\eta}}{\rho n} \mathbb{E} \|\mathbf{X}^t - \bar{\mathbf{X}}^t\|_F^2 + \frac{144L^2\tilde{\eta}^3\beta^2(1-\beta)}{n\rho^2(1-\mu)} \mathbb{E} \|\mathbf{M}^t - \bar{\mathbf{M}}^t\|_F^2 + \frac{3L^2\tilde{\eta}}{2(1-\mu)(1-\beta)} \mathbb{E} \|\mathbf{e}^{(t)}\|^2 + \mathbb{E}[f(\bar{\mathbf{x}}^t) - f^*].$$

Note that $\Phi^0 = \mathbb{E}[f(\bar{\mathbf{x}}^0)] - f^*$ and that $\Phi^t \geq 0$ for any t . Then adding the three inequalities from

the lemmas as described above gives

$$\begin{aligned} \Phi^{t+1} \leq \Phi^t - \frac{\tilde{\eta}}{4} \|\nabla f(\bar{\mathbf{x}}^{(t)})\|^2 &+ \left(\frac{3L^2\tilde{\eta}^3\beta^2}{(1-\beta)^2(1-\mu)^2} - \frac{\tilde{\eta}}{4} \right) \mathbb{E} \left\| \frac{1}{n} \sum_{i=1}^n \nabla f_i(\mathbf{x}_i^{(t)}) \right\|^2 \\ &+ \frac{L\tilde{\eta}^2\sigma^2}{n} + \frac{3L^2\tilde{\eta}^3\beta^2\sigma^2}{2(1-\mu)(1-\beta)} + \frac{312L^2\tilde{\eta}^3(1-\beta)^2\tilde{\zeta}^2}{\rho^2}. \end{aligned}$$

Since, $\eta \leq \frac{(1-\mu)(1-\beta)^2}{4\beta L}$, we have that $\frac{3L^2\tilde{\eta}^3\beta^2}{(1-\beta)^2(1-\mu)^2} \leq \frac{\tilde{\eta}}{4}$. Rearranging the terms and averaging over t , we get

$$\begin{aligned} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E} \|\nabla f(\bar{\mathbf{x}}^{(t)})\|^2 &\leq \frac{4}{\tilde{\eta}T} (\Phi^0 - \Phi^T) + \frac{L\tilde{\eta}\sigma^2}{n} + \frac{6L^2\tilde{\eta}^2\beta^2\sigma^2}{(1-\mu)} + \frac{1248L^2\tilde{\eta}^2(1-\beta)^2\tilde{\zeta}^2}{\rho^2} \\ &\leq \frac{1}{\tilde{\eta}T} 4(f(\bar{\mathbf{x}}^0) - f^*) + \tilde{\eta} \left(\frac{L\sigma^2}{n} \right) + \tilde{\eta}^2 \left(\frac{L^2\rho^2\sigma^2(1-\beta)}{(1-\mu)} + \frac{1248L^2(1-\beta)^2\tilde{\zeta}^2}{\rho^2} \right). \\ &\leq \frac{1}{\tilde{\eta}T} 4(f(\bar{\mathbf{x}}^0) - f^*) + \tilde{\eta} \left(\frac{L\sigma^2}{n} \right) + \tilde{\eta}^2 \left(\frac{L^2\sigma^2(1-\beta)}{(1-\mu)} + \frac{1248L^2\tilde{\zeta}^2}{\rho^2} \right) \\ &\leq \frac{1}{\tilde{\eta}T} 4(f(\bar{\mathbf{x}}^0) - f^*) + \tilde{\eta} \left(\frac{L\sigma^2}{n} \right) + \tilde{\eta}^2 \left(\frac{1249L^2\tilde{\zeta}^2}{\rho^2} \right). \end{aligned}$$

Choosing an appropriate steps-size η proves the theorem. \square

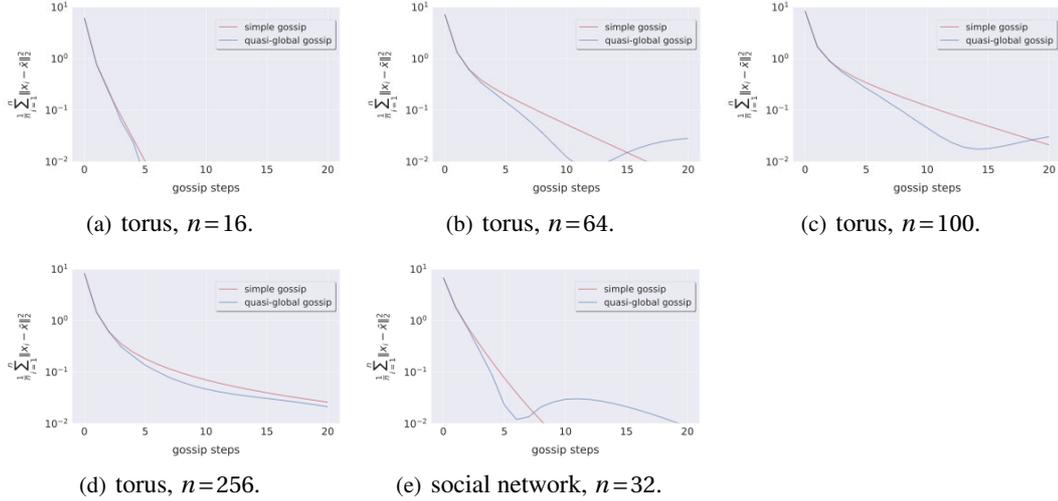


Figure 18.4 – More results on understanding QG-DSGDm through the aspect of distributed consensus averaging problem on various communication topologies and scales. QG-DSGDm without gradient update step (as in (9.4)) still presents faster convergence (to a relative high precision) than the normal gossip algorithm.

18.4 Additional Results

18.4.1 Results on Distributed Average Consensus Problem

Figure 18.4 illustrates the results for average consensus problem on other communication topologies and topology scales.

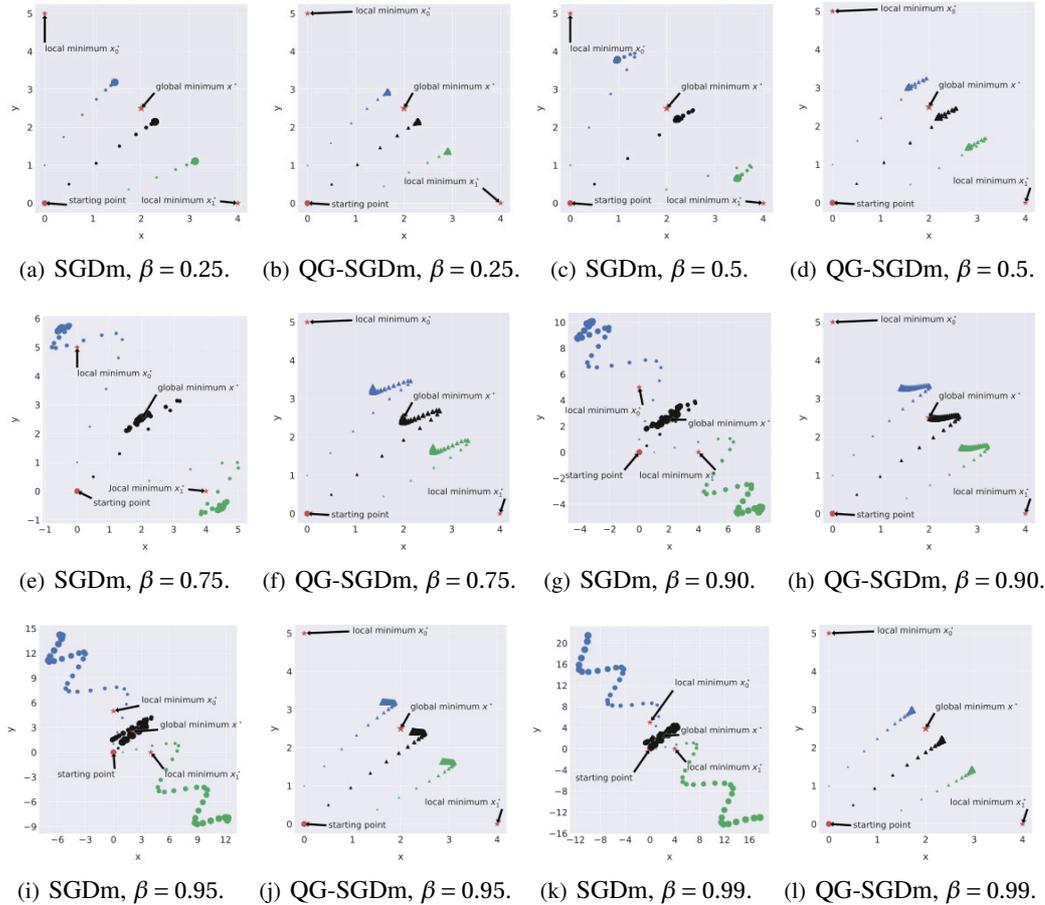


Figure 18.5 – **The ineffectiveness of local momentum acceleration under heterogeneous data setup:** the local momentum buffer accumulates “biased” gradients, causing unstable and oscillation behaviors. The gradient is estimated by the direction from a given model to the local minimum with a constant update magnitude. The size of marker will increase by the number of update steps; colors blue and green indicate the local models of two workers (after performing local update), while color black is the synchronized global model. Uniform weight averaging is performed after each update step, and the new gradients will be computed on the averaged model.

18.4.2 Results on 2D Illustration

Following the 2D illustration in Figure 9.2, we elaborate below the various choices of momentum factor for SGDm and QG-SGDm in Figure 18.5. We can witness that the effectiveness of local

momentum (oscillation) in SGDM is always impacted by the data heterogeneity, no matter the choices of momentum factor. While for QG-SGDM, there exists a trade-off between stabilized optimization and fast convergence, controlled by the momentum factor β . Note that we always set $\mu := \beta$ in QG-SGDM, which may result in undesirable behavior.

18.4.3 The Learning Curves on CV tasks

Figure 18.6 visualizes the learning curves for training ResNet-EvoNorm-20 on CIFAR-10, in terms of various degrees of non-i.i.d.-ness and network topologies (Ring and Social topology).

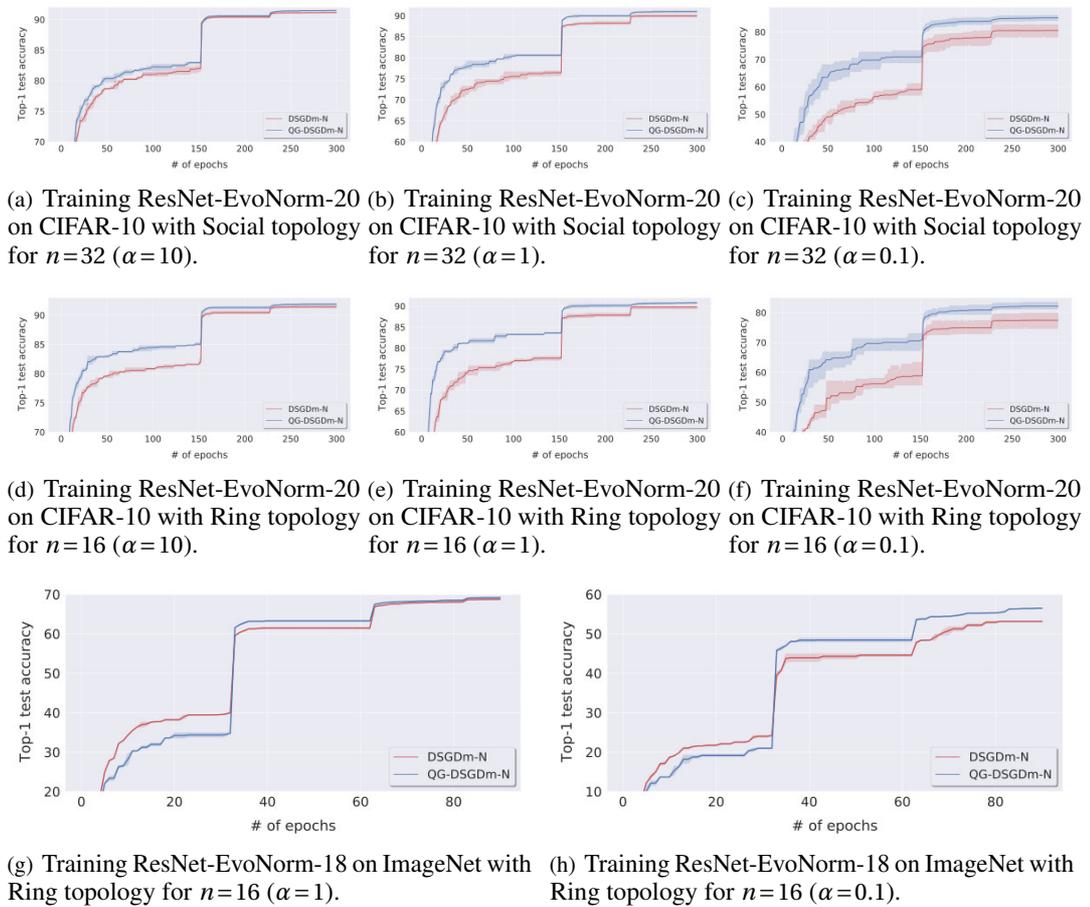


Figure 18.6 – Learning curves for cv tasks.

18.4.4 The Ineffectiveness of Tuning Momentum Factor for DSGM-N

Figure 18.7 shows that tuning momentum factor for DSGM-N cannot alleviate the training difficulty caused by heterogeneity.

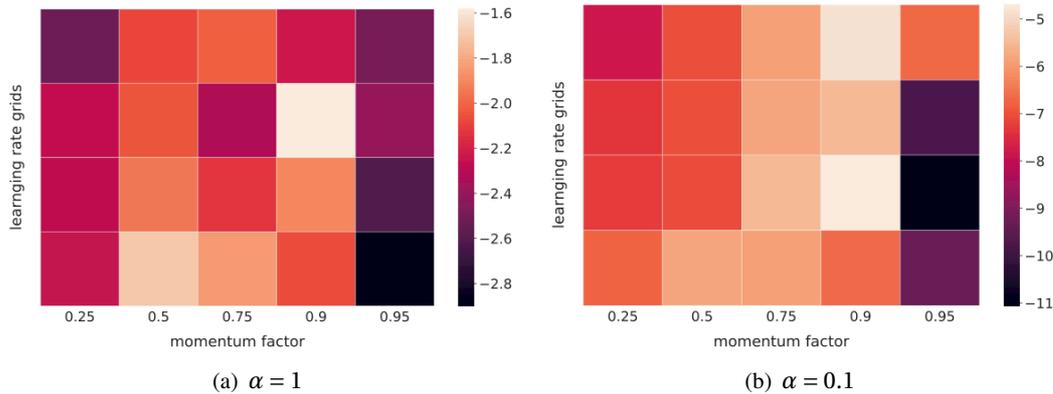


Figure 18.7 – The ineffectiveness of tuning momentum factors for DSGDm-N, for training ResNet-EvoNorm-20 on CIFAR-10. We illustrate the performance gap between DSGDm-N (various combination of learning rate and momentum factor) and QG-DSGDm-N (tuned learning rate from the grid with default momentum factor 0.9).

Table 18.1 – **The test top-1 accuracy of decentralized algorithms evaluated on various topology scales and non-i.i.d.-ness**, for training ResNet-EvoNorm-20 on CIFAR-10. The results are over three random seeds, with sufficient learning rate tuning. The table corresponds to Figure 9.6 in the main paper.

Methods	Ring ($n=16$)		Ring ($n=32$)		Ring ($n=48$)	
	$\alpha = 1$	$\alpha = 0.1$	$\alpha = 1$	$\alpha = 0.1$	$\alpha = 1$	$\alpha = 0.1$
SGDm-N (centralized)	92.18 ± 0.19		91.92 ± 0.33		91.63 ± 0.25	
DSGDm-N	89.98 ± 0.10	77.48 ± 2.67	88.46 ± 0.29	78.17 ± 1.63	85.54 ± 0.33	73.67 ± 0.90
QG-DSGDm-N	91.28 ± 0.38	82.20 ± 1.27	90.27 ± 0.07	83.18 ± 1.11	89.75 ± 0.32	80.28 ± 1.52

18.4.5 The Superior Performance of QG-DSGDm-N Generalize to Various Topology Scales

Table 18.1 further showcases the generality of the predominant performance gain of quasi-global momentum on various topology scales (n).

18.4.6 Multiple-step QG-DSGDm-N variant

Table 18.2 illustrates the performance for the multiple-step variant of QG-DSGDm-N. We can witness that tuning the value of τ cannot lead to a significant performance gain.

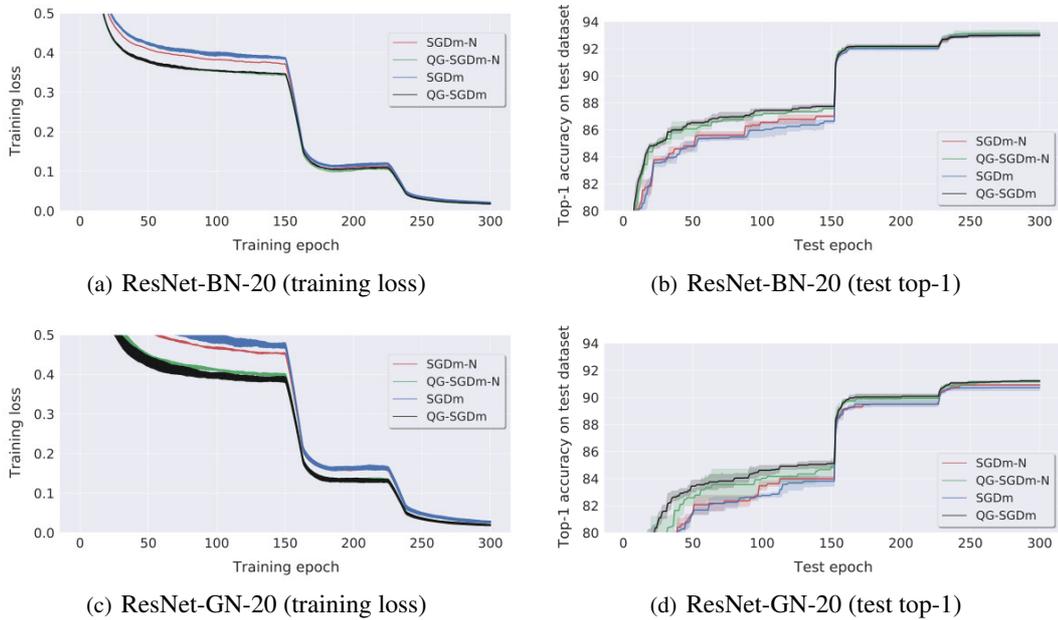
18.4.7 Understanding QG-DSGDm and QG-DSGDm-N on the Single Worker Case

Recall that the single worker case of QG-DSGDm and QG-DSGDm-N refers to QG-SGDm and QG-SGDm-N.

Figure 18.8 studies the learning behavior (learning curves for both training loss and top-1 test accuracy, as well as final best test accuracy) of QG-SGDm and QG-SGDm-N on two

Table 18.2 – Ablation study for the variant of multiple-step QG-DSGDm-N (illustrated in Algorithm 20), for training ResNet-EvoNorm-20 on CIFAR-10. The results are averaged over three seeds with tuned learning rate.

Methods	Ring ($n=16$)	
	$\alpha = 1$	$\alpha = 0.1$
SGDm-N (centralized)	92.18 \pm 0.19	
DSGD	88.88 \pm 0.26	74.55 \pm 2.07
DSGDm-N	89.98 \pm 0.10	77.48 \pm 2.67
QG-DSGDm-N ($\tau = 1$)	91.28 \pm 0.38	82.20 \pm 1.27
QG-DSGDm-N ($\tau = 2$)	91.11 \pm 0.18	82.25 \pm 1.68
QG-DSGDm-N ($\tau = 3$)	91.04 \pm 0.01	81.57 \pm 2.21
QG-DSGDm-N ($\tau = 4$)	91.26 \pm 0.25	82.55 \pm 1.55

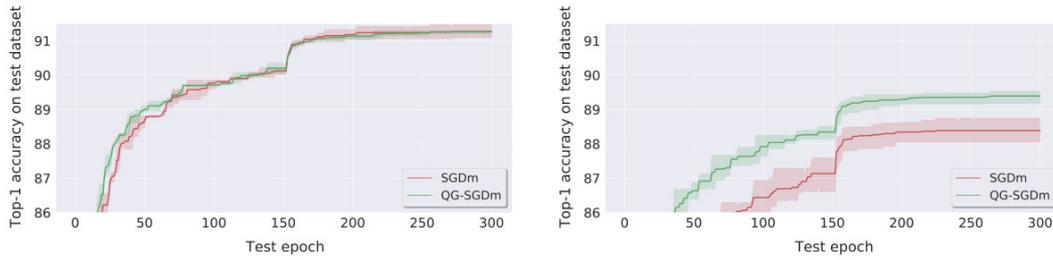


	SGDm-N	QG-SGDm-N	SGDm	QG-SGDm
BN	93.01 \pm 0.12	93.17 \pm 0.30	92.90 \pm 0.15	92.95 \pm 0.13
GN	90.92 \pm 0.08	91.15 \pm 0.06	90.61 \pm 0.15	91.23 \pm 0.01

Figure 18.8 – Understanding the learning behavior of QG-SGDm and QG-SGDm-N, for training ResNet-20 on CIFAR-10 with mini-batch size of 32.

different normalization methods (BN and GN) for ResNet-20 on CIFAR-10. In general Nesterov momentum variants outperforms that of HeavyBall momentum, and we can witness a larger performance gain when the optimization is challenging (e.g. in the of using GN replacement).

Figure 18.9 further investigates the impact of weight decay on Nesterov momentum variants. Excluding weight decay from the training procedure is detrimental to the final generalization performance. We also notice larger benefits of QG-SGDm-N when the optimization procedure is fragile/unstable.



(a) ResNet-BN-20.

(b) ResNet-GN-20.

	w/ weight decay		w/o weight decay	
	SGDm-N	QG-SGDm-N	SGDm-N	QG-SGDm-N
BN	93.01 ± 0.12	93.17 ± 0.30	91.26 ± 0.19	91.28 ± 0.03
GN	90.92 ± 0.08	91.15 ± 0.06	88.39 ± 0.35	89.39 ± 0.20

Figure 18.9 – Understanding the impact of weight decay on QG-SGDm-N, for training ResNet-20 on CIFAR-10 with mini-batch size of 32.

Figure 18.10 in addition illustrates the curves of weight norm and effective step-size during the optimization procedure, to interpret the potential causes of the performance gain.

18.4.8 Understanding QG-DSGDm on the Single Worker Case via Toy Function

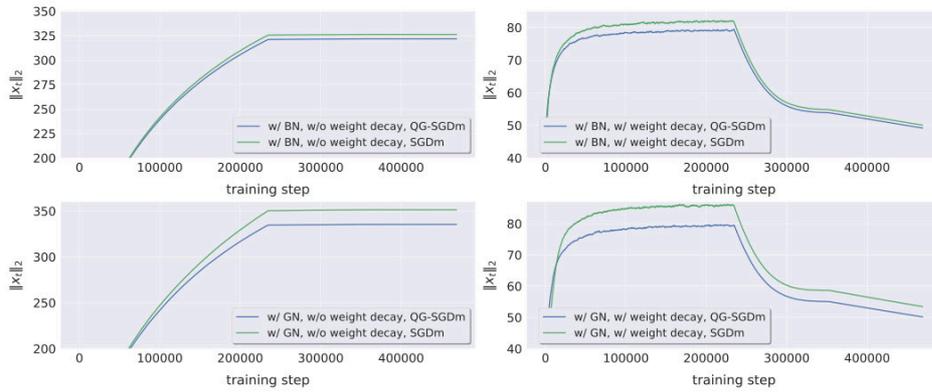
Similar to Lucas et al. (2019), we first optimize the Rosenbrock function, defined as $f(x, y) = (y - x^2)^2 + 100(x - 1)^2$.

Figure 9.4 illustrates the stabilized optimization trajectory in QG-SGDm (much less oscillation than SGDm).

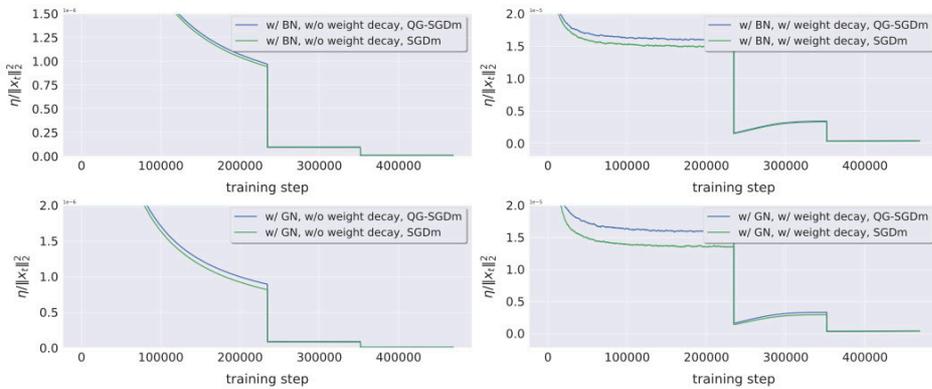
We further study a simple non-convex toy problem Lucas et al. (2019):

$$f(x, y) = \log(e^x + e^{-x}) + 10 \log \left(e^{e^x(y - \sin(ax))} + e^{-e^x(y - \sin(ax))} \right),$$

in Figure 18.11. In our experiments, we choose $a = 8$ and $b = 10$, and initialize the optimizer at $(x, y) = (-2, 0)$.



(a) Weight norm $\|\mathbf{x}_t\|_2$.



(b) Effective stepsize $\frac{\eta}{\|\mathbf{x}_t\|_2}$.

Figure 18.10 – Understanding QG-SGDm through the lens of the effective step-size, for training ResNet-20 on CIFAR-10 with mini-batch size of 32.

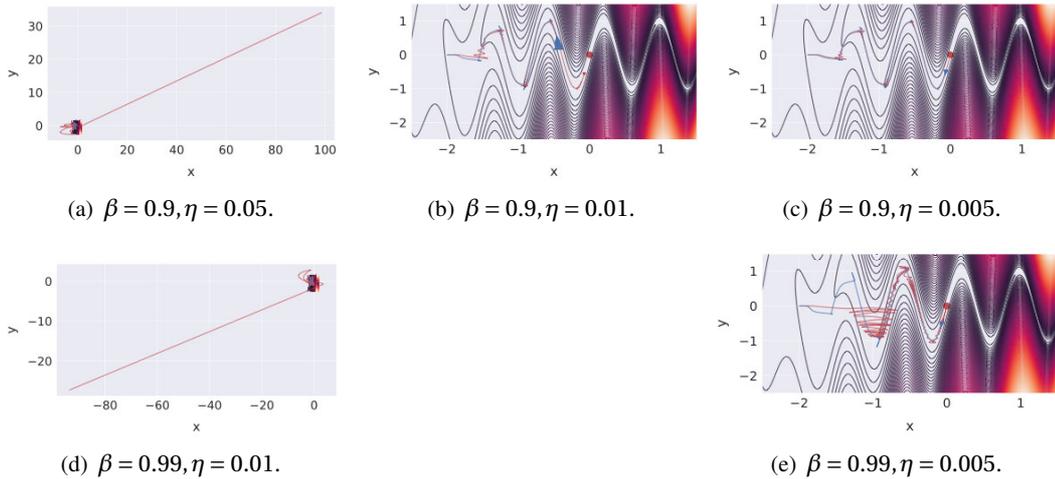
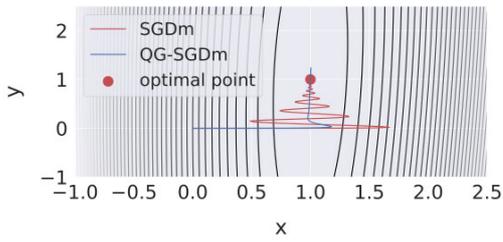
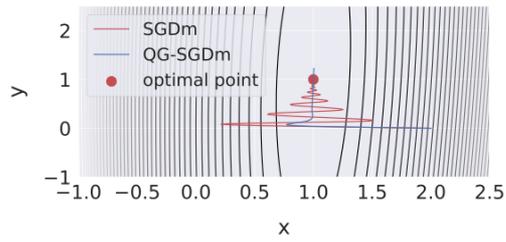


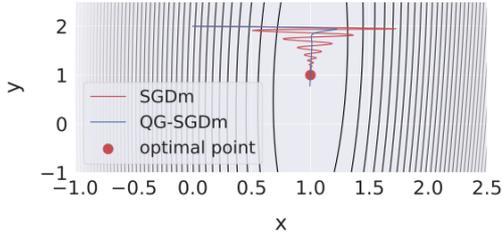
Figure 18.11 – Understanding the optimization trajectory of QG-SGDm and Heavy-ball momentum SGD (SGDm), via a 2D toy function $f(x, y) = \log(e^x + e^{-x}) + 10\log\left(e^{e^x(y - \sin(8x))} + e^{-e^x(y - \sin(8x))}\right)$. This function has an optimal value at $(x, y) = (0, 0)$. Red line corresponds to SGDm and blue line indicates QG-SGDm. Red line illustrates larger oscillation than QG-SGDm on the optimization trajectory.



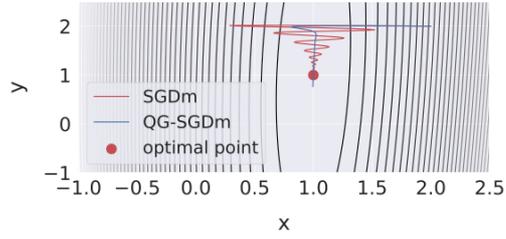
(a) $\beta = 0.9, \eta = 0.001$, initial point $(0, 0)$.



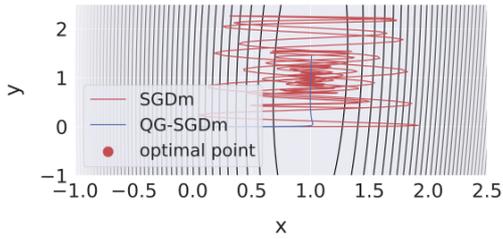
(b) $\beta = 0.9, \eta = 0.001$, initial point $(2, 0)$.



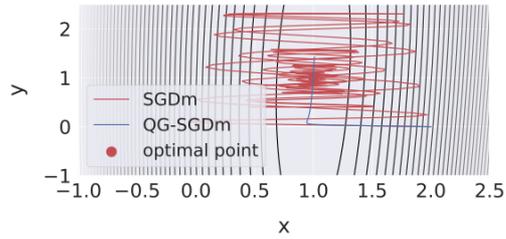
(c) $\beta = 0.9, \eta = 0.001$, initial point $(0, 2)$.



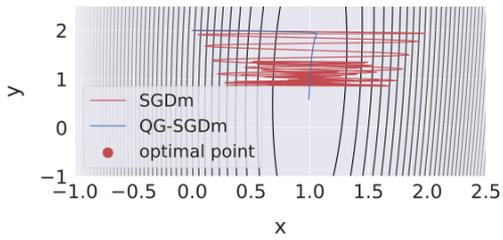
(d) $\beta = 0.9, \eta = 0.001$, initial point $(2, 2)$.



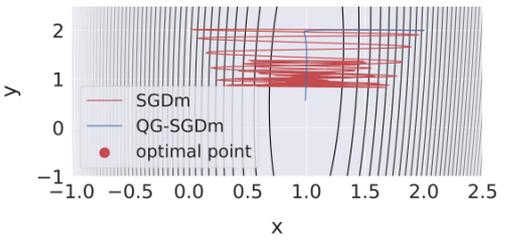
(e) $\beta = 0.99, \eta = 0.001$, initial point $(0, 0)$.



(f) $\beta = 0.99, \eta = 0.001$, initial point $(2, 0)$.



(g) $\beta = 0.99, \eta = 0.001$, initial point $(0, 2)$.



(h) $\beta = 0.99, \eta = 0.001$, initial point $(2, 2)$.

Figure 18.12 – Understanding the optimization trajectory of QG-SGDm and Heavy-ball momentum SGD (SGDm), via a 2D toy function $f(x, y) = (y - x^2)^2 + 100(x - 1)^2$. This function has a global minimum at $(x, y) = (1, 1)$. Red line corresponds to SGDm and blue line indicates QG-SGDm. Red line illustrates larger oscillation than QG-SGDm on the optimization trajectory.

Bibliography

- (2020a). The gpt-3 economy. <https://bdtechtalks.com/2020/09/21/gpt-3-economy-business-model/>.
- (2020b). Openai presents gpt-3, a 175 billion parameters language model. <https://developer.nvidia.com/blog/openai-presents-gpt-3-a-175-billion-parameters-language-model/>.
- (2021). Worldwide internet of things infrastructure forecast, 2021-2025. <https://www.idc.com/research/viewtoc.jsp?containerId=US48097621>.
- Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.
- Achille, A., Rovere, M., and Soatto, S. (2019). Critical learning periods in deep networks. In *International Conference on Learning Representations (ICLR)*.
- Ahn, S., Hu, S. X., Damianou, A., Lawrence, N. D., and Dai, Z. (2019). Variational information distillation for knowledge transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9163–9171.
- Alistarh, D., De Sa, C., and Konstantinov, N. (2018). The convergence of stochastic gradient descent in asynchronous shared memory. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, pages 169–178.
- Alistarh, D., Grubic, D., Li, J., Tomioka, R., and Vojnovic, M. (2017). Qsgd: Communication-efficient sgd via gradient quantization and encoding. In *Advances in Neural Information Processing Systems*, volume 30.
- Alvarez, J. M. and Salzmann, M. (2016). Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pages 2270–2278.
- Alvarez, J. M. and Salzmann, M. (2017). Compression-aware training of deep networks. In *Advances in Neural Information Processing Systems*, pages 856–867.
- Andreux, M., du Terrail, J. O., Beguier, C., and Tramel, E. W. (2020). Siloed federated learning for multi-centric histopathology datasets. In *Domain Adaptation and Representation Transfer, and Distributed and Collaborative Learning*, pages 129–139. Springer.

Bibliography

- Anil, R., Pereyra, G., Passos, A., Ormandi, R., Dahl, G. E., and Hinton, G. E. (2018). Large scale distributed neural network training through online distillation. *arXiv preprint arXiv:1804.03235*.
- Arjevani, Y., Carmon, Y., Duchi, J. C., Foster, D. J., Srebro, N., and Woodworth, B. (2019). Lower bounds for non-convex stochastic optimization. *arXiv preprint arXiv:1912.02365*.
- Assran, M., Loizou, N., Ballas, N., and Rabbat, M. (2019). Stochastic gradient push for distributed deep learning. In *International Conference on Machine Learning*, pages 344–353. PMLR.
- Balu, A., Jiang, Z., Tan, S. Y., Hedge, C., Lee, Y. M., and Sarkar, S. (2021). Decentralized deep learning using momentum-accelerated consensus. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3675–3679. IEEE.
- Bellec, G., Kappel, D., Maass, W., and Legenstein, R. (2018). Deep rewiring: Training very sparse deep networks. In *ICLR - International Conference on Learning Representations*.
- Bellet, A., Guerraoui, R., Taziki, M., and Tommasi, M. (2018). Personalized and private peer-to-peer machine learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 84, pages 473–481.
- Ben-David, S., Blitzer, J., Crammer, K., Kulesza, A., Pereira, F., and Vaughan, J. W. (2010). A theory of learning from different domains. *Machine learning*, 79(1-2):151–175.
- Bengio, Y., Léonard, N., and Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.
- Berahas, A. S., Iakovidou, C., and Wei, E. (2019). Nested distributed gradient methods with adaptive quantized communication. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 1519–1525. IEEE.
- Bernstein, J., Wang, Y.-X., Azizzadenesheli, K., and Anandkumar, A. (2018). signsgd: Compressed optimisation for non-convex problems. In *International Conference on Machine Learning*, pages 560–569. PMLR.
- Bijral, A. S., Sarwate, A. D., and Srebro, N. (2016). On data dependence in distributed stochastic optimization. *arXiv preprint arXiv:1603.04379*.
- Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konečný, J., Mazzocchi, S., McMahan, B., et al. (2019). Towards federated learning at scale: System design. *Proceedings of Machine Learning and Systems*, 1:374–388.
- Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer.
- Bottou, L. and Bousquet, O. (2007). The tradeoffs of large scale learning. *Advances in neural information processing systems*, 20.

- Bottou, L., Curtis, F. E., and Nocedal, J. (2018). Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311.
- Boyd, S., Ghosh, A., Prabhakar, B., and Shah, D. (2006). Randomized gossip algorithms. *IEEE transactions on information theory*, 52(6):2508–2530.
- Brand, A., Allen, L., Altman, M., Hlava, M., and Scott, J. (2015). Beyond authorship: attribution, contribution, collaboration, and credit. *Learned Publishing*, 28(2):151–155.
- Brock, A., Donahue, J., and Simonyan, K. (2019). Large scale GAN training for high fidelity natural image synthesis. In *International Conference on Learning Representations*.
- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. In *NeurIPS 2020 - Advances in Neural Information Processing Systems*.
- Buciluă, C., Caruana, R., and Niculescu-Mizil, A. (2006). Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541.
- Caldas, S., Wu, P., Li, T., Konečný, J., McMahan, H. B., Smith, V., and Talwalkar, A. (2018). Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*.
- Carli, R., Bullo, F., and Zampieri, S. (2010a). Quantized average consensus via dynamic coding/decoding schemes. *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal*, 20(2):156–175.
- Carli, R., Fagnani, F., Frasca, P., Taylor, T., and Zampieri, S. (2007). Average consensus on networks with transmission noise or quantization. In *2007 European Control Conference (ECC)*, pages 1852–1857. IEEE.
- Carli, R., Fagnani, F., Frasca, P., and Zampieri, S. (2010b). Gossip consensus algorithms via quantized communication. *Automatica*, 46(1):70–80.
- Carreira-Perpinán, M. A. and Idelbayev, Y. (2018). “learning-compression” algorithms for neural net pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8532–8541.
- Chang, H., Shejwalkar, V., Shokri, R., and Houmansadr, A. (2019). Cronus: Robust and heterogeneous collaborative learning with black-box knowledge transfer. *arXiv preprint arXiv:1912.11279*.
- Chaudhari, P., Choromanska, A., Soatto, S., LeCun, Y., Baldassi, C., Borgs, C., Chayes, J., Sagun, L., and Zecchina, R. (2017). Entropy-sgd: Biasing gradient descent into wide valleys. In *International Conference on Learning Representations*.

Bibliography

- Chaudhari, P. and Soatto, S. (2018). Stochastic gradient descent performs variational inference, converges to limit cycles for deep networks. In *2018 Information Theory and Applications Workshop (ITA)*, pages 1–10. IEEE.
- Chavdarova, T., Gidel, G., Fleuret, F., and Lacoste-Julien, S. (2019). Reducing noise in gan training with variance reduced extragradient. *Advances in Neural Information Processing Systems*, 32.
- Chen, H.-Y. and Chao, W.-L. (2021). Fed{be}: Making bayesian model ensemble applicable to federated learning. In *International Conference on Learning Representations*.
- Chen, K. and Huo, Q. (2016). Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 5880–5884. IEEE.
- Chen, T., Frankle, J., Chang, S., Liu, S., Zhang, Y., Wang, Z., and Carbin, M. (2020). The lottery ticket hypothesis for pre-trained bert networks. *Advances in neural information processing systems*, 33:15834–15846.
- Chrabaszcz, P., Loshchilov, I., and Hutter, F. (2017). A downsampled variant of ImageNet as an alternative to the CIFAR datasets. *arXiv preprint arXiv:1707.08819*.
- Collins, M. (2002). Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 conference on empirical methods in natural language processing (EMNLP 2002)*, pages 1–8.
- Conneau, A. and Lample, G. (2019). Cross-lingual language model pretraining. In *Advances in Neural Information Processing Systems*, pages 7059–7069.
- Courbariaux, M., Bengio, Y., and David, J.-P. (2015). Binaryconnect: Training deep neural networks with binary weights during propagations. In *NeurIPS - Advances in Neural Information Processing Systems*, pages 3123–3131.
- Dagan, I., Glickman, O., and Magnini, B. (2005). The pascal recognising textual entailment challenge. In *Machine Learning Challenges Workshop*, pages 177–190. Springer.
- Dai, A. M. and Le, Q. V. (2015). Semi-supervised sequence learning. In *Advances in neural information processing systems*, volume 28.
- Dai, X., Yin, H., and Jha, N. (2019). Nest: A neural network synthesis tool based on a grow-and-prune paradigm. *IEEE Transactions on Computers*.
- Das, R., Acharya, A., Hashemi, A., Sanghavi, S., Dhillon, I. S., and Topcu, U. (2020). Faster non-convex federated learning via global and local momentum. *arXiv preprint arXiv:2012.04061*.
- Daskalakis, C., Ilyas, A., Syrgkanis, V., and Zeng, H. (2018). Training GANs with optimism. In *International Conference on Learning Representations*.

- Davis, A., Gardner, B. B., and Gardner, M. R. (1941). Deep South. *University of Chicago Press, Chicago, IL*.
- Defazio, A. and Bottou, L. (2019). On the ineffectiveness of variance reduced optimization for deep learning. In *Advances in Neural Information Processing Systems*.
- Dekel, O., Gilad-Bachrach, R., Shamir, O., and Xiao, L. (2012). Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, 13(1).
- Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.
- Deng, Y., Kamani, M. M., and Mahdavi, M. (2020). Adaptive personalized federated learning. *arXiv preprint arXiv:2003.13461*.
- Dettmers, T. and Zettlemoyer, L. (2019). Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*.
- Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *ACL - Association for Computational Linguistics*.
- Diakonikolas, J. and Orecchia, L. (2017). Accelerated extra-gradient descent: A novel accelerated first-order method. *arXiv preprint arXiv:1706.04680*.
- Diao, E., Ding, J., and Tarokh, V. (2021). HeteroFL: Computation and communication efficient federated learning for heterogeneous clients. In *International Conference on Learning Representations*.
- Dieuleveut, A. and Patel, K. K. (2019). Communication trade-offs for local-sgd with large step size. *Advances in Neural Information Processing Systems*, 32.
- Dimakis, A. G., Kar, S., Moura, J. M., Rabbat, M. G., and Scaglione, A. (2010). Gossip algorithms for distributed signal processing. *Proceedings of the IEEE*, 98(11):1847–1864.
- Dinh, L., Pascanu, R., Bengio, S., and Bengio, Y. (2017). Sharp minima can generalize for deep nets. In *ICML - Proceedings of the 34th International Conference on Machine Learning*, pages 1019–1028.
- Doan, T. T., Maguluri, S. T., and Romberg, J. (2018). Accelerating the convergence rates of distributed subgradient methods with adaptive quantization. *arXiv preprint arXiv:1810.13245*.
- Dodge, J., Ilharco, G., Schwartz, R., Farhadi, A., Hajishirzi, H., and Smith, N. (2020). Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *arXiv preprint arXiv:2002.06305*.

Bibliography

- Dolan, W. B. and Brockett, C. (2005). Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.
- Draxler, F., Veschgini, K., Salmhofer, M., and Hamprecht, F. (2018). Essentially no barriers in neural network energy landscape. In *International conference on machine learning*, pages 1309–1318. PMLR.
- Drumond, M., Lin, T., Jaggi, M., and Falsafi, B. (2018). Training dnns with hybrid block floating point. In *Advances in Neural Information Processing Systems*.
- Duchi, J. C., Agarwal, A., and Wainwright, M. J. (2011). Dual averaging for distributed optimization: Convergence analysis and network scaling. *IEEE Transactions on Automatic control*, 57(3):592–606.
- Duchi, J. C., Bartlett, P. L., and Wainwright, M. J. (2012). Randomized smoothing for stochastic optimization. *SIAM Journal on Optimization*, 22(2):674–701.
- Dvornik, N., Schmid, C., and Mairal, J. (2019). Diversity with cooperation: Ensemble methods for few-shot classification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3723–3731.
- Elliott, D., Frank, S., Sima'an, K., and Specia, L. (2016). Multi30k: Multilingual english-german image descriptions. *arXiv preprint arXiv:1605.00459*.
- Foret, P., Kleiner, A., Mobahi, H., and Neyshabur, B. (2021). Sharpness-aware minimization for efficiently improving generalization. In *International Conference on Learning Representations*.
- Fort, S. and Ganguli, S. (2019). Emergent properties of the local geometry of neural loss landscapes. *arXiv preprint arXiv:1910.05929*.
- Frankle, J. and Carbin, M. (2019). The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR - International Conference on Learning Representations*.
- Frankle, J., Dziugaite, G. K., Roy, D. M., and Carbin, M. (2019). Stabilizing the lottery ticket hypothesis. *arXiv preprint arXiv:1903.01611*.
- Frankle, J., Schwab, D. J., and Morcos, A. S. (2020). The early phase of neural network training. In *International Conference on Learning Representations (ICLR)*.
- Furlanello, T., Lipton, Z., Tschannen, M., Itti, L., and Anandkumar, A. (2018). Born again neural networks. In *International Conference on Machine Learning*, pages 1607–1616. PMLR.
- Gaier, A. and Ha, D. (2019). Weight agnostic neural networks. In *Advances in Neural Information Processing Systems*, pages 5365–5379.
- Gal, Y., Hron, J., and Kendall, A. (2017). Concrete dropout. In *NeurIPS - Advances in Neural Information Processing Systems*, pages 3581–3590.

- Gale, T., Elsen, E., and Hooker, S. (2019). The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*.
- Garipov, T., Izmailov, P., Podoprikin, D., Vetrov, D. P., and Wilson, A. G. (2018). Loss surfaces, mode connectivity, and fast ensembling of dnns. In *Advances in Neural Information Processing Systems*, pages 8789–8798.
- Geyer, R. C., Klein, T., and Nabi, M. (2017). Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557*.
- Ghadimi, S. and Lan, G. (2016). Accelerated gradient methods for nonconvex nonlinear and stochastic programming. *Mathematical Programming*, 156(1-2):59–99.
- Gidel, G., Berard, H., Vignoud, G., Vincent, P., and Lacoste-Julien, S. (2019). A variational inequality perspective on generative adversarial networks. In *International Conference on Learning Representations*.
- Gitman, I., Lang, H., Zhang, P., and Xiao, L. (2019). Understanding the role of momentum in stochastic gradient methods. In *Advances in Neural Information Processing Systems*, pages 9633–9643.
- Golatkar, A. S., Achille, A., and Soatto, S. (2019). Time matters in regularizing deep networks: Weight decay and data augmentation affect early learning dynamics, matter little near convergence. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 10678–10688.
- Goldberg, A. E. (1995). *Construction grammar*. Wiley.
- Golmant, N., Vemuri, N., Yao, Z., Feinberg, V., Gholami, A., Rothauge, K., Mahoney, M. W., and Gonzalez, J. (2018). On the computational inefficiency of large batch sizes for stochastic gradient descent. *arXiv preprint arXiv:1811.12941*.
- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.
- Goodfellow, I. J., Vinyals, O., and Saxe, A. M. (2015). Qualitatively characterizing neural network optimization problems. In *ICLR - International Conference on Learning Representations*.
- Gotmare, A., Keskar, N. S., Xiong, C., and Socher, R. (2019). A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation. In *International Conference on Learning Representations*.
- Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. (2017). Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*.

Bibliography

- Guha, N., Talwalkar, A., and Smith, V. (2019). One-shot federated learning. *arXiv preprint arXiv:1902.11175*.
- Guo, T., Lin, T., and Antulov-Fantulin, N. (2019). Exploring interpretable lstm neural networks over multi-variable data. In *International Conference on Machine Learning*, pages 2494–2504. PMLR.
- Guo, Y., Lin, T., and Tang, X. (2021). Towards federated learning on time-evolving heterogeneous data. *arXiv preprint arXiv:2112.13246*.
- Guo, Y., Yao, A., and Chen, Y. (2016). Dynamic network surgery for efficient DNNs. In *NeurIPS - Advances in Neural Information Processing Systems*, pages 1379–1387.
- Gupta, V., Serrano, S. A., and DeCoste, D. (2020). Stochastic weight averaging in parallel: Large-batch training that generalizes well. In *International Conference on Learning Representations (ICLR)*.
- Haddadpour, F., Kamani, M. M., Mokhtari, A., and Mahdavi, M. (2021). Federated learning with compression: Unified analysis and sharp guarantees. In *International Conference on Artificial Intelligence and Statistics*, pages 2350–2358. PMLR.
- Han, S., Pool, J., Narang, S., Mao, H., Tang, S., Elsen, E., Catanzaro, B., Tran, J., and Dally, W. J. (2017). DSD: regularizing deep neural networks with dense-sparse-dense training flow. In *ICLR - International Conference on Learning Representations*.
- Han, S., Pool, J., Tran, J., and Dally, W. (2015). Learning both weights and connections for efficient neural network. In *NeurIPS - Advances in Neural Information Processing Systems*, pages 1135–1143.
- Haruki, K., Suzuki, T., Hamakawa, Y., Toda, T., Sakai, R., Ozawa, M., and Kimura, M. (2019). Gradient noise convolution (gnc): Smoothing loss function for distributed large-batch sgd. *arXiv preprint arXiv:1906.10822*.
- Hassibi, B. and Stork, D. (1992). Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, volume 5.
- He, C., Avestimehr, S., and Annavaram, M. (2020a). Group knowledge transfer: Collaborative training of large cnns on the edge. In *Advances in Neural Information Processing Systems*.
- He, C., Li, S., So, J., Zhang, M., Wang, H., Wang, X., Vepakomma, P., Singh, A., Qiu, H., Shen, L., et al. (2020b). Fedml: A research library and benchmark for federated machine learning. *arXiv preprint arXiv:2007.13518*.
- He, H. and Choi, J. (2020). Establishing strong baselines for the new decade: Sequence tagging, syntactic and semantic parsing with bert. In *The Thirty-Third International Flairs Conference*.

- He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.
- He, K., Zhang, X., Ren, S., and Sun, J. (2016a). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.
- He, K., Zhang, X., Ren, S., and Sun, J. (2016b). Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer.
- He, T., Zhang, Z., Zhang, H., Zhang, Z., Xie, J., and Li, M. (2019a). Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 558–567.
- He, Y., Kang, G., Dong, X., Fu, Y., and Yang, Y. (2018). Soft filter pruning for accelerating deep convolutional neural networks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2234–2240.
- He, Y., Liu, P., Wang, Z., Hu, Z., and Yang, Y. (2019b). Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Hendriks, H., Bach, F., and Massoulié, L. (2019). Accelerated decentralized optimization with local updates for smooth and strongly convex objectives. In *International Conference on Artificial Intelligence and Statistics*, pages 897–906. PMLR.
- Hinton, G. (2012). Neural networks for machine learning.
- Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.
- Hochreiter, S. and Schmidhuber, J. (1997a). Flat minima. *Neural Computation*, 9(1):1–42.
- Hochreiter, S. and Schmidhuber, J. (1997b). Long short-term memory. *Neural computation*, 9(8):1735–1780.
- Hoffer, E., Banner, R., Golan, I., and Soudry, D. (2018). Norm matters: efficient and accurate normalization schemes in deep networks. In *Advances in Neural Information Processing Systems*, pages 2160–2170.
- Hoffer, E., Hubara, I., and Soudry, D. (2017). Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *NIPS - Advances in Neural Information Processing Systems*, pages 1731–1741.
- Hoffman, J., Mohri, M., and Zhang, N. (2018). Algorithms and theory for multiple-source adaptation. In *Advances in Neural Information Processing Systems*, pages 8246–8256.

Bibliography

- Hong, M., Hajinezhad, D., and Zhao, M.-M. (2017). Prox-PDA: The proximal primal-dual algorithm for fast distributed nonconvex optimization and learning over networks. In *International Conference on Machine Learning (ICML)*, pages 1529–1538.
- Horváth, S., Kovalev, D., Mishchenko, K., Stich, S., and Richtárik, P. (2019). Stochastic distributed learning with gradient quantization and variance reduction. *arXiv preprint arXiv:1904.05115*.
- Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. (2019). Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR.
- Howard, J. and Ruder, S. (2018). Universal language model fine-tuning for text classification.
- Hsieh, K., Phanishayee, A., Mutlu, O., and Gibbons, P. (2020). The non-iid data quagmire of decentralized machine learning. In *International Conference on Machine Learning*, pages 4387–4398. PMLR.
- Hsu, T.-M. H., Qi, H., and Brown, M. (2019). Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335*.
- Hsu, T.-M. H., Qi, H., and Brown, M. (2020). Federated visual classification with real-world data distribution. In *European Conference on Computer Vision (ECCV)*.
- Hu, W., Li, C. J., Li, L., and Liu, J.-G. (2019). On the diffusion approximation of nonconvex stochastic gradient descent. *Annals of Mathematical Sciences and Applications*, 4(1).
- Huang, G., Li, Y., Pleiss, G., Liu, Z., Hopcroft, J. E., and Weinberger, K. Q. (2017a). Snapshot ensembles: Train 1, get m for free. In *ICLR - International Conference on Learning Representations*.
- Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017b). Densely connected convolutional networks. In *CVPR - Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708.
- Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. Q. (2016). Deep networks with stochastic depth. In *European Conference on Computer Vision*, pages 646–661. Springer.
- Huang, Z. and Wang, N. (2017). Like what you like: Knowledge distill via neuron selectivity transfer. *arXiv preprint arXiv:1707.01219*.
- Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. (2016). Binarized neural networks. In *Advances in neural information processing systems*, volume 29.
- Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. (2017). Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898.

- Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR.
- Izmailov, P., Podoprikin, D., Gariyov, T., Vetrov, D., and Wilson, A. G. (2018). Averaging weights leads to wider optima and better generalization. In *UAI - Conference on Uncertainty in Artificial Intelligence*.
- Jain, P., Kakade, S., Kidambi, R., Netrapalli, P., and Sidford, A. (2018). Parallelizing stochastic gradient descent for least squares regression: mini-batching, averaging, and model misspecification. *Journal of Machine Learning Research*, 18.
- Jastrzębski, S., Kenton, Z., Arpit, D., Ballas, N., Fischer, A., Bengio, Y., and Storkey, A. (2018). Three factors influencing minima in SGD. In *International Conference on Artificial Neural Networks 2018*.
- Jastrzebski, S., Kenton, Z., Ballas, N., Fischer, A., Bengio, Y., and Storkey, A. (2019). On the relation between the sharpest directions of DNN loss and the SGD step length. In *International Conference on Learning Representations (ICLR)*.
- Jastrzebski, S., Szymczak, M., Fort, S., Arpit, D., Tabor, J., Cho, K., and Geras, K. (2020). The break-even point on optimization trajectories of deep neural networks. In *International Conference on Learning Representations (ICLR)*.
- Jeong, E., Oh, S., Kim, H., Park, J., Bennis, M., and Kim, S.-L. (2018). Communication-efficient on-device machine learning: Federated distillation and augmentation under non-iid private data. *arXiv preprint arXiv:1811.11479*.
- Jiang, L. and Lin, T. (2022). Test-time robust personalization for federated learning. *arXiv preprint arXiv:2205.10920*.
- Jiang, Z., Balu, A., Hegde, C., and Sarkar, S. (2017). Collaborative deep learning in fixed topology networks. *Advances in Neural Information Processing Systems*, 30.
- Jiang, Z., Balu, A., Hegde, C., and Sarkar, S. (2021). On consensus-optimality trade-offs in collaborative deep learning. *Frontiers in artificial intelligence*, page 130.
- Johnson, T., Agrawal, P., Gu, H., and Guestrin, C. (2020). AdaScale SGD: A user-friendly algorithm for distributed training. In *International Conference on Machine Learning (ICML)*, pages 4911–4920.
- Juditsky, A., Nemirovski, A., and Tauvel, C. (2011). Solving variational inequalities with stochastic mirror-prox algorithm. *Stochastic Systems*, 1(1):17–58.
- Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., et al. (2021). Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210.

Bibliography

- Karimireddy, S. P., Jaggi, M., Kale, S., Mohri, M., Reddi, S. J., Stich, S. U., and Suresh, A. T. (2021). Mime: Mimicking centralized stochastic algorithms in federated learning. In *Advances in Neural Information Processing Systems 34*.
- Karimireddy, S. P., Kale, S., Mohri, M., Reddi, S. J., Stich, S. U., and Suresh, A. T. (2020). Scaffold: Stochastic controlled averaging for on-device federated learning. In *International Conference on Machine Learning*. PMLR.
- Karimireddy, S. P., Rebjock, Q., Stich, S. U., and Jaggi, M. (2019). Error feedback fixes SignSGD and other gradient compression schemes. In *ICML - Proceedings of the 36th International Conference on Machine Learning*, pages 3252–3261.
- Kempe, D., Dobra, A., and Gehrke, J. (2003). Gossip-based computation of aggregate information. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pages 482–491. IEEE.
- Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. (2017). On large-batch training for deep learning: Generalization gap and sharp minima. In *ICLR - International Conference on Learning Representations*.
- Kim, J., Park, S., and Kwak, N. (2018). Paraphrasing complex network: Network compression via factor transfer. In *Advances in Neural Information Processing Systems*, pages 2760–2769.
- Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *ICLR - International Conference on Learning Representations*.
- Kleinberg, B., Li, Y., and Yuan, Y. (2018). An alternative view: When does sgd escape local minima? In *International Conference on Machine Learning*, pages 2698–2707. PMLR.
- Koloskova, A., Lin, T., and Stich, S. U. (2021). An improved analysis of gradient tracking for decentralized machine learning.
- Koloskova, A., Lin, T., Stich, S. U., and Jaggi, M. (2020a). Decentralized deep learning with arbitrary communication compression. In *International Conference on Learning Representations*.
- Koloskova, A., Loizou, N., Boreiri, S., Jaggi, M., and Stich, S. (2020b). A unified theory of decentralized sgd with changing topology and local updates. In *International Conference on Machine Learning*, pages 5381–5393. PMLR.
- Koloskova, A., Stich, S., and Jaggi, M. (2019). Decentralized stochastic optimization and gossip algorithms with compressed communication. In *International Conference on Machine Learning*, pages 3478–3487. PMLR.
- Komodakis, N. and Zagoruyko, S. (2017). Paying more attention to attention: improving the performance of convolutional neural networks via attention transfer. In *ICLR*.

- Konečný, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., and Bacon, D. (2016). Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*.
- Kong, L., Lin, T., Koloskova, A., Jaggi, M., and Stich, S. U. (2021a). Consensus control for decentralized deep learning. In *International Conference on Machine Learning*.
- Kong, L., Lin, T., Stich, S. U., and Jaggi, M. (2021b). Slimmable training for heterogeneous federated learning systems.
- Koratana, A., Kang, D., Bailis, P., and Zaharia, M. (2019). Lit: Learned intermediate representation training for model compression. In *International Conference on Machine Learning*, pages 3509–3518. PMLR.
- Korpelevich, G. M. (1976). The extragradient method for finding saddle points and other problems. *Matecon*, 12:747–756.
- Kovaleva, O., Romanov, A., Rogers, A., and Rumshisky, A. (2019). Revealing the dark secrets of BERT. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4356–4365, Hong Kong, China. Association for Computational Linguistics.
- Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images.
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.
- Kuncheva, L. I. and Whitaker, C. J. (2003). Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine learning*, 51(2):181–207.
- Lacoste-Julien, S., Schmidt, M., and Bach, F. (2012). A simpler approach to obtaining an $\mathcal{O}(1/t)$ convergence rate for the projected stochastic subgradient method. *arXiv preprint arXiv:1212.2002*.
- LeCun, Y., Denker, J., and Solla, S. (1989). Optimal brain damage. In *Advances in neural information processing systems*, volume 2.
- LeCun, Y., Kanter, I., and Solla, S. A. (1991). Second order properties of error surfaces: Learning time and generalization. In *Advances in neural information processing systems*, pages 918–924.
- Lee, J. D., Lin, Q., Ma, T., and Yang, T. (2015). Distributed stochastic variance reduced gradient methods and a lower bound for communication complexity. *arXiv preprint arXiv:1507.07595*.
- Lee, N., Ajanthan, T., and Torr, P. H. (2019). SNIP: Single-shot network pruning based on connection sensitivity. In *ICLR - International Conference on Learning Representations*.

Bibliography

- Leng, C., Dou, Z., Li, H., Zhu, S., and Jin, R. (2018). Extremely low bit neural network: Squeeze the last bit out with admm. In *Thirty-Second AAAI Conference on Artificial Intelligence*.
- Li, D. and Wang, J. (2019). Fedmd: Heterogenous federated learning via model distillation. *arXiv preprint arXiv:1910.03581*.
- Li, H., De, S., Xu, Z., Studer, C., Samet, H., and Goldstein, T. (2017a). Training quantized nets: A deeper understanding. In *NeurIPS - Advances in Neural Information Processing Systems*, pages 5811–5821.
- Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. (2018). Visualizing the loss landscape of neural nets. In *NeurIPS - Advances in Neural Information Processing Systems*, pages 6391–6401.
- Li, M. (2017). *Scaling distributed machine learning with system and algorithm co-design*. PhD thesis, Intel.
- Li, M., Zhang, T., Chen, Y., and Smola, A. J. (2014). Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 661–670. ACM.
- Li, Q., Tai, C., et al. (2017b). Stochastic modified equations and adaptive stochastic gradient algorithms. In *ICML - Proceedings of the 34th International Conference on Machine Learning*, pages 2101–2110.
- Li, T., Sahu, A. K., Talwalkar, A., and Smith, V. (2020a). Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60.
- Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., and Smith, V. (2020b). Federated optimization in heterogeneous networks. *Proceedings of Machine Learning and Systems*, 2:429–450.
- Li, T., Sanjabi, M., Beirami, A., and Smith, V. (2020c). Fair resource allocation in federated learning. In *International Conference on Learning Representations*.
- Li, X., Jiang, M., Zhang, X., Kamp, M., and Dou, Q. (2021). FedBN: Federated learning on non-IID features via local batch normalization. In *International Conference on Learning Representations*.
- Li, Y., Wei, C., and Ma, T. (2019). Towards explaining the regularization effect of initial large learning rate in training neural networks. In *Advances in Neural Information Processing Systems*, pages 11669–11680.
- Lian, X., Zhang, C., Zhang, H., Hsieh, C.-J., Zhang, W., and Liu, J. (2017). Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. *Advances in Neural Information Processing Systems*, 30.

- Lian, X., Zhang, W., Zhang, C., and Liu, J. (2018). Asynchronous decentralized parallel stochastic gradient descent. In *International Conference on Machine Learning (ICML)*, pages 3043–3052.
- Lin, T., Karimireddy, S. P., Stich, S. U., and Jaggi, M. (2021). Quasi-global momentum: Accelerating decentralized deep learning on heterogeneous data. In *International Conference on Machine Learning*.
- Lin, T., Kong, L., Stich, S., and Jaggi, M. (2020a). Extrapolation for large-batch training in deep learning. In *International Conference on Machine Learning*, pages 6094–6104. PMLR.
- Lin, T., Kong, L., Stich, S. U., and Jaggi, M. (2020b). Ensemble distillation for robust model fusion in federated learning. In *Advances in Neural Information Processing Systems*.
- Lin, T., Stich, S. U., Barba, L., Dmitriev, D., and Jaggi, M. (2020c). Dynamic model pruning with feedback. In *International Conference on Learning Representations*.
- Lin, T., Stich, S. U., Patel, K. K., and Jaggi, M. (2020d). Don’t use large mini-batches, use local sgd. In *International Conference on Learning Representations*.
- Lin, Y., Han, S., Mao, H., Wang, Y., and Dally, B. (2018). Deep gradient compression: Reducing the communication bandwidth for distributed training. In *International Conference on Learning Representations*.
- Liu, C., Salzmann, M., Lin, T., Tomioka, R., and Süsstrunk, S. (2020a). On the loss landscape of adversarial training: Identifying challenges and how to overcome them. In *Advances in Neural Information Processing Systems*.
- Liu, F., Lin, T., and Jaggi, M. (2021a). Understanding memorization from the perspective of optimization via efficient influence estimation.
- Liu, H., Brock, A., Simonyan, K., and Le, Q. (2020b). Evolving normalization-activation layers. *Advances in Neural Information Processing Systems*, 33:13539–13550.
- Liu, I.-J., Peng, J., and Schwing, A. (2019a). Knowledge flow: Improve upon your teachers. In *International Conference on Learning Representations*.
- Liu, N. F., Gardner, M., Belinkov, Y., Peters, M. E., and Smith, N. A. (2019b). Linguistic knowledge and transferability of contextual representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1073–1094, Minneapolis, Minnesota. Association for Computational Linguistics.
- Liu, Y., Lin, T., Koloskova, A., and Stich, S. U. (2021b). Communication-efficient gradient tracking for decentralized deep learning.
- Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019c). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Bibliography

- Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. (2019d). Rethinking the value of network pruning. In *ICLR - International Conference on Learning Representations*.
- Loshchilov, I. and Hutter, F. (2017). SGDR: stochastic gradient descent with restarts. In *ICLR - International Conference on Learning Representations*.
- Louizos, C., Welling, M., and Kingma, D. P. (2018). Learning sparse neural networks through l_0 regularization. In *ICLR - International Conference on Learning Representations*.
- Lu, S., Zhang, X., Sun, H., and Hong, M. (2019). GNSD: A gradient-tracking based nonconvex stochastic algorithm for decentralized optimization. In *2019 IEEE Data Science Workshop (DSW)*, pages 315–321. IEEE.
- Lu, Y. and De Sa, C. (2020). Moniqua: Modulo quantized communication in decentralized sgd. In *International Conference on Machine Learning*.
- Lucas, J., Sun, S., Zemel, R., and Grosse, R. (2019). Aggregated momentum: Stability through passive damping. In *International Conference on Learning Representations*.
- Luo, P., Wang, X., Shao, W., and Peng, Z. (2019a). Towards understanding regularization in batch normalization. In *International Conference on Learning Representations*.
- Luo, Q., Lin, J., Zhuo, Y., and Qian, X. (2019b). Hop: Heterogeneity-aware decentralized training. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 893–907.
- Lym, S., Choukse, E., Zangeneh, S., Wen, W., Erez, M., and Shanhavi, S. (2019). Prunetrain: Gradual structured pruning from scratch for faster neural network training. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*.
- Ma, J. and Yarats, D. (2019). Quasi-hyperbolic momentum and Adam for deep learning. In *International Conference on Learning Representations*.
- Ma, N., Zhang, X., Zheng, H.-T., and Sun, J. (2018a). Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 116–131.
- Ma, S., Bassily, R., and Belkin, M. (2018b). The power of interpolation: Understanding the effectiveness of sgd in modern over-parametrized learning. In *International Conference on Machine Learning*, pages 3325–3334. PMLR.
- Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. (2011). Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.
- Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

- Maddox, W. J., Izmailov, P., Garipov, T., Vetrov, D. P., and Wilson, A. G. (2019). A simple baseline for bayesian uncertainty in deep learning. In *Advances in Neural Information Processing Systems*, pages 13153–13164.
- Mallya, A., Davis, D., and Lazebnik, S. (2018). Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 67–82.
- Mandt, S., Hoffman, M. D., and Blei, D. M. (2017). Stochastic gradient descent as approximate bayesian inference. *The Journal of Machine Learning Research*, 18(1):4873–4907.
- Mansour, Y., Mohri, M., and Rostamizadeh, A. (2009). Domain adaptation with multiple sources. In *Advances in neural information processing systems*, pages 1041–1048.
- Marcinkiewicz, M. A. (1994). Building a large annotated corpus of english: The penn treebank. *Using Large Corpora*, page 273.
- Martens, J. and Sutskever, I. (2012). Training deep and recurrent networks with hessian-free optimization. In *Neural networks: Tricks of the trade*, pages 479–535. Springer.
- Masters, D. and Luschi, C. (2018). Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*.
- McCandlish, S., Kaplan, J., Amodei, D., and Team, O. D. (2018). An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*.
- McCoy, T., Pavlick, E., and Linzen, T. (2019). Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3428–3448, Florence, Italy. Association for Computational Linguistics.
- McDonald, R., Hall, K., and Mann, G. (2010). Distributed training strategies for the structured perceptron. In *Human language technologies: The 2010 annual conference of the North American chapter of the association for computational linguistics*, pages 456–464.
- McDonald, R., Mohri, M., Silberman, N., Walker, D., and Mann, G. (2009). Efficient large-scale distributed training of conditional maximum entropy models. volume 22.
- McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. (2017). Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR.
- McMahan, H. B., Moore, E., Ramage, D., and y Arcas, B. A. (2016). Federated learning of deep networks using model averaging. *arXiv preprint arXiv:1602.05629*.
- Merity, S., Keskar, N. S., and Socher, R. (2018). Regularizing and optimizing lstm language models. In *ICLR - International Conference on Learning Representations*.

Bibliography

- Merity, S., Xiong, C., Bradbury, J., and Socher, R. (2017). Pointer sentinel mixture models. In *ICLR - International Conference on Learning Representations*.
- Mi, F., Kong, L., Lin, T., Yu, K., and Faltings, B. (2020). Generalized class incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 240–241.
- Mi, F., Lin, T., and Faltings, B. (2021). Representation memorization for fast learning new knowledge without forgetting. *arXiv preprint arXiv:2108.12596*.
- Micaelli, P. and Storkey, A. J. (2019). Zero-shot knowledge transfer via adversarial belief matching. In *Advances in Neural Information Processing Systems*, pages 9547–9557.
- Mishchenko, K., Gorbunov, E., Takáč, M., and Richtárik, P. (2019). Distributed learning with compressed gradient differences. *arXiv preprint arXiv:1901.09269*.
- Mishchenko, K., Kovalev, D., Shulgin, E., Richtárik, P., and Malitsky, Y. (2020). Revisiting stochastic extragradient. In *International Conference on Artificial Intelligence and Statistics*, pages 4573–4582. PMLR.
- Mocanu, D. C., Mocanu, E., Stone, P., Nguyen, P. H., Gibescu, M., and Liotta, A. (2018). Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):2383.
- Mohri, M., Sivek, G., and Suresh, A. T. (2019). Agnostic federated learning. In *International Conference on Machine Learning*, pages 4615–4625. PMLR.
- Molchanov, D., Ashukha, A., and Vetrov, D. (2017). Variational dropout sparsifies deep neural networks. In *ICML - International Conference on Machine Learning*, pages 2498–2507. JMLR.org.
- Mosbach, M., Andriushchenko, M., and Klakow, D. (2021). On the stability of fine-tuning {bert}: Misconceptions, explanations, and strong baselines. In *International Conference on Learning Representations*.
- Mostafa, H. and Wang, X. (2019). Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *ICML - International Conference on Machine Learning*.
- Mozer, M. C. and Smolensky, P. (1989). Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *NeurIPS - Advances in Neural Information Processing Systems*, pages 107–115.
- Nadiradze, G., Sabour, A., Davies, P., Li, S., and Alistarh, D. (2021). Asynchronous decentralized sgd with quantized and local updates. volume 34.

- Nakov, P., Ritter, A., Rosenthal, S., Sebastiani, F., and Stoyanov, V. (2016). SemEval-2016 task 4: Sentiment analysis in twitter. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1–18, San Diego, California. Association for Computational Linguistics.
- Narang, S., Elsen, E., Diamos, G., and Sengupta, S. (2017). Exploring sparsity in recurrent neural networks. In *ICLR - International Conference on Learning Representations*.
- Nayak, G. K., Mopuri, K. R., Shaj, V., Radhakrishnan, V. B., and Chakraborty, A. (2019). Zero-shot knowledge distillation in deep networks. In *International Conference on Machine Learning*, pages 4743–4751. PMLR.
- Nedic, A. (2020). Distributed gradient methods for convex machine learning problems in networks: Distributed optimization. *IEEE Signal Processing Magazine*, 37(3):92–101.
- Nedic, A., Olshevsky, A., Ozdaglar, A., and Tsitsiklis, J. N. (2008). Distributed subgradient methods and quantization effects. In *2008 47th IEEE Conference on Decision and Control*, pages 4177–4184. IEEE.
- Nedić, A., Olshevsky, A., and Rabbat, M. G. (2018). Network topology and communication-computation tradeoffs in decentralized optimization. *Proceedings of the IEEE*, 106(5):953–976.
- Nedic, A. and Ozdaglar, A. (2009). Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control*, 54(1):48–61.
- Neelakantan, A., Vilnis, L., Le, Q. V., Sutskever, I., Kaiser, L., Kurach, K., and Martens, J. (2015). Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*.
- Neglia, G., Xu, C., Towsley, D., and Calbi, G. (2020). Decentralized gradient methods: does topology matter? In *International Conference on Artificial Intelligence and Statistics (AISTATS)*.
- Neklyudov, K., Molchanov, D., Ashukha, A., and Vetrov, D. P. (2017). Structured bayesian pruning via log-normal multiplicative noise. In *NeurIPS - Advances in Neural Information Processing Systems*, pages 6775–6784.
- Nemirovski, A. (2004). Prox-method with rate of convergence $o(1/t)$ for variational inequalities with lipschitz continuous monotone operators and smooth convex-concave saddle point problems. *SIAM Journal on Optimization*, 15(1):229–251.
- Nesterov, Y. (1983). A method of solving a convex programming problem with convergence rate $o(1/k^2)$. In *Soviet Mathematics Doklady*, volume 27, pages 372–376.
- Nesterov, Y. and Spokoiny, V. (2017). Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566.
- Neyshabur, B. (2017). Implicit regularization in deep learning. *PhD Thesis*, arXiv:1709.01953.

Bibliography

- Niven, T. and Kao, H.-Y. (2019). Probing neural network comprehension of natural language arguments. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4658–4664, Florence, Italy. Association for Computational Linguistics.
- Palogiannidi, E., Kolovou, A., Christopoulou, F., Kokkinos, F., Iosif, E., Malandrakis, N., Papageorgiou, H., Narayanan, S. S., and Potamianos, A. (2016). Tweester at semeval-2016 task 4: Sentiment analysis in twitter using semantic-affective model adaptation. In *SemEval@NAACL-HLT*, pages 155–163.
- Pan, T., Liu, J., and Wang, J. (2020). D-SPIDER-SFO: A decentralized optimization algorithm with faster convergence rate for nonconvex problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1619–1626.
- Park, S. and Kwak, N. (2019). Feed: Feature-level ensemble for knowledge distillation. *arXiv preprint arXiv:1909.10754*.
- Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830.
- Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.
- Peters, M. E., Ruder, S., and Smith, N. A. (2019). To tune or not to tune? adapting pretrained representations to diverse tasks.
- Prasanna, S., Rogers, A., and Rumshisky, A. (2020). When BERT plays the lottery, all tickets are winning.
- Pu, S. and Nedić, A. (2020). Distributed stochastic gradient tracking methods. *Mathematical Programming*, pages 1–49.
- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.
- Radiya-Dixit, E. and Wang, X. (2020). How fine can fine-tuning be? learning efficient language models. In *International Conference on Artificial Intelligence and Statistics*, pages 2435–2443. PMLR.
- Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical*

-
- Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.
- Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. (2016). Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer.
- Recht, B., Re, C., Wright, S., and Niu, F. (2011). Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. *Advances in neural information processing systems*, 24.
- Reddi, S. J., Charles, Z., Zaheer, M., Garrett, Z., Rush, K., Konečný, J., Kumar, S., and McMahan, H. B. (2021). Adaptive federated optimization. In *International Conference on Learning Representations*.
- Reisizadeh, A., Mokhtari, A., Hassani, H., and Pedarsani, R. (2019a). An exact quantized decentralized gradient descent algorithm. *IEEE Transactions on Signal Processing*, 67(19):4934–4947.
- Reisizadeh, A., Taheri, H., Mokhtari, A., Hassani, H., and Pedarsani, R. (2019b). Robust and communication-efficient collaborative learning. *Advances in Neural Information Processing Systems*, 32.
- Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407.
- Romero, A., Ballas, N., Kahou, S. E., Chassang, A., Gatta, C., and Bengio, Y. (2015). Fitnets: Hints for thin deep nets. In *International Conference on Learning Representations*.
- Rosenbrock, H. (1960). An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175–184.
- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252.
- Sagun, L., Evcı, U., Guney, V. U., Dauphin, Y., and Bottou, L. (2018). Empirical analysis of the hessian of over-parametrized neural networks. *ICLR workshop*.
- Sajjad, H., Dalvi, F., Durrani, N., and Nakov, P. (2020). Poor man’s bert: Smaller and faster transformer models. *arXiv preprint arXiv:2004.03844*.
- Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2019). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.
- Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. (2018). How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pages 2483–2493.

Bibliography

- Scaman, K., Bach, F., Bubeck, S., Lee, Y. T., and Massoulié, L. (2017). Optimal algorithms for smooth and strongly convex distributed optimization in networks. In *international conference on machine learning*, pages 3027–3036. PMLR.
- Scaman, K., Bach, F., Bubeck, S., Massoulié, L., and Lee, Y. T. (2018). Optimal algorithms for non-smooth distributed optimization in networks. *Advances in Neural Information Processing Systems*, 31.
- Schwartz, R., Dodge, J., Smith, N. A., and Etzioni, O. (2020). Green ai. *Communications of the ACM*, 63(12):54–63.
- Seide, F. and Agarwal, A. (2016). Cntk: Microsoft’s open-source deep-learning toolkit. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 2135–2135.
- Seide, F., Fu, H., Droppo, J., Li, G., and Yu, D. (2014). 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns. In *Fifteenth Annual Conference of the International Speech Communication Association*. Citeseer.
- Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge university press.
- Shallue, C. J., Lee, J., Antognini, J., Sohl-Dickstein, J., Frostig, R., and Dahl, G. E. (2019). Measuring the effects of data parallelism on neural network training. *Journal of Machine Learning Research* 20 (2019) 1-49.
- Sharma, C., Narayanan, V., and Balamurugan, P. (2019). A simple and fast distributed accelerated gradient method. In *OPT2019: 11th Annual Workshop on Optimization for Machine Learning*.
- Shi, W., Ling, Q., Wu, G., and Yin, W. (2015). Extra: An exact first-order algorithm for decentralized consensus optimization. *SIAM Journal on Optimization*, 25(2):944–966.
- Shoham, N., Avidor, T., Keren, A., Israel, N., Benditkis, D., Mor-Yosef, L., and Zeitak, I. (2019). Overcoming forgetting in federated learning on non-iid data. *arXiv preprint arXiv:1910.07796*.
- Shokri, R. and Shmatikov, V. (2015). Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1310–1321.
- Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *ICLR - International Conference on Learning Representations*.
- Singh, N., Data, D., George, J., and Diggavi, S. (2021). Squarm-sgd: Communication-efficient momentum sgd for decentralized optimization. *IEEE Journal on Selected Areas in Information Theory*, 2(3):954–969.
- Singh, S. P. and Jaggi, M. (2020). Model fusion via optimal transport. In *Advances in Neural Information Processing Systems*.

- Smith, S. L., Kindermans, P.-J., Ying, C., and Le, Q. V. (2018). Don't decay the learning rate, increase the batch size. In *ICLR - International Conference on Learning Representations*.
- Smith, S. L. and Le, Q. V. (2018). A bayesian perspective on generalization and stochastic gradient descent. In *ICLR - International Conference on Learning Representations*.
- Smith, V., Chiang, C.-K., Sanjabi, M., and Talwalkar, A. S. (2017). Federated multi-task learning. In *Advances in Neural Information Processing Systems*, pages 4424–4434.
- Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., and Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.
- Sollich, P. and Krogh, A. (1996). Learning with ensembles: How overfitting can be useful. In *Advances in neural information processing systems*, pages 190–196.
- Srinivas, S., Subramanya, A., and Venkatesh Babu, R. (2017). Training sparse neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 138–145.
- Stich, S., Mohtashami, A., and Jaggi, M. (2021). Critical parameters for scalable distributed learning with large batches and asynchronous updates. In *International Conference on Artificial Intelligence and Statistics*, pages 4042–4050. PMLR.
- Stich, S. U. (2019a). Local SGD converges fast and communicates little. In *ICLR - International Conference on Learning Representations*.
- Stich, S. U. (2019b). Unified optimal analysis of the (stochastic) gradient method. *arXiv preprint arXiv:1907.04232*.
- Stich, S. U., Cordonnier, J.-B., and Jaggi, M. (2018). Sparsified sgd with memory. *Advances in Neural Information Processing Systems*, 31.
- Stich, S. U. and Karimireddy, S. P. (2020). The error-feedback framework: Better rates for sgd with delayed gradients and compressed updates. *Journal of Machine Learning Research*, 21:1–36.
- Stickland, A. C. and Murray, I. (2019). Bert and pals: Projected attention layers for efficient adaptation in multi-task learning. In *International Conference on Machine Learning*, pages 5986–5995. PMLR.
- Strom, N. (2015). Scalable distributed dnn training using commodity gpu cloud computing. In *Sixteenth Annual Conference of the International Speech Communication Association*.
- Strubell, E., Ganesh, A., and McCallum, A. (2019). Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy. Association for Computational Linguistics.

Bibliography

- Su, J., Wen, Z., Lin, T., and Guan, Y. (2022). Learning disentangled behaviour patterns for wearable-based human activity recognition. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 6(1):1–19.
- Sun, C., Qiu, X., Xu, Y., and Huang, X. (2019). How to fine-tune bert for text classification? In *China National Conference on Chinese Computational Linguistics*, pages 194–206. Springer.
- Sun, H. and Hong, M. (2019). Distributed non-convex first-order optimization and information processing: Lower complexity bounds and rate optimal algorithms. *IEEE Transactions on Signal processing*, 67(22):5912–5928.
- Sun, L. and Lyu, L. (2021). Federated model distillation with noise-free differential privacy. In *IJCAI-2021 - Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*.
- Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning*, pages 1139–1147. PMLR.
- Taheri, H., Mokhtari, A., Hassani, H., and Pedarsani, R. (2020). Quantized decentralized stochastic learning over directed graphs. In *International Conference on Machine Learning*, pages 9324–9333. PMLR.
- Takáč, M., Bijral, A., Richtárik, P., and Srebro, N. (2013). Mini-batch primal and dual methods for svms. In *International Conference on Machine Learning*, pages 1022–1030. PMLR.
- Tan, M. and Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR.
- Tang, H., Gan, S., Zhang, C., Zhang, T., and Liu, J. (2018a). Communication compression for decentralized training. *Advances in Neural Information Processing Systems*, 31.
- Tang, H., Lian, X., Qiu, S., Yuan, L., Zhang, C., Zhang, T., and Liu, J. (2019). Deepsqueeze: Decentralization meets error-compensated compression. *arXiv preprint arXiv:1907.07346*.
- Tang, H., Lian, X., Yan, M., Zhang, C., and Liu, J. (2018b). d^2 : Decentralized training over decentralized data. In *International Conference on Machine Learning*, pages 4848–4856. PMLR.
- Tenney, I., Das, D., and Pavlick, E. (2019). BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601, Florence, Italy. Association for Computational Linguistics.
- Tian, Y., Krishnan, D., and Isola, P. (2020). Contrastive representation distillation. In *International Conference on Learning Representations*.

- Tjong Kim Sang, E. F. and De Meulder, F. (2003). Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.
- Tsianos, K. I. and Rabbat, M. G. (2016). Efficient distributed online prediction and stochastic optimization with approximate distributed averaging. *IEEE Transactions on Signal and Information Processing over Networks*, 2(4):489–506.
- Tsitsiklis, J. N. (1984). Problems in decentralized decision making and computation. Technical report, Massachusetts Inst of Tech Cambridge Lab for Information and Decision Systems.
- Tung, F. and Mori, G. (2019). Similarity-preserving knowledge distillation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1365–1374.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.
- Vogels, T., He, L., Koloskova, A., Lin, T., Karimireddy, S. P., Stich, S. U., and Jaggi, M. (2021). Relaysun for decentralized deep learning on heterogeneous data.
- Vogels, T., Karimireddy, S. P., and Jaggi, M. (2020). Powergossip: Practical low-rank communication compression in decentralized deep learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Voorhees, E. M., Tice, D. M., et al. (1999). The trec-8 question answering track evaluation. In *TREC*, volume 1999, page 82. Citeseer.
- Wang, A., Pruksachatkun, Y., Nangia, N., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. (2019a). Superglue: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems*, pages 3261–3275.
- Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2018). Glue: A multi-task benchmark and analysis platform for natural language understanding. In *ACL - Association for Computational Linguistics*.
- Wang, H., Yurochkin, M., Sun, Y., Papailiopoulos, D., and Khazaeni, Y. (2020a). Federated learning with matched averaging. In *International Conference on Learning Representations*.
- Wang, J. and Joshi, G. (2019). Adaptive communication strategies to achieve the best error-runtime trade-off in local-update SGD. In *SysML*.
- Wang, J., Liu, Q., Liang, H., Joshi, G., and Poor, H. V. (2020b). Tackling the objective inconsistency problem in heterogeneous federated optimization. In *Advances in Neural Information Processing Systems*.

Bibliography

- Wang, J., Sahu, A. K., Joshi, G., and Kar, S. (2020c). Exploring the error-runtime trade-off in decentralized optimization. In *2020 54th Asilomar Conference on Signals, Systems, and Computers*, pages 910–914.
- Wang, J., Sahu, A. K., Yang, Z., Joshi, G., and Kar, S. (2019b). Matcha: Speeding up decentralized sgd via matching decomposition sampling. In *2019 Sixth Indian Control Conference (ICC)*, pages 299–300. IEEE.
- Wang, J., Tantia, V., Ballas, N., and Rabbat, M. (2020d). SlowMo: Improving communication-efficient distributed SGD with slow momentum. In *International Conference on Learning Representations (ICLR)*.
- Wang, W. and Srebro, N. (2019). Stochastic nonconvex optimization with large minibatches. In *Algorithmic Learning Theory*, pages 857–882. PMLR.
- Wangni, J., Wang, J., Liu, J., and Zhang, T. (2018). Gradient sparsification for communication-efficient distributed optimization. *Advances in Neural Information Processing Systems*, 31.
- Warstadt, A., Singh, A., and Bowman, S. R. (2019). Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641.
- Wen, W., Wang, Y., Yan, F., Xu, C., Wu, C., Chen, Y., and Li, H. (2018). Smoothout: Smoothing out sharp minima to improve generalization in deep learning. *arXiv preprint arXiv:1805.07898*.
- Wen, W., Xu, C., Yan, F., Wu, C., Wang, Y., Chen, Y., and Li, H. (2017). Terngrad: Ternary gradients to reduce communication in distributed deep learning. In *Advances in neural information processing systems*, volume 30.
- Wen, Y., Luk, K., Gazeau, M., Zhang, G., Chan, H., and Ba, J. (2019). Interplay between optimization and generalization of stochastic gradient descent with covariance noise. *arXiv preprint arXiv:1902.08234*.
- Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. (2019). Huggingface’s transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.
- Wu, A., Zheng, W.-S., Guo, X., and Lai, J.-H. (2019). Distilled person re-identification: Towards a more scalable system. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1187–1196.
- Wu, Y. and He, K. (2018). Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19.
- Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google’s neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

- Xiao, L. and Boyd, S. (2004). Fast linear iterations for distributed averaging. *Systems & Control Letters*, 53(1):65–78.
- Xiao, L., Boyd, S., and Lall, S. (2005). A scheme for robust distributed sensor fusion based on average consensus. In *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.*, pages 63–70. IEEE.
- Xin, R., Khan, U. A., and Kar, S. (2020). Variance-reduced decentralized stochastic optimization with accelerated convergence. *IEEE Transactions on Signal Processing*, 68:6255–6271.
- Xing, C., Arpit, D., Tsirigotis, C., and Bengio, Y. (2018). A walk with SGD. *arXiv preprint arXiv:1802.08770*.
- Xu, Y., Yuan, Z., Yang, S., Jin, R., and Yang, T. (2019). On the convergence of (stochastic) gradient descent with extrapolation for non-convex minimization. In *IJCAI*, pages 4003–4009.
- Yan, Y., Yang, T., Li, Z., Lin, Q., and Yang, Y. (2018). A unified analysis of stochastic momentum methods for deep learning. In *International Joint Conference on Artificial Intelligence*, pages 2955–2961.
- Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., and Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, volume 32.
- Yao, Z., Gholami, A., Lei, Q., Keutzer, K., and Mahoney, M. W. (2018). Hessian-based analysis of large batch training and robustness to adversaries. In *NeurIPS - Advances in Neural Information Processing Systems*, pages 4949–4959.
- Ye, J., Lu, X., Lin, Z., and Wang, J. Z. (2018). Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *ICLR - International Conference on Learning Representations*.
- Yin, D., Pananjady, A., Lam, M., Papailiopoulos, D., Ramchandran, K., and Bartlett, P. (2018). Gradient diversity: a key ingredient for scalable distributed learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1998–2007.
- Yin, P., Lyu, J., Zhang, S., Osher, S., Qi, Y., and Xin, J. (2019). Understanding straight-through estimator in training activation quantized neural nets. *arXiv preprint arXiv:1903.05662*.
- You, K., Long, M., Wang, J., and Jordan, M. I. (2020a). How does learning rate decay help modern neural networks?
- You, S., Xu, C., Xu, C., and Tao, D. (2017a). Learning from multiple teacher networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1285–1294.
- You, Y., Gitman, I., and Ginsburg, B. (2017b). Scaling sgd batch size to 32k for imagenet training. *arXiv preprint arXiv:1708.03888*, 6(12):6.

Bibliography

- You, Y., Hseu, J., Ying, C., Demmel, J., Keutzer, K., and Hsieh, C.-J. (2019a). Large-batch training for lstm and beyond. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16.
- You, Y., Li, J., Reddi, S., Hseu, J., Kumar, S., Bhojanapalli, S., Song, X., Demmel, J., Keutzer, K., and Hsieh, C.-J. (2020b). Large batch optimization for deep learning: Training bert in 76 minutes. In *International Conference on Learning Representations*.
- You, Y., Zhang, Z., Demmel, J., Keutzer, K., and Hsieh, C.-J. (2017c). ImageNet training in 24 minutes. *arXiv preprint arXiv:1709.05011*.
- You, Y., Zhang, Z., Hsieh, C.-J., Demmel, J., and Keutzer, K. (2018). Imagenet training in minutes. In *Proceedings of the 47th International Conference on Parallel Processing*. ACM.
- You, Z., Yan, K., Ye, J., Ma, M., and Wang, P. (2019b). Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 32.
- Yu, H., Jin, R., and Yang, S. (2019a). On the linear speedup analysis of communication efficient momentum sgd for distributed non-convex optimization. In *International Conference on Machine Learning*, pages 7184–7193. PMLR.
- Yu, H., Yang, S., and Zhu, S. (2019b). Parallel restarted sgd with faster convergence and less communication: Demystifying why model averaging works for deep learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5693–5700.
- Yuan, D., Xu, S., Zhao, H., and Rong, L. (2012). Distributed dual averaging method for multi-agent optimization with quantized communication. *Systems & Control Letters*, 61(11):1053–1061.
- Yuan, K. and Alghunaim, S. A. (2021). Removing data heterogeneity influence enhances network topology dependence of decentralized sgd. *arXiv preprint arXiv:2105.08023*.
- Yuan, K., Alghunaim, S. A., Ying, B., and Sayed, A. H. (2020a). On the influence of bias-correction on distributed stochastic optimization. *IEEE Transactions on Signal Processing*, 68:4352–4367.
- Yuan, K., Ling, Q., and Yin, W. (2016). On the convergence of decentralized gradient descent. *SIAM Journal on Optimization*, 26(3):1835–1854.
- Yuan, K., Xu, W., and Ling, Q. (2020b). Can primal methods outperform primal-dual methods in decentralized dynamic optimization? *IEEE Transactions on Signal Processing*, 68:4466–4480.
- Yurochkin, M., Agarwal, M., Ghosh, S., Greenewald, K., Hoang, N., and Khazaeni, Y. (2019). Bayesian nonparametric federated learning of neural networks. In *International Conference on Machine Learning*, pages 7252–7261. PMLR.
- Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. In *BMVC*.

- Zellers, R., Bisk, Y., Schwartz, R., and Choi, Y. (2018). SWAG: A large-scale adversarial dataset for grounded commonsense inference. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 93–104, Brussels, Belgium. Association for Computational Linguistics.
- Zhang, C., Bengio, S., and Singer, Y. (2019a). Are all layers created equal? *arXiv preprint arXiv:1902.01996*.
- Zhang, G., Wang, C., Xu, B., and Grosse, R. (2019b). Three mechanisms of weight decay regularization. In *International Conference on Learning Representations*.
- Zhang, J., De Sa, C., Mitliagkas, I., and Ré, C. (2016). Parallel sgd: When does averaging help? *arXiv preprint arXiv:1606.07365*.
- Zhang, J., Karimireddy, S. P., Veit, A., Kim, S., Reddi, S. J., Kumar, S., and Sra, S. (2020). Why ADAM beats SGD for attention models. In *Advances in Neural Information Processing Systems*.
- Zhang, X., Trmal, J., Povey, D., and Khudanpur, S. (2014). Improving deep neural network acoustic models using generalized maxout networks. In *2014 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 215–219. IEEE.
- Zhang, X., Zhao, J., and LeCun, Y. (2015). Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, volume 28.
- Zhao, M., Dufter, P., Yaghoobzadeh, Y., and Schütze, H. (2020a). Quantifying the contextualization of word representations with semantic class probing. In *Findings of EMNLP*.
- Zhao, M., Lin, T., Jaggi, M., and Schütze, H. (2020b). Masking as an efficient alternative to finetuning for pretrained language models. In *Empirical Methods in Natural Language Processing*.
- Zhao, Y., Li, M., Lai, L., Suda, N., Civin, D., and Chandra, V. (2018). Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*.
- Zhou, F. and Cong, G. (2018). On the convergence properties of a k-step averaging stochastic gradient descent algorithm for nonconvex optimization. In *IJCAI-18 - Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 3219–3227.
- Zhou, H., Lan, J., Liu, R., and Yosinski, J. (2019). Deconstructing lottery tickets: Zeros, signs, and the supermask. In *Advances in Neural Information Processing Systems*, pages 3592–3602.
- Zhou, Y., Pu, G., Ma, X., Li, X., and Wu, D. (2020). Distilled one-shot federated learning. *arXiv preprint arXiv:2009.07999*.
- Zhu, M. and Gupta, S. (2017). To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*.

Bibliography

Zhu, Z., Wu, J., Yu, B., Wu, L., and Ma, J. (2019). The anisotropic noise in stochastic gradient descent: Its behavior of escaping from minima and regularization effects. In *ICML - Proceedings of the 36th International Conference on Machine Learning*.

Zinkevich, M., Weimer, M., Li, L., and Smola, A. (2010). Parallelized stochastic gradient descent. volume 23.

Tao LIN

CONTACT INFORMATION	INJ 340, EPFL 1015 Lausanne, Switzerland	+41 77 957 5667 tao.lin@epfl.ch tlin-tao-lin.github.io
RESEARCH INTERESTS	Collaborative learning (distributed, federated and decentralized methods); Optimization and generalization for deep learning.	
EMPLOYMENT	EPFL, Switzerland Research assistant advised by Prof. Martin Jaggi and Prof. Babak Falsafi.	02.2017 – present
EDUCATION	École Polytechnique Fédérale de Lausanne (EPFL) , Switzerland Ph.D. student, Computer Science Advisors: Prof. Martin Jaggi and Prof. Babak Falsafi (ACM & IEEE fellow)	Sep. 2017 – Sep. 2022
	École Polytechnique Fédérale de Lausanne (EPFL) , Switzerland M.S., Communication Systems	Sep. 2014 – Feb. 2017
	Zhejiang University (ZJU) , P.R. China B.S., System Science and Engineering (with honors)	Sep. 2010 – Jun. 2014
RESEARCH GRANTS	1. Algorithm Design for Memory Efficient Localized Model Parallelism, 144K CHF from Huawei (serve as co-PI). 01.2022–06.2023. 2. DIGIPREDICT, a cross-disciplinary program (3 million EUR from EU Horizon 2020 projects) coordinated by EPFL and other 9 European partners to develop a digital twin that can detect serious complications in Covid-19 patients (project team member). 01.2021–01.2022. 3. ColTraIn: Co-located DNN Training and Inference, 300K CHF from Microsoft Research (supported Ph.D. student). 01.2017–12.2019.	
PUBLICATIONS	Google Scholar citations: 1380, h-index: 16, i10-index: 20.	

★ indicates equal contribution; † indicates corresponding authorship.

Peer-reviewed conference publications:

1. **Tao Lin**, Sai Praneeth Karimireddy, Sebastian U. Stich, Martin Jaggi. “Quasi-global Momentum: Accelerating Decentralized Deep Learning on Heterogeneous Data”. International Conference on Machine Learning (ICML) 2021.
2. Lingjing Kong*, **Tao Lin***†, Anastasia Koloskova, Martin Jaggi, Sebastian U. Stich. “Consensus Control for Decentralized Deep Learning”. International Conference on Machine Learning (ICML) 2021.
3. **Tao Lin***†, Lingjing Kong*, Sebastian U. Stich, Martin Jaggi. “Ensemble Distillation for Robust Model Fusion in Federated Learning”. Neural Information Processing Systems (NeurIPS) 2020.
4. Mengjie Zhao*, **Tao Lin***, Martin Jaggi, Hinrich Schütze. “Masking as an Efficient Alternative to Finetuning for Pretrained Language Models”. Empirical Methods in Natural Language Processing (EMNLP) 2020.
5. **Tao Lin***†, Lingjing Kong*, Sebastian U. Stich, Martin Jaggi. “Extrapolation for Large-batch Training in Deep Learning”. International Conference on Machine Learning (ICML) 2020.

6. **Tao Lin**, Sebastian U. Stich, Luis Barba, Daniil Dmitriev, Martin Jaggi. “Dynamic Model Pruning with Feedback”. International Conference on Learning Representations (ICLR) 2020.
7. Anastasia Koloskova*, **Tao Lin***, Sebastian U. Stich, Martin Jaggi. “Decentralized Deep Learning with Arbitrary Communication Compression”. International Conference on Learning Representations (ICLR) 2020.
8. **Tao Lin**, Sebastian U. Stich, Kumar Kshitij Patel, Martin Jaggi. “Don’t Use Large Mini-Batches, Use Local SGD”. International Conference on Learning Representations (ICLR) 2020.
9. **Tao Lin***, Tian Guo*, Karl Aberer. “Learning the Trend in Time Series via Hybrid Neural Networks”. International Joint Conference on Artificial Intelligence (IJCAI) 2017.
10. Jie Su, **Tao Lin**, Zhenyu Wen, Yu Guan. “Learning Disentangled Behaviour Patterns for Wearable-based Human Activity Recognition”. ACM International Conference on Ubiquitous Computing (UbiComp) 2022.
11. Thijs Vogels*, Lie He*, Anastasia Koloskova, **Tao Lin**, Sai Praneeth Karimireddy, Sebastian U. Stich, Martin Jaggi. “RelaySum for Decentralized Deep Learning on Heterogeneous Data”. Neural Information Processing Systems (NeurIPS) 2021.
12. Anastasia Koloskova, **Tao Lin**, Sebastian U. Stich. “An Improved Analysis of Gradient Tracking for Decentralized Machine Learning”. Neural Information Processing Systems (NeurIPS) 2021.
13. Chen Liu, Mathieu Salzmann, **Tao Lin**, Ryota Tomioka, Sabine Süsstrunk. “On the Loss Landscape of Adversarial Training: Identifying Challenges and How to Overcome Them”. Neural Information Processing Systems (NeurIPS) 2020.
14. Tian Guo, **Tao Lin**, Nino Antulov-Fantulin. “Exploring Interpretable LSTM Neural Networks over Multi-Variable Data”. International Conference on Machine Learning (ICML) 2019.
15. Mario Drummond, **Tao Lin**, Martin Jaggi, Babak Falsafi. “Training DNNs with Hybrid Block Floating Point”. Neural Information Processing Systems (NeurIPS) 2018.
16. Rachid Guerraoui, Anne-Marie Kermarrec, **Tao Lin**, Richeek Patra (alphabetical order). “Heterogeneous Recommendations: What You Might Like To Read After Watching Interstellar”. International Conference on Very Large Data Bases (VLDB) 2017.

Peer-reviewed journal publications:

1. Jie Su, Zhenyu Wen, **Tao Lin**, Yu Guan. “Learning Disentangled Behaviour Patterns for Wearable-based Human Activity Recognition”. In Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT) 2022.
2. Zhenyu Wen, **Tao Lin**, Renyu Yang, Shouling Ji, Rajiv Ranjan, Alexander Romanovsky, Changting Lin, Jie Xu. “Security-Aware Microservice Orchestration under Uncertainty of Geo-distributed Clouds”. IEEE Transactions on Parallel and Distributed Systems 2019.
3. Hao Li*, Xu Wang*, Daria Rukina, Qingyao Huang, **Tao Lin**, Vincenzo Sorrentino, Hongbo Zhang, Danny Arends, Aaron McDaid, Peiling Luan, Naveed Zari, Laura A. Velázquez-Villegas, Karim Gariani, Zoltan Kutalik, Kristina Schoonjans, Richard A Radcliffe, Pjotr Prins, Stephan Morgenthaler, Robert W. Williams, Johan Auwerx. “An integrated systems genetics and omics toolkit to optimize the identification of gene function”. Cell Systems 2018.
4. Zhenyu Wen, Renyu Yang, Peter Garraghan, **Tao Lin**, Jie Xu and Michael Rovatsos. “Fog Orchestration for IoT Services: Issues, Challenges and Directions”. IEEE Internet Computing, IEEE Computer Society 2017.

5. *Laurent Mouchiroud, Vincenzo Sorrentino, Evan G. Williams, Matteo Cornaglia, Michael V. Prochaux, **Tao Lin**, Amandine A. Nicolet-dit-Félix, Gopal Krishnamani, Tarik Ouhmad, Martin A.M. Gijs, Bart Deplancke, Johan Auwerx.* “The Movement Tracker: A flexible system for automated movement analysis in invertebrate model organisms”. *Current Protocols in Neuroscience* 2016.

Book chapter:

1. *Renyu Yang, Zhenyu Wen, David McKee, **Tao Lin**, Jie Xu and Peter Garraghan.* “Fog Orchestration and Simulation for IoT Services”. Book Chapter, *Fog and Fogonomics Challenges and Practices of Fog Computing, Networking, Strategy, and Economics*, John Wiley & Sons, 2020.

Peer-reviewed workshop publications:

1. *Futong Liu, **Tao Lin**[†], Martin Jaggi.* “Understanding Memorization from the Perspective of Optimization via Efficient Influence Estimation”. *Neural Information Processing Systems (NeurIPS) 2021, OPT workshop track*, oral presentation.
2. *Yongxin Guo*, **Tao Lin***, Xiaoying Tang.* “A New Analysis Framework for Federated Learning on Time-Evolving Heterogeneous Data”. *International Conference on Machine Learning (ICML) 2021, FL-ICML workshop track*.
3. *Fei Mi*, Lingjing Kong*, **Tao Lin**, Kaicheng Yu, Boi Faltings.* “Generalized Class Incremental Learning”. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2020, workshop track*.
4. *Tian Guo*, **Tao Lin***, Yao Lu.* “An interpretable LSTM neural network for autoregressive exogenous model”. *International Conference on Learning Representations (ICLR) 2018, workshop track*.

AWARDS

- Outstanding performance bonus awarded by EPFL, 2021.
- Top Reviewer awarded at AISTATS, ICML, and NeurIPS.
- Teaching Assistant Award for Machine Learning Course, 2016.
- Zhejiang University Outstanding Graduates, 2014.
- 1st prize of MITSUBISHIELECTRIC Automation Competition, 2013.
- Excellent Merit Student of Zhejiang University, 2011, 2012, 2013.

REFERENCES

1. Professor Martin Jaggi at EPFL: martin.jaggi@epfl.ch
2. Professor Babak Falsafi at CMU & EPFL: babak.falsafi@epfl.ch
3. Professor Dan Alistarh at IST Austria: dan.alistarh@ist.ac.at
4. Professor Sebastian U. Stich at CISPA (Helmholtz Center for Information Security/Saarland University): stich@cispa.de

SERVICES

Reviewing activity:

- ICML 2019, 2020, 2021, 2022. Top reviewer at ICML 2020. Expert reviewer at ICML 2021.
- NeurIPS 2019, 2020, 2021. Top reviewer at NeurIPS 2019.
- ICLR 2021, 2022.
- AAAI 2020.
- AISTATS 2022. Top reviewer at AISTATS 2022.
- IEEE Transactions on Pattern Analysis and Machine Intelligence.
- IEEE Transactions on Neural Networks and Learning Systems.
- ACM Computing Surveys.
- Optimization for machine learning (OPT) workshop 2020, 2021.
- IEEE International Conference On Joint Cloud Computing (IEEE JCC) 2019, 2020.

- Workshop on Continual Learning in Computer Vision (CVPR 2022).

Others:

- EPFL EDIC (Doctoral School in Computer and Communication Sciences) admission committee, 2022 Fall.
- Organizing committee of EPFL CIS EdgeAI 2022 Summer School.

TEACHING EXPERIENCE	Teaching assistant: Analysis I	EPFL, Fall 2019
	Instructor: Anna Lachowska, Ph.D.	
	Teaching assistant: Machine Learning	EPFL, Fall 2016/2018/2020/2021
	Instructor: Martin Jaggi, Ph.D. & Nicolas Flammarion, Ph.D.	
	Teaching assistant: Deep Learning	EPFL, Spring 2018/2019
	Instructor: François Fleuret, Ph.D.	
Teaching assistant: Signals and Systems	EPFL, Spring 2020	
Instructor: Yanina Shkel, Ph.D.		
Teaching assistant: Systems for data science	EPFL, Spring 2021	
Instructor: Kermarrec Anne-Marie, Ph.D.		

SUPERVISION
OF JUNIOR
RESEARCHERS

Master theses:

1. Futong Liu (now at Bloomberg), *Understanding Memorization from the Perspective of Optimization via Efficient Influence Estimation* (published at a NeurIPS 2021 workshop). 03.2021–08.2021.
2. Lingjing Kong (now Ph.D. student at CMU), *Slimmable Training for Heterogeneous Federated Learning Systems*. 02.2021–06.2021.
3. Yue Liu (now Ph.D. student at University of Toronto), *Variance Reduction for Decentralized Training over Heterogeneous Data* (under submission). 09.2020–04.2021.
4. Akhilesh Gotmare (now at Salesforce), *Layer-wise Model Parallel Training of Deep Neural Networks*. 09.2018–03.2019.

Semester projects:

1. Aleksandr Timofeev, *Towards a Better Representation Learning for Decentralized Deep Learning under Heterogeneous Data*. 09.2021–present.
2. Wei Shi, *Understanding and Improving Self-Supervised Representation Learning under Uncurated Data*. 09.2021–present.
3. Liangze Jiang, *Towards Personalized Federated Learning with Robustness to Out-of-Distribution Shift*. 09.2021–present.
4. Freya Behrens, *Measuring the Impact of Model and Input Heterogeneity in Federated Learning*. 09.2020–01.2021.
5. Manjot Singh, *Scientific Study on Different Pruning Techniques on BERT Downstream Tasks*. 02.2020–06.2020.
6. Lingjing Kong (now Ph.D. student at CMU), *Ensemble Distillation for Robust Model Fusion in Federated Learning* (published at NeurIPS 2020). 02.2020–06.2020.
7. Fedor Moiseev, *Efficient Federated Deep Learning Algorithm Design with Non-IID Client Data*. 09.2019–01.2020.
8. Lingjing Kong (now Ph.D. student at CMU), *Extrapolation Methods for Large-batch Training* (published at ICML 2020). 02.2019–06.2019.
9. Peilin Kang (now at ByteDance), *A Comparison of Model-parallel Training Methods for Deep Learning*. 02.2019–06.2019.

Junior Ph.D. students mentored:

1. Yongxin Guo, Ph.D. student at The Chinese University of Hong Kong, Shenzhen (CUHK-SZ). 09.2020–present.
2. Jie Su, Ph.D. student at Newcastle University. 02.2020–present.