Machine Learning: Foundations, Methodologies, and Applications

Rachid Guerraoui
Nirupam Gupta
Rafael Pinot

Robust Machine Learning

Distributed Methods for Safe AI

Springer

Machine Learning: Foundations, Methodologies, and Applications

Books published in this series focus on the theory and computational foundations, advanced methodologies and practical applications of machine learning, ideally combining mathematically rigorous treatments of a contemporary topics in machine learning with specific illustrations in relevant algorithm designs and demonstrations in real-world applications. The intended readership includes research students and researchers in computer science, computer engineering, electrical engineering, data science, and related areas seeking a convenient medium to track the progresses made in the foundations, methodologies, and applications of machine learning.

Topics considered include all areas of machine learning, including but not limited to:

- Decision tree
- Artificial neural networks
- Kernel learning
- Bayesian learning
- Ensemble methods
- Dimension reduction and metric learning
- Reinforcement learning
- Meta learning and learning to learn
- Imitation learning
- Computational learning theory
- Probabilistic graphical models
- Transfer learning
- Multi-view and multi-task learning
- Graph neural networks
- Generative adversarial networks
- Federated learning

This series includes monographs, introductory and advanced textbooks, and state-of-the-art collections. Furthermore, it supports Open Access publication mode.

Rachid Guerraoui • Nirupam Gupta • Rafael Pinot

Robust Machine Learning

Distributed Methods for Safe AI

Springer

Rachid Guerraoui
Computer Science
École Polytechnique Fédérale de Lausanne
Lausanne, Switzerland

Nirupam Gupta
Computer Science
École Polytechnique Fédérale de Lausanne
Lausanne, Switzerland

Rafael Pinot
Mathematics
Sorbonne Université
Paris, France

ISSN 2730-9908 ISSN 2730-9916 (electronic)
Machine Learning: Foundations, Methodologies, and Applications
ISBN 978-981-97-0687-7 ISBN 978-981-97-0688-4 (eBook)
https://doi.org/10.1007/978-981-97-0688-4

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Paper in this product is recyclable.

To Anne
To Naniji (Vimal)
To Karin and Philippe

Preface

Over the last two decades, so-called Artificial Intelligence (AI) systems have been capable of impressive feats. They beat world champions in various games (including Chess, Go, and Poker), some of which have been considered, throughout history, as ultimate measures of human intelligence. These systems also exhibited impressive results in several important applications such as scientific discovery, banking, and healthcare. Recently, conversational agents have taken AI to the next level, directly interacting with millions of users. To some extent, these new technologies often pass the celebrated Turing test, as many forget that they are interacting with a machine. Overall, AI-based technologies are capable of incredible prowess, and many expect that this is just the beginning of the "AI era." Nevertheless, beyond these achievements, a fundamental question arises: *Can we trust AI systems?* For us the answer is no, at least not in their current form.

AI systems make mistakes. Sometimes they are anecdotal, as in the context of games, but some mistakes can have a consequential impact on critical applications such as healthcare and banking. Although the media alert the public about the dangers of a so-called "strong AI," they should highlight the weaknesses of the existing AI systems and the serious damage that can result from these systems when they malfunction. We present a set of methods for building a new generation of robust AI systems. Before discussing the scope of our book and the nature of our robustness methods, we first discuss the very meaning of AI and the main sources of fragility in existing AI-based technologies.

In short, we conceive AI as the ability of a computer system to solve a problem that humans thought that only they could solve. For example, the system that first beat Garry Kasparov in Chess was called "AI." Several decades later, we stopped calling that system AI, but we then reserved this term for the one capable of beating Lee Sedol in Go. Today, our attention is focused on conversational agents such as ChatGPT that many consider as an ultimate form of AI. Essentially, whenever we are surprised by the "intellectual skills" of a machine, we call it AI. When the surprise effect passes, we stop calling it so. Clearly, the skills we call AI have changed over time and, maybe not surprisingly, the computer systems supporting them have also changed. However, recent AI systems have a common denominator: They make

use of machine learning algorithms, where the computing system learns from data (observations) in order to adjust its behavior. In fact, the availability of huge amounts of data has been the key enabler of modern AI-based technologies.

This dependency on data, however, is also the Achille's heel of AI systems. The data, which can come from a wide variety of sources, are not always trustworthy. Some sources can provide erroneous or corrupted data. With current machine learning algorithms, a single "bad" source can lead the entire learning scheme to make important mistakes. Currently, to handle the huge amounts of data available, machine learning algorithms are often deployed over a large number of computing machines. Consequently, as this number increases, the likelihood of machine errors also increases. Indeed, software bugs are prevalent and hardware faults also occur. Furthermore, machines can sometimes be hacked by malicious players, either directly or indirectly through viruses. Some of these players attempt to corrupt the entire learning procedure, merely for the pleasure of claiming to have destroyed an important system. Others attempt to influence the learning procedure for their own benefit.

Building machine learning schemes that are robust to these events is paramount to transitioning AI from being a mere spectacle capable of momentary feats to a dependable tool with guaranteed safety. In this book, we cover some effective techniques for achieving such robustness. Essentially, we present machine learning algorithms that do not trust any individual data source or computing units. We assume that a majority of data sources and machines are trustworthy; otherwise, no correct learning can be ensured. But, we consider that a fraction of those data sources and machines can be faulty. The technical challenge is due to the fact that we consider that, a priori, these adversarial data sources and computing units are indistinguishable from correct ones. We contrast the robust methods that can withstand various types of faults with classic methods that can be corrupted as soon as either a data source is poisoned or a computing unit is hacked. From a high-level perspective, we show how to go incrementally from a traditional learning technique, which is inherently fragile, to new forms of learning based on robust methods.

Purpose of the Book and Target Audience

This book is intended for students, researchers, and practitioners interested in AI systems in general, and in machine learning schemes in particular. The book requires certain basic prerequisites in linear algebra, calculus, and probability. Some understanding of computer architectures and networking infrastructures would be helpful. Yet, the book is written with undergraduate students in mind, for those without advanced knowledge in mathematics or computer science. In short, the purpose of this book is threefold. First, we seek to offer a comprehensive framework for practitioners to be able to implement safe AI systems that use robust machine learning algorithms. Second, we intend for this book to serve as the foundation for a curriculum that could be taught to undergraduate students interested in studying the

next-generation machine learning techniques that should clearly be more robust than the current generation. Lastly, we want this book to also serve as a corpus of results that can be extended and enhanced with new theoretical findings, thus advancing the field of safe AI. The content of this manuscript is based in part on research papers that have been peer-reviewed in major machine learning conferences. We have been working on these methods for over 7 years, from 2016 to 2023. They are inspired by distributed computing techniques that have been explored for over 30 years. Naturally, some of our work was influenced by adjoining research that was published by other groups in related fields. Proper citations of their work are provided at the end of each chapter.

Acknowledgments

We are very grateful to the members of the Distributed Computing Laboratory at EPFL: graduate students, postdoctoral researchers, and lab administrators. We thank current and former members of our lab at EPFL who have been working with us on the topic of the book, namely, Youssef Allouah, Peva Blanchard, El Mahdi El Mhamdi, Sadegh Farhadkhani, Arsany Guirguis, Lê Nguyên Hoang, John Stephan, Geovani Rizk, Sébastien Rouault, and Julien Steiner. They contributed to many of the results we present in the manuscript. We would also like to thank other members of our lab who work in other related areas: Clement Burgel, Martina Camaioni, Abdellah El Mrini, Ahmed Jellouli, Jovan Komatovic, Anastasiia Kucherenko, Matteo Monti, Antoine Murat, Diana Petrescu, Pierre-Louis Roman, Manuel Vidigueira, Gauthier Voron, and Athanasios Xygkis. They provided insightful comments on earlier versions of the manuscript. We thank our colleagues Hagit Attiya, Francis Bach, Waheed U. Bajwa, Marco Canini, Thinh Doan, Anne-Marie Kermarrec, Andrei Kucharavy, Ji Liu, and Nitin H. Vaidya for providing constructive feedback on an initial draft of the manuscript. We are also grateful to Holly Cogliati-Bauereis, whose writing tips and corrections have significantly improved the general writing of the manuscript. Last but not least, we thank France Faille, whose expert administrative support has enabled us to uninterruptedly focus on the research work we present in this manuscript, and Fabien Salvi for providing critical system support.

Lausanne, Switzerland — Rachid Guerraoui
Lausanne, Switzerland — Nirupam Gupta
Paris, France — Rafael Pinot
November, 2023

Contents

Acronyms

Although we try to minimize the use of acronyms for the sake of readability, the following acronyms are often used in the text.

AI	Artificial Intelligence
CWMed	Coordinate-Wise Median
CWTM	Coordinate-Wise Trimmed Mean
DMGD	Distributed Mini-batch GD
DMHB	Distributed Mini-batch Heavy Ball
DSGD	Distributed SGD
ERM	Empirical Risk Minimization
GD	Gradient Descent
GeoMed	Geometric Median
NNM	Nearest-Neighbor Mixing
SGD	Stochastic Gradient Descent

Notation

We use bold lowercase to denote vectors and functions with multi-dimensional outputs and standard lowercase letters to denote scalars and real-valued functions. Depending on the context, we use either a calligraphic font or uppercase letters to denote ensembles: most of the time calligraphic font, sometimes uppercase letters to denote subsets or elements of a set of sets. The cardinality of a finite set S is denoted by $|S|$. For a positive integer N, we denote by $[N]$ the set $\{1, \ldots, N\}$. The set of real values and d-dimensional real-valued vectors are denoted by $\mathbb{R}$ and $\mathbb{R}^d$, respectively. Furthermore, we denote by $\mathbb{R}^+$ and $\mathbb{R}^+_*$ the sets of non-negative and strictly positive real values, respectively. For a real value $v \in \mathbb{R}$, $|v|$ denotes its absolute value, and $\lceil v \rceil$ denotes the smallest integer greater than or equal to v. For a vector $\mathbf{v} \in \mathbb{R}^d$, we denote its k-th coordinate (or element) by $[\mathbf{v}]_k$ for all $k \in [d]$ and its transpose by $\mathbf{v}^\intercal$. The inner product of two vectors $\mathbf{v_1}, \mathbf{v_2} \in \mathbb{R}^d$ is denoted by $\langle \mathbf{v_1}, \mathbf{v_2} \rangle \coloneqq \mathbf{v_1}^\intercal \mathbf{v_2}$. The Euclidean norm of a vector $\mathbf{v} \in \mathbb{R}^d$ is denoted by $\|\mathbf{v}\| \coloneqq \sqrt{\langle \mathbf{v}, \mathbf{v} \rangle}$.

For a (differentiable) real-valued function $\mathrm{L} : \mathbb{R}^d \to \mathbb{R}$, its gradient at any point $\boldsymbol{\theta} \in \mathbb{R}^d$ is denoted by $\nabla \mathrm{L}(\boldsymbol{\theta})$. Whenever necessary, we write $\nabla_{\boldsymbol{\theta}}$ to specify that the gradient is with respect to the argument $\boldsymbol{\theta}$. For a function $\phi : \mathbb{R}^+ \to \mathbb{R}^+$, we use $\mathrm{O}_a(\cdot)$ to denote the Bachmann–Landau notation for describing the infinite behavior of ϕ. Specifically, we write $\phi(a) \in \mathrm{O}_a(\psi(a))$ if $\lim_{a \to \infty} \frac{\phi(a)}{\psi(a)} < \infty$. We extend $\mathrm{O}_a(\cdot)$ to $\mathrm{O}_{a_1, a_2}(\cdot)$ for describing the asymptotic behavior when both a_1 and a_2 approach infinity. Lastly, we use the notation $\mathrm{O}_\star(\cdot)$ and $\Omega_\star(\cdot)$ to describe behavior of ϕ up to a multiplicative constant. Specifically, we write $\phi(a) \in \mathrm{O}_\star(\psi(a))$ if there exists a constant $c \in \mathbb{R}^+$ such that $\phi(a) \leq c\psi(a)$ for all $a \in \mathbb{R}^+$, and $\phi(a) \in \Omega_\star(\psi(a))$ if there exists constant c such that $\phi(a) \geq c\psi(a)$ for all $a \in \mathbb{R}^+$.

Chapter 1
Context and Motivation

Abstract The history of *artificial intelligence* is deeply related to that of *computer science*, in general, and *machine learning*, in particular. In this chapter, we give a perspective of the salient points of this history, before discussing more precisely the very meaning of computers *learning* to become *intelligent*. We then highlight the fragility of current machine learning schemes and the dangerous implications of this fragility on many critical applications, before discussing the meaning of making such learning *robust*. We conclude the chapter by reviewing the content of the book and discussing the intended audience.

1.1 Brief History of Artificial Intelligence

1.1.1 Can Machines Think?

Alan Turing asked this question in 1950, only a few years after inventing his universal machine (that is the model of our modern computer). Turing defined the very meaning of *thinking*, through his famous *intelligence test*. Roughly speaking, a machine passes the test if a human exchanging text messages with the machine cannot tell whether the responses obtained come from the machine or from another human. Interestingly, Turing predicted in those days that, at the end of the (twentieth) century, "one will be able to speak of machines thinking without expecting to be contradicted." Clearly, Turing was too optimistic. But only about the timeline; he was off by only a few decades. In 2023, conversational agents often pass the Turing test.

Around the period of Turing's work, McCulloch, Pitts, and Wiener proposed the first *intelligent* cybernetic system, seeking to reproduce the functioning of human intelligence. Their goal was to build a description of human perceptions by using electrical circuits and to simulate brain mechanisms based on a formal model. Their model paved the way for the idea of a neural network. Concomitantly, Claude Shannon proposed a remote-controlled mouse that was able to find its way out of a labyrinth while *remembering* its course.

R. Guerraoui et al., *Robust Machine Learning*, Machine Learning: Foundations, Methodologies, and Applications, https://doi.org/10.1007/978-981-97-0688-4_1

In 1956, John McCarthy and Marvin Minsky hosted a historical meeting: the Dartmouth Summer Research Project on Artificial Intelligence (DSRPAI). The goal was to bring together top researchers from various fields for an open-ended discussion on *Artificial Intelligence (AI)*; the term AI was coined by McCarthy at that very event. However, there has never been any agreed definition of the term AI, and the event did not produce any concrete results. Yet, this event somehow launched decades of activities in various kinds of research called AI. In particular, a year later, the first neural network, called the *perceptron*, was designed by Frank Rosenblatt.

What Is AI, Anyway?
We define AI as *the ability of a machine to solve a problem that humans believed they were the only ones capable of solving*. In other words, we talk about AI whenever a machine surprises us by solving a specific task. However, after the surprise effect, we cease using this term (at least regarding the specific task). This is the case, for example, with the game of chess. When IBM's Deep Blue beat Garry Kasparov, the then reigning world champion, we called that machine an AI. Today, even a smartphone is capable of beating all chess world champions, and we do not talk about AI anymore for chess. Then came the game of Go, arguably more challenging than chess due to the huge number of possible combinations. Again, a machine was eventually able to beat the world champion. We talked about AI again, but not for long.

Similarly, a decade ago, no machine was comparable to a human in language or image recognition. Today (in 2023), machines arguably compete with humans in both these domains. Machines have also become much better at generating images and producing language. Currently, e-mails are completed, online texts are translated, videos are transcribed, reports are generated, and media outlets are published in an automatic manner. Furthermore, recommender systems determine the content we see on social media and even the products we buy from online shops. With their capacity to generate images and texts, machines are likely creating a large fraction of the media we consume.

Although that which we call AI has evolved over the years, the pragmatic purpose remains unchanged: surpassing human capabilities in cognitive tasks such as vision, natural language processing, knowledge representation, and reasoning. The initial idealistic goal of replicating brain functions has been, however, largely supplanted by data-driven methodologies, the so-called *machine learning*.

1.1.2 Machine Learning

The idea of a machine that learns is as old as computer science is. It dates back to the days of Alan Turing. Indeed, he conjectured that *learning* is the key to passing his *intelligence test*. In short, Turing explained that it is physically impossible for a machine to a priori store (in its memory) all possible cases that it might need to deal

with in order to perform a challenging task. A more reasonable alternative is to have a machine learn from its experience and to adapt. In fact, some form of *learning* was already at the heart of his universal machine, as we explain below.

Before the *universal Turing machine*, invented in 1936, there were only specialized calculators wherein the algorithm was hardwired into the machine. For example, Pascal's machine was specialized in performing additions, while Leibniz's machine was specialized in multiplications. Although these machines could compute their respective operations faster than any human, their utility was limited to the operation they were hardwired for, i.e., addition or multiplication. There was no way for Pascal's machine to learn multiplication, or for Leibniz's machine to learn other operations. Whereas, although more slowly than Pascal's and Leibniz's machines, a human could learn the addition algorithm, then the multiplication algorithm (or any other one) eventually performs a wide range of operations. Humans in this sense are *smarter* than machines because of their ability to *learn* new algorithms.

The very characteristic of a *universal Turing machine* that makes it *smart* is also its ability to *learn* new algorithms. Specifically, Turing hardwired into his machine a meta-algorithm that executes the inputs and outputs of algorithms provided to it as *data*. Data and algorithms essentially become the same thing. The addition and multiplication algorithms become *inputs* represented in the form of a *table* that can be executed by a Turing machine. It was conjectured that all the algorithms can be modeled in the form of a *table* that could be then executed by a Turing machine. Until now, no one has disproved this conjecture. Turing did not witness the impact of his universal machine, but many engineers continued his seminal work and perfected the development of his machine, until it became the modern computer.

A Simple Yet Powerful Principle

Interestingly, Turing also anticipated *automatized* forms of learning that goes beyond having humans write programs in the form of data (tables) that the machine could digest (learn) and then execute. In 1947, Turing gave a public lecture explicitly calling for "a machine that can learn from experience" with the "possibility of letting the machine alter its own instructions." In a report entitled "Intelligent Machinery," which he did not publish, Turing introduced many of the central concepts that today we call AI. Many of his ideas were later reinvented by others, such as the idea of training a network of artificial neurons to perform specific tasks. Essentially, Turing predicted that the learning process would not need human participation. Specifically, the *data* (representing an algorithm) would not be fed to a Turing machine by a human as a *table* (i.e., an executable program), rather the data could be absorbed by the machine through a *learning program*.

The notion of a learning program is the driving principle behind modern-day *machine learning*. Essentially, machine learning schemes assume a generic *model* to solve the target task, typically based on a mathematical function, and it deduces patterns in the available data to make predictions on a priori unseen data. The initial model is said to be generic in the sense that it has *unspecified parameters*, e.g., a function $a_1x + a_2$ where the *parameters* a_1 and a_2 are a priori *unspecified*. Deducing patterns, which is commonly referred to as *training*, is a procedure

that consists in computing model *parameters* that optimize a pre-defined *loss function*, i.e., to minimize the number of mistakes when applying predictions on the data at hand. Model training is often achieved through means of an optimization algorithm, such as a *gradient-descent method* that searches efficiently through all the possible parameter values. In the case of natural language processing, machine learning models, by training on several sentences and combinations of words, parse previously unseen sentences or combinations of words to predict a new set of words or sentences. In the case of image recognition, machine learning models are trained to recognize objects, such as cars or dogs, by training on several images with and without the objects. The model can be viewed as the *hypothesis* and the parameters as the *tailoring of the hypothesis* to a specific task.

1.1.3 Applications of Machine Learning

Turing speculated that computers would one day play chess. He designed a program capable of playing an entire game against a human. The program assigns values to each game state and selects the move with the highest value. However, the program was too complex to be run by the early computers of Turing's time. Turing executed his program manually against a colleague of his in 1952. Just over 50 years later, in 1997, Deep Blue, a chess computer designed at Carnegie Melon University and built by IBM, beat the then reigning world champion, Garry Kasparov. Deep Blue's 256 parallel processors enabled it to examine 200 million possible moves per second and to look ahead to as many as 14 turns of play.

Learning Games

Fourteen years later, in 2011, another IBM computer called Watson, 100 times faster than Deep Blue, won the game Jeopardy, against the TV quiz show's two greatest all-time champions. Watson had 2800 microprocessors and could calculate up to 80 trillion operations per second. More than 100 parallel algorithms analyzed each of the questions asked to Watson and found different plausible answers, while another set of algorithms ranked the answers. The very fact that a machine could win at Jeopardy attracted much attention. For many observers, while chess was considered the iconic representation of *human intelligence*, Jeopardy was considered the iconic representation of *being human*. In addition to the architectural difference between DeepBlue and Watson, there was an important difference between the underlying programs, thus illustrating the evolution of the concept of *machine learning*. On the one hand, the engineers of the chess algorithm relied on the so-called *symbolic AI*: a set of rules designed by chess champions and executed by a specific program called an *expert system*. The engineers of the Jeopardy algorithm, on the other hand, relied on a massive amount of *unstructured data* (the equivalent of one million books) representing natural language patterns.

Four years later, in 2015, another machine, AlphaGo, developed by DeepMind Technologies, was able to beat the strongest players in the board game Go. The

comparison with chess is interesting to note. The legal number of chess positions is around 10^{40}, whereas it is over 10^{170} in Go, which is much more than the 10^{80} atoms in the observable universe. In fact, if every known atom in the universe were itself a universe containing within it 10^{80} atoms of its own, the number of total atoms would still fall short of the possible legal positions in Go. As pointed out earlier, Deep Blue followed human-programmed rules by assigning values to possible outcomes of different moves, ranking them based on the advantages they would bring to the machine. Go was too complex to explore different possibilities. It needed a completely different approach. AlphaGo deduced rules by heavily relying on a *reinforcement learning* policy applied to millions of games where the machine basically played against itself. Unlike Deep Blue or Watson, AlphaGo *learned to get better by itself*. This made it very difficult to interpret some of its moves, which were surprising to the game experts, but this was revealed to be very successful.

A few years later, in 2019, poker and bridge also fell into the hands of machine learning. This also came as a surprise. Microsoft founder Bill Gates, a keen player, considered bridge to be well-protected from this trend. Unlike chess and Go, most of the information, especially the cards, is not open and available on the board, but hidden from view. A French company, NukkAI, could beat the best bridge players by using a neuro-symbolic approach that combines two very different paths to learning. The first path makes essentially random choices, following AlphaGo's strategy. The second uses rules, as does Watson. A similar combination, involving random choices and searches, enabled Pluribus, a robot developed by Facebook and Carnegie Melon University, to beat several poker champions in Las Vegas. The robot not only learned to bluff but could also see through bluffs. Surprisingly, Pluribus has learned to deal with liars, and to lie itself, a skill attributed to humans alone, until now.

Applications in Daily Life

The aforementioned intellectually challenging tasks, namely chess, Jeopardy, Go, bridge, and poker, made it possible to experiment with machine learning techniques and to illustrate their power. But they were only "games." Other, more critical, challenges were also considered. Machine learning techniques have indeed been successful in various domains, such as traffic prediction (e.g., on Google Maps), text analysis (e.g., spam e-mail filters), banking/finance (e.g., credit decisions), social networking (e.g., recognizing pictures and content recommendations), and personal assistance (e.g., Siri and ChatGPT). Furthermore, machine learning is foreseen as being the motor of important advances in healthcare, e.g., for the identification and diagnosis, through computer vision, of diseases such as skin cancers that are difficult to detect during the initial stages. Personalized disease prevention and treatment using predictive analytics is also a significant possible application. In the same vein, the large amount of data collected from satellites, real-time social-media updates, website information, etc., could be used by sophisticated machine learning systems to predict epidemics around the world.

Last but not least, many expect machine learning schemes to be a principled way for predicting natural disasters and for finding pragmatic approaches to deal with global warming. Virtually every domain of science and engineering can benefit

from machine learning schemes, and the current rate at which this technology is being adopted in our daily lives reflects widespread confidence in its potential.

1.2 Robust Machine Learning

1.2.1 Fragility of Machine Learning

The quality of a machine learning scheme is directly proportional to the quality of its training. Modern machine learning schemes make use of a massive amount of (a) data and (b) computing resources. Consider, for instance, ChatGPT-3.5, one of the most popular *deep-learning* schemes for language processing. ChatGPT-3.5 used *10,000 Nvidia graphics processing units (GPUs)* to train a neural network, in early 2022, with approximately *175 billion parameters* on a massive corpus of text data (approximately *570GB of data*), including web pages, books, and other sources. The volume of training data and the size of the models have been growing ever since. Training models of such unprecedented scale has been made possible due to *distributed computing* that involves a collective effort from multiple machines. The computations are distributed across several machines in the same data center and sometimes across remotely located machines connected through sophisticated communication protocols. Models can also be trained on a diffuse network of edge devices that perform local updates on their local data. In order to obtain *global model updates*, each machine performs *independent model updates* that are periodically averaged through means of a central server. While the distribution of the training procedure makes it feasible to develop complex models and promises high accuracy, it also makes these modern machine learning schemes *fragile*. In fact, because the data sources come from multiple origins, typically on the Web, they are prone to corruption. Dishonest players can, for instance, inject corrupted data that can be collected by data harvesting bots, or they can send corrupted data to an aggregator, such as a chatbot, a spam filter, or a database of user profiles. Poor-quality information produces subpar results, regardless of the sophistication level of the model architecture. History shows that it does not take much to do this. In fact, the fragility can come not only from bad data but also from malfunctioning machines. When involving a large number of machines for model training, the probability that some of them suffer software bugs or hardware errors is also very high. Their geographic distribution can make some of them the target of specific attacks.

Many now classic machine learning schemes rely on *averaging information* from different data sources and machines. In this case, it suffices that one source is bad for the entire learning procedure to be corrupted. Dishonest players can easily have complete control over the training data seen by their device, in addition to control over the local model updates sent to a central server. Even if the entire training is done on a single machine, dishonest players can infiltrate the training database by

inserting incorrect or misleading information by hacking data sources. If the model is trained on corrupted data, it will draw unintended and even harmful conclusions. There are several examples of machine learning systems that caused dangerous errors. In 2019, when a deep network was trained on the "person" categories of the ImageNet dataset for classifying faces into different categories, as part of the ImageNet Roulette project, the model quickly started using racial and gender slurs to label people. In 2018, IBM's Watson proposed dangerous treatments to cancer patients. In 2017, in Hong Kong, UK-based Tyndaris Investments provoked the loss of 20 million dollars per day. In short, ensuring the robustness of machine learning systems is paramount to protecting our own lives.[1]

A fundamental difference between ensuring the robustness of machine learning schemes and classic computer systems is interesting to highlight. When used in critical applications, traditional computer systems are isolated from the outside world by using firewalls, passwords, data encryption, and other access-control measures. But machine learning systems are data hungry, and their success lies in their being open: They need to harvest information from various data sources, sometimes including anonymous and unverified sources on the Web. Also, modern machine learning schemes need a large number of computing machines to perform independent computations, which increases the probability that some of them will malfunction.

1.2.2 The Quest for Robust Machine Learning

Robustness, in computer science, generally refers to the ability of a software-reliant system to continue working over a long period of time and to do so consistently with its specifications, despite the presence of internal and external failures (e.g., faulty inputs). Although different forms of robustness in the context of machine learning could be considered, in this book we consider the specific aspect of *model training*. In particular, by robustness of a machine learning system, we refer to the ability of training accurate models—despite bad data and malfunctioning machines. We focus specifically on the optimization procedure that training a machine learning model encompasses. We present a set of methods that make this optimization procedure *reliable*, despite *unreliable* data sources and computing machines.

Here, a question is in order: *How can we build a robust machine learning scheme if we cannot trust either the data or the machines?* Clearly, if the entire data are bad and all the machines are corrupted, then no learning can be achieved. We assume that a fraction of the data is good and that a fraction of the machines function correctly. But, unlike most work in classic machine learning, we consider

[1] These anomalies are not due to malicious intent, but due to poor training. Nevertheless, they illustrate just how dangerous a mistake can be when we carelessly apply machine learning models to real-life situations.

a scenario where a fraction of the data can be bad and a fraction of the machines can malfunction. We consider a general framework that encompasses many modern machine learning schemes: the so-called *server-based* architecture. Essentially, a single machine (i.e., the server) is given the responsibility of maintaining the model. The server updates the parameters of the model by using information provided by other machines known as *nodes*. These node machines host the data. Periodically, in order to provide new information for improving the model, the server broadcasts the current parameters of the model to all the nodes so that each node can test the model on its data. This methodology is sometimes referred to as (server-based) *federated learning*. We present, in this book, techniques for ensuring that the optimization procedure run by the server succeeds, even if some of the nodes do not function properly.

How to Know If the Training Succeeded?
Another fundamental question is in order: *What is a successful training procedure if some of the nodes are corrupted, e.g., their data are incorrect?* Intuitively, an accurate model is one that is generated based on only the fraction of correct machines. Then, a robust training method will be considered as successful if it manages to ignore the incorrect machines and data. Of course, this task would be easy if we know which machines are incorrect in advance. The server would simply disregard the information they provide. The challenge is that, in practice, all the machines initially function correctly, until some unknown time beyond which some machines would begin deviating from their correct behavior. Furthermore, this deviation might not be easy to detect, as the server would not be able to distinguish the differences from the information provided by the node, which are due to heterogeneous data, from differences that are due to the misbehaving nodes.

The above discussion brings us to the next question: *What does it really mean for a machine to malfunction?* To achieve genuine robustness, we consider that the notion of a malfunction can be very general. In short, we assume that some of the machines (nodes) can return arbitrarily bad information to the server. In the parlance of distributed computing, these machines can be called *Byzantine*. This way of modeling the misbehavior of some nodes is general, in the sense that it encompasses many cases: corrupted data at the node level, a software or hardware bug, and even the situation where the machine has been hacked by a malicious party. In fact, it is convenient to view the design of a robust machine learning technique as a *game* against an adversary that controls a fraction of the machines and their data. Machines controlled by the adversary are called *adversarial* nodes. This adversary is free to choose the best possible strategy, and a robust machine learning scheme should strive for an accurate model regardless of the adversary's strategy.

Remark 1.1 Although the terminology and general adversarial (i.e., Byzantine) abstraction we consider are inspired by classic distributed computing, the settings are different. In classic distributed computing, it is typical to ensure the robustness of an application by *replicating its state* on several machines. The clients of the service then use some *consensus*-based communication primitive to update, in the same order, the state of the application on the correct replicas, and they return a

correct response. Machine learning is a very specific application. Essentially, state replication across several machines is not feasible. The machines (nodes) in machine learning host different data and, in many cases, they do not seek to share their local data due to privacy and ownership issues.

A Simple Yet General Framework
The framework we consider might seem very simple: We assume a *trusted* server that interacts with several nodes to learn a model on their data by using an optimization algorithm. This choice is deliberate for pedagogical reasons. However, it is without loss of generality. Specifically, though we consider a single server that is not adversarial, the presented techniques for achieving robustness to adversarial nodes could be extended to a case with several servers, a fraction of which can also be adversarial. In fact, these robust machine learning schemes can be extended to a completely decentralized peer-to-peer setting, where a node can either play the role of a node or a server. We also consider *mini-batch gradient descent* (mini-batch GD) to be the primary optimization algorithm of our study, as current machine learning schemes are predominantly based upon first-order methods that build on top of mini-batch GDs. Hence, ensuring the robustness of this gradient-descent method constitutes a first step toward ensuring the robustness of machine learning schemes that use other optimization algorithms.

1.3 Content of the Book

1.3.1 Summary

We present fundamental techniques for robustness to adversarial nodes in federated machine learning. We begin progressively by first explaining that classic machine learning algorithms are not robust to even a single adversarial node. Essentially, the linearity of classic aggregation schemes (e.g., averaging) makes them vulnerable against a few nodes that deviate from the prescribed optimization procedure. We introduce ways for tolerating a fraction of adversarial nodes by using *non-linear robust aggregation* schemes that satisfy the critical property of *robust averaging*. Specifically, upon using these *robust aggregation* schemes, a standard machine learning system can attain an optimal *breakdown point* that is the largest minimum fraction of adversarial nodes that renders the machine learning scheme vacuous. These aggregation schemes, however, yield suboptimal *training errors*. To address this shortcoming, we present the notion of *pre-aggregation*: a clustering of vectors whose composition with most robust aggregation schemes yields *order-optimal training error*. However, the resulting robust machine learning schemes can be *impractical*. Specifically, their *gradient complexity*, i.e., the number of gradients computed by the honest (non-adversarial) nodes during the training procedure, is orders of magnitude higher than that of the classic (non-robust) machine learning algorithms. This is mainly due to the retention of gradient noise (an artifact of data

sub-sampling) in the training error when dealing with adversarial nodes. Lastly, we show that the use of *Polyak's momentum* in the local computations at the nodes' end diminishes the effect of gradient noise in the training error, thereby conferring more practicality to robust machine learning schemes. We highlight the interaction between the three essential elements of robust machine learning: *breakdown point*, *training error*, and *gradient complexity*. Note that, in addition to discussing robustness, in this book, we also present pedagogical analyses of the convergence of gradient-descent algorithms, which arguably form the backbone of machine learning, in both (strongly) convex and non-convex regimes. Beyond standard convergence techniques, in the latter part of the book, we also present the technique of Lyapunov analysis for studying convergence. All the proofs presented are self-contained and present a complete reasoning behind the results discussed throughout the book.

Remark 1.2 This book does not address the question of *privacy*: the way a server can learn an accurate model while ensuring that the data held by each node remain private. We also do not address notions of robustness that do not pertain to the optimization procedure of the training phase, such as *evasion attacks* (a.k.a. robustness to *adversarial examples*) or robustness to distribution shift. These problems, although primary in studying the safety of AI systems, are (in general) orthogonal to the problem that we consider.

1.3.2 Organization

In Chap. 2, we recall the basics of machine learning. We focus on presenting a clear problem statement. We introduce fundamental optimization tools that are needed to understand the robustness issue in the training phase. Notably, we recall relevant notions such as supervised machine learning, empirical risk minimization, surrogate loss functions (stochastic/mini-batch), gradient-descent methods, and gradient complexity.

In Chap. 3, we explain our (server-based) machine learning framework and introduce a few distributed gradient-descent methods. Specifically, we explain the underlying principle of distributed (stochastic) gradient descent by analyzing in detail the cases of non-convex, as well as strongly convex training loss functions. We derive gradient complexities in both cases and present mathematical tools that lay the foundation for the subsequent chapters.

Chapter 4 is a primer on our concept of robustness in machine learning. We first highlight the fragility of current machine learning schemes, before presenting the central notion of robust aggregation. In the chapter, we introduce a precise metric for robustness, through the complementary notions of *breakdown point*, *training error*, and *gradient complexity*. These notions are then illustrated through a robust variant of distributed mini-batch gradient descent that replaces the linear aggregation at the server (i.e., averaging) with a non-linear robust aggregation rule. For ease

of presentation and pedagogical purposes, we make some simplifications in this chapter. For this reason, the presented algorithm could be considered impractical. We revisit these simplifications in the following chapters.

In Chap. 5, we focus on minimizing the training error in the presence of adversarial machines. We first discuss the potential limitation of robustness due to *gradient dissimilarity*, an artifact of *heterogeneity* in data held by the nodes. We then show that many robust aggregation rules, though they can provide the optimal breakdown point, yield a suboptimal training error under gradient dissimilarity. We present the technique of *pre-aggregation* that circumvents this shortcoming and achieves optimality.

In Chap. 6, we focus on gradient complexity in robust machine learning. Although a composition of a robust aggregation rule with pre-aggregation yields order-optimal training errors, the resulting gradient complexity can be orders of magnitude higher than a traditional (non-robust) machine learning algorithm. This could greatly discourage the use of these robustness schemes. We highlight the important role of *gradient momentum at local level* in regulating the gradient complexity, thereby rendering robust machine learning more practical.

1.4 Chapter Notes

In this chapter, we have briefly recalled the evolution of machine learning: from the idea of artificial intelligence [16, 25, 26] to building machines that can play cognitive games [2, 9, 12, 24] to utilizing machines in healthcare [6, 10]. Modern machine learning systems, such as ChatGPT [3], derive their power from large-scale complex models that are trained over vast amounts of data. These automated systems are critically shaping the course of modern society, with applications ranging from healthcare to military [7, 11, 15, 18, 20, 22]. There is no denying that machine learning is rapidly transforming the realm of information technology, touching almost every aspect of our lives[14]. Although these changes could significantly improve the quality of life in general and might even help us find ways to fight against emerging global issues, such as climate change [21], they should be used with a great amount of caution.

We have highlighted that the classic machine learning algorithms, especially those proposed for developing large-scale models, are not robust to the challenges prevalent in real-world scenarios, specifically the element of *corruption*. It has been shown that the effectiveness of machine learning schemes can become highly compromised, when either the training data are adulterated or when some machines malfunction during the training procedure [1, 4, 5, 8, 13, 23, 27]. Using such *poorly trained* models in public-oriented applications could have severe consequences [17, 19]. Protecting machine learning systems against corrupted data and malfunctioning machines, thereby paving the path to *safe AI*, was our key motivation in developing the *robust machine learning* methods we present in this book.

References

1. Baruch M, Baruch G, Goldberg Y (2019) A little is enough: Circumventing defenses for distributed learning. In: Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, 8–14 December 2019, Long Beach, CA, USA
2. Brown N, Sandholm T (2019) Superhuman AI for multiplayer poker. Science 365(6456):885–890
3. Brown T, Mann B, Ryder N, Subbiah M, Kaplan JD, Dhariwal P, Neelakantan A, Shyam P, Sastry G, Askell A et al (2020) Language models are few-shot learners. In: Advances in Neural Information Processing Systems, vol 33, pp 1877–1901
4. Cao X, Jia J, Zhang Z, Gong NZ (2022) FedRecover: recovering from poisoning attacks in federated learning using historical information. In: 2023 IEEE Symposium on Security and Privacy (SP). IEEE Computer Society, pp 326–343
5. Carlini N, Terzis A (2022) Poisoning and backdooring contrastive learning. In: International Conference on Learning Representations
6. Cuocolo R, Caruso M, Perillo T, Ugga L, Petretta M (2020) Machine learning in oncology: a clinical appraisal. Cancer Lett 481:55–62
7. Donepudi PK (2017) AI and machine learning in banking: a systematic literature review. Asian J Appl Sci Eng 6(3):157–162
8. Fang M, Cao X, Jia J, Gong N (2020) Local model poisoning attacks to Byzantine-Robust federated learning. In: 29th USENIX Security Symposium (USENIX Security 20), pp 1605–1622
9. Ferrucci D, Brown E, Chu-Carroll J, Fan J, Gondek D, Kalyanpur AA, Lally A, Murdock JW, Nyberg E, Prager J et al (2010) Building Watson: An overview of the DeepQA project. AI Mag 31(3):59–79
10. Ferrucci D, Levas A, Bagchi S, Gondek D, Mueller ET (2013) Watson: beyond jeopardy! Artif Intell 199:93–105
11. Hoijtink M, Planqué-van Hardeveld A (2022) Machine learning and the platformization of the military: a study of Google's machine learning platform TensorFlow. Int Polit Sociol 16(2):olab036
12. Hsu F.-H. (2002) Behind deep blue: building the computer that defeated the world chess champion. Princeton University Press, Princeton
13. Kairouz P, McMahan HB, Avent B, Bellet A, Bennis M, Bhagoji AN, Bonawitz K, Charles Z, Cormode G, Cummings R et al (2021) Advances and open problems in federated learning. Found Trends®Mach Learn 14(1–2):1–210
14. Kaplan A, Haenlein M (2019) Siri, Siri, in my hand: who's the fairest in the land? On the interpretations, illustrations, and implications of artificial intelligence. Business Horizons 62(1):15–25
15. Leo M, Sharma S, Maddulety K (2019) Machine learning in banking risk management: a literature review. Risks 7(1):29
16. McCulloch WS, Pitts W (1943) A logical calculus of the ideas immanent in nervous activity. Bull Math Biophys 5(4):115–133
17. Nakagawa K, Moukheiber L, Celi L, Patel M, Mahmood F, Gondim D, Hogarth M, Levenson R (2023) AI in pathology: what could possibly go wrong? In: Seminars in diagnostic pathology. Elsevier, Amsterdam
18. Namiot D, Ilyushin E, Chizhov I (2021) Military applications of machine learning. Int J Open Inf Technol 10(1):69–76
19. Neri E, Coppola F, Miele V, Bibbolino C,Grassi R (2020) Artificial intelligence: who is responsible for the diagnosis? Radiol Med 125:517–521

20. Pati S, Baid U, Edwards B, Sheller M, WangS.-H., Reina GA, Foley P, Gruzdev A, Karkada D, Davatzikos C et al (2022) Federated learning enables big data for rare cancer boundary detection. Nat Commun 13(1):7346
21. Rolnick D, Donti PL, Kaack LH, Kochanski K, Lacoste A, Sankaran K, Ross AS, Milojevic-Dupont N, Jaques N, Waldman-Brown A et al (2022) Tackling climate change with machine learning. ACM Comput Surv 55(2):1–96
22. Shailaja K, Seetharamulu B, Jabbar M (2018) Machine learning in healthcare: a review. In: 2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA). IEEE, pp 910–914
23. Shejwalkar V, Houmansadr A (2021) Manipulating the Byzantine: optimizing model poisoning attacks and defenses for federated learning. In: NDSS
24. Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M et al (2016) Mastering the game of Go with deep neural networks and tree search. Nature 529(7587):484–489
25. Turing A (1950) Computing machinery and intelligence. Mind 59(236):433–460
26. Wiener N (1948) Cybernetics; or control and communication in the animal and the machine. Wiley, Hoboken
27. Xie C, Koyejo O, Gupta I (2019) Fall of empires: breaking byzantine-tolerant SGD by inner product manipulation. In: Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22–25, 2019, p 83

Chapter 2
Basics of Machine Learning

Abstract Machine learning consists in designing algorithms that exploit data (sometimes called observations) in order to acquire domain knowledge and perform an automated decision-making task. Contrary to most conventional computing tasks, learning algorithms are *data-dependent* in the sense that they build *task-specific* models and improve upon them using the data fed to them. Machine learning algorithms are mainly classified into four classes: *supervised learning, unsupervised learning, semi-supervised learning*, and *reinforcement learning*. Each of these classes has its own interest and peculiarities. In this book, we focus on supervised classification to illustrate and to formalize the main concepts of robust machine learning. Most of the robustness techniques we discuss however in the book can also be applied to other machine learning classes. In this chapter, we present the fundamentals of supervised learning, through the specifics of the *supervised classification* task, and we review some of the standard optimization algorithms used for solving this task.

2.1 Supervised Classification

In supervised learning, the data (or observations) consist of *input–output* pairs, and the algorithm uses these pairs to build a function, a.k.a., a *predictor*, that maps an *input* to an *output* by identifying a pattern in the observations. This predictor can then be used to provide labels for *unseen* inputs. For example, as illustrated in Fig. 2.1, the predictor can take a picture as an input and tell us whether this image represents an elephant, a chicken, a mouse, or a snake.[1] When the outputs are categorical, the supervised-learning problem is also called *supervised classification*. We formalize this concept below and quickly go over the driving principles behind the selection of a good predictor.

[1] The animal icons we use in this figure are courtesy of the Freepick team. See the following links for the respective icons: Elephant, Chicken, Mouse and Snake.

R. Guerraoui et al., *Robust Machine Learning*, Machine Learning: Foundations, Methodologies, and Applications, https://doi.org/10.1007/978-981-97-0688-4_2

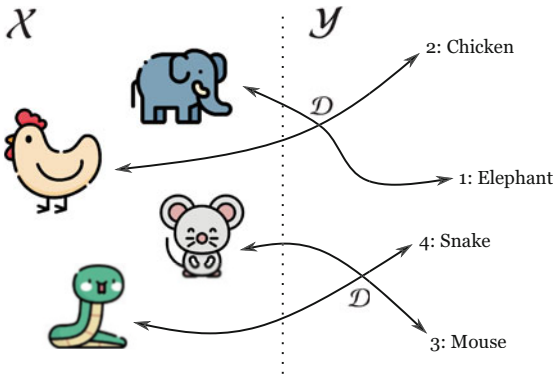

Fig. 2.1 An example of a classification task with a set of images $\mathcal{X}$ and a set of labels $\mathcal{Y}$. The classifier C maps an image $\mathbf{x} \in \mathcal{X}$ to its *most probable* label as per the probability distribution $\mathcal{D}$ defined over the space $\mathcal{X} \times \mathcal{Y}$

2.1.1 Problem Statement

Consider an input space $\mathcal{X}$ and a discrete (finite-sized) output space $\mathcal{Y}$ representing the set of possible labels for elements in $\mathcal{X}$. These spaces are assumed to be associated with a probability distribution $\mathcal{D}$ defined over $\mathcal{X} \times \mathcal{Y}$. This means that an input–output pair $(\mathbf{x}, y) \in \mathcal{X} \times \mathcal{Y}$ can be drawn as per distribution $\mathcal{D}$. A supervised classification problem consists in finding a function $C : \mathcal{X} \to \mathcal{Y}$, called a *classifier*, that reproduces $\mathcal{D}$ as accurately as possible. Specifically, the goal of a supervised-learning algorithm is to produce a classifier $C : \mathcal{X} \to \mathcal{Y}$ that maps each input $\mathbf{x} \in \mathcal{X}$ to its *most probable* output $y \in \mathcal{Y}$ as per the distribution $\mathcal{D}$. Figure 2.1 illustrates an example of such a classification problem in the context of images.

To measure the quality of classifier C, we use the notion of *misclassification error*, defined to be the probability that C does not predict the correct output for a random pair $(\mathbf{x}, y) \sim \mathcal{D}$. Specifically, the misclassification error of C is given by

$$\mathbb{P}_{(\mathbf{x},y)\sim\mathcal{D}}\left[C(\mathbf{x}) \neq y\right] = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}\left[\mathbb{1}\{C(\mathbf{x}) \neq y\}\right] , \tag{2.1}$$

where $\mathbb{1}\{\cdot\}$ is the indicator function. Consider the example in Fig. 2.1. In this example, suppose that a classifier C labels all the images of elephants as chickens but labels the remainder of the images correctly. Then, assuming that an image is drawn uniformly at random from the four possible images, the misclassification error of C equals 0.25. That is, C would correctly classify an image in this case with an unconditional probability of 0.75.

The consequences of a misclassification are interpreted differently according to the application we consider. On the one hand, a predictor that is correct with probability 95%, i.e., with misclassification error of 5%, can be considered as sufficiently accurate to classify animal images. On the other hand, the same misclassification error might be considered extremely unsafe in the context of a medical application, e.g., tumor identification from X-ray imaging, where even a single misclassification can have severe consequences.

Bayes Optimal Classifier

Given a probability distribution $\mathcal{D}$, we can easily show that the optimal prediction function on $\mathcal{X} \times \mathcal{Y}$ is

$$C^{\text{opt}} : \mathbf{x} \mapsto \underset{y \in \mathcal{Y}}{\text{argmax}}\, \mathbb{P}_{(\mathbf{x},y)\sim\mathcal{D}}\left[y \mid \mathbf{x}\right] . \tag{2.2}$$

This function is called the *Bayes optimal classifier*. It is optimal in the sense that no other classifier $C : \mathcal{X} \to \mathcal{Y}$ has a smaller misclassification error on $\mathcal{D}$. While it is possible in theory to find an optimal classifier, a more realistic objective of a machine learning algorithm is to find an *approximately optimal classifier* with a misclassification error that is as close as possible to C^{opt}.

A typical strategy to approximate the Bayes optimal classifier is as follows. First, we assign arbitrary indices to the label set $\mathcal{Y}$, i.e., fix an arbitrary bijection $\Lambda : \mathcal{Y} \to \{1, \ldots, |\mathcal{Y}|\}$. This allows to have for each $y \in \mathcal{Y}$ a unique index $\Lambda(y) \in \{1, \ldots, |\mathcal{Y}|\}$. Then, we define a set of functions $\mathcal{H} \subset \{h : \mathcal{X} \to \mathbb{R}^{|\mathcal{Y}|}\}$, where for every $h \in \mathcal{H}$ and $\mathbf{x}$, $h(\mathbf{x})$ is meant to approximate the probability measure $\mathbb{P}_{(\mathbf{x},y)\sim\mathcal{D}}\left[y \mid \mathbf{x}\right]$. This set is also known as the *hypothesis class*. In this context, the purpose of a machine learning algorithm is to find the hypothesis $h^* \in \mathcal{H}$ that best matches the conditional probability. Then, by analogy with (2.2), the end classifier is defined to be

$$C : \mathbf{x} \mapsto \Lambda^{-1}(z) , \quad \text{where} \quad z \in \underset{k \in \{1,\ldots,|\mathcal{Y}|\}}{\text{argmax}} \left[h^*(\mathbf{x})\right]_k \tag{2.3}$$

and $\left[h^*(\mathbf{x})\right]_k$ is the k-th element of the vector $h^*(\mathbf{x})$

If the hypothesis class $\mathcal{H}$ is sufficiently well-defined, then the classifier C defined above will have a misclassification error close to that of the Bayes optimal classifier. We intentionally avoid explaining here what we mean by "well-defined," as our main focus is on how to optimize over a given set $\mathcal{H}$, i.e., how to find h^*. However, choosing the appropriate hypothesis class $\mathcal{H}$ is non-trivial in general. See Remark 2.1 for more details.

Empirical Risk Minimization

In practice, however, we do not know a priori the ground-truth distribution $\mathcal{D}$. We only have access to a training set $\mathcal{S}$ comprising a finite number, let us say m, of input–output pairs. Specifically, $\mathcal{S} \coloneqq \{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_m}, y_m)\}$, where for all $i \in [m]$, $(\mathbf{x_i}, y_i) \in \mathcal{X} \times \mathcal{Y}$. In the supervised-learning setting, it is assumed that these input–output pairs are drawn independently and identically according to the distribution $\mathcal{D}$. Then, given the hypothesis class $\mathcal{H}$, the learning algorithm produces a hypothesis $h^* \in \mathcal{H}$ that minimizes the *empirical risk* measured on $\mathcal{S}$. This problem, a.k.a. *empirical risk minimization* (ERM), reduces to seeking a solution h^* to the following optimization problem:

$$\min_{h \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^{m} \ell\left(h(\mathbf{x_i}), y_i\right) ,$$

where $\ell : \mathbb{R}^{|\mathcal{Y}|} \times \mathcal{Y} \to \mathbb{R}$ is a *point-wise loss function*. The loss function ℓ measures how well a hypothesis h does on predicting the output y of an input $\mathbf{x}$. The smaller the value of $\ell\left(h(\mathbf{x}), y\right)$ the better h performs on the sample $(\mathbf{x}, y)$. If this loss function is sufficiently well chosen, the average of this loss over $\mathcal{S}$ is an estimate of the *expected loss* that the hypothesis h incurs over the true distribution $\mathcal{D}$. The following section presents some common choices for the hypothesis class and the point-wise loss function.

Remark 2.1 We intentionally ignore the connection between the ERM and the performance of h^* on unseen samples drawn from $\mathcal{D}$. This connection (a.k.a. generalization) lies at the core of supervised learning and is highly related to the selection of the hypothesis class $\mathcal{H}$. This concept is, however, orthogonal to the topic presented in this book that focuses on the optimization part of the machine learning algorithm. For more discussion on why solving the ERM (or a regularized version of the ERM) is essential to finding a hypothesis with good generalization capabilities, we refer the interested reader to Sect. 2.3 and references therein.

2.1.2 Hypothesis Classes and Loss Functions

In principle, we could choose the hypothesis class $\mathcal{H}$ to be any set of functions from $\mathcal{X}$ to $\mathbb{R}^{|\mathcal{Y}|}$. However, optimizing over a general function space is hard. Furthermore, controlling the size of such sets is also arguably challenging. To overcome this conundrum, in machine learning we typically use *parameterized hypotheses*. This largely simplifies the ERM problem, while the size of the hypothesis class can be adjusted by controlling the size of the *parameter space*. Below we present two special cases of the classification problem where we present possible hypothesis classes and associated loss functions.

An Important Special Case: Binary Classification with Linear Models
To study classification from a theoretical standpoint, it is often easier to consider the *binary classification* setting where $|\mathcal{Y}| = 2$. In this context, it is standard to consider a setup slightly different from the one mentioned above. Specifically, the output space $\mathcal{Y}$ is mapped to $\{-1, 1\}$, i.e., we chose Λ such that $\Lambda(y) \in \{-1, 1\}$ for all $y \in \mathcal{Y}$, instead of $\{1, 2\}$. Furthermore, the hypothesis space only considers real-valued functions $h : \mathcal{X} \to \mathbb{R}$, and the classifier is given by $C(\mathbf{x}) \coloneqq \Lambda^{-1}\left(\operatorname{sign}\left(h(\mathbf{x})\right)\right)$, where for any real value r, $\operatorname{sign}(r) = 1$ if $r \geq 0$ and -1 otherwise. All other notions (such as the definition of the ERM) adapt easily to this slightly modified setting.

The simplest hypothesis class for binary classification is the class of *linear models*. Consider a supervised-learning setting where $\mathcal{X} \subset \mathbb{R}^d$ and $\Lambda(y) \in \{-1, 1\}$ for all $y \in \mathcal{Y}$. Then a class of linear models, with *parameter space* $\Theta \subseteq \mathbb{R}^d$, is given

by

$$\mathcal{H} := \{\mathbf{x} \mapsto h_{\boldsymbol{\theta}}(\mathbf{x}) := \boldsymbol{\theta}^{\mathsf{T}}\mathbf{x} \,|\, \boldsymbol{\theta} \in \Theta\}, \tag{2.4}$$

where $(\cdot)^{\mathsf{T}}$ denotes the transpose operation. In (2.4), $\boldsymbol{\theta}$ is the *parameter* of a hypothesis, and the set Θ is the *parameter space* of the hypothesis class. While such a linear hypothesis class is usually used for simple classification tasks, it can sometimes be useful in building intuitions about the difficulty of the classification task at hand, e.g., by designing lower bounds. We illustrate, in Fig. 2.2 (left), a linear model for a classification problem with $\mathcal{X} = \mathbb{R}^2$ and with $\mathcal{Y} = \{-1, 1\}$.

Loss function ℓ plays a central role in finding a good model parameter. An obvious choice for the loss function to match the notion of misclassification error that we presented in Sect. 2.1.1 is the 0/1 loss function given by $\ell(h_{\boldsymbol{\theta}}(\mathbf{x}), y) := \mathbb{1}\{\text{sign}(h_{\boldsymbol{\theta}}(\mathbf{x})) \neq \Lambda(y)\}$. However, the non-differentiability (and non-convexity) of the indicator renders the resulting ERM problem difficult to solve from an optimization viewpoint. Hence, to promote solvability of the ERM problem, we use differentiable *surrogate loss functions* instead. For example in the binary classification problem, we can use the squared loss

$$\ell_{\text{squared}}(h_{\boldsymbol{\theta}}(\mathbf{x}), y) := (h_{\boldsymbol{\theta}}(\mathbf{x})\Lambda(y) - 1)^2, \ \forall (\mathbf{x}, y) \in \mathcal{X} \times \mathcal{Y}. \tag{2.5}$$

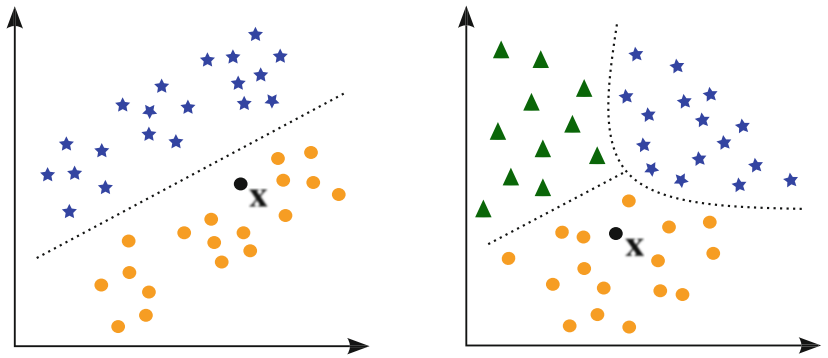

Fig. 2.2 On the left: Example of a linear model built on a training set $\mathcal{S}$ of data points in $\mathcal{X} \times \mathcal{Y} := \mathbb{R}^2 \times \{-1, 1\}$. An input with output 1 is visualized by a blue star, and otherwise by an orange disc. The dotted line represents a linear classifier belonging to the class defined in (2.4) that minimizes the empirical risk on $\mathcal{S}$ where the loss is simply the 0/1 loss function. For any new input $\mathbf{x} \in \mathcal{X}$, this model can be used to predict the output of $\mathbf{x}$. This is simply done by predicting 1 if $\mathbf{x}$ is above the dotted line and -1 otherwise. In the example shown, $\mathbf{x}$ is given the label -1. On the right: Example of a non-linear model built on a training set $\mathcal{S}$ of data points in $\mathcal{X} \times \mathcal{Y} := \mathbb{R}^2 \times \{1, 2, 3\}$. Similar to the linear classifier, the dotted line represents a hypothesis that minimizes the empirical risk on $\mathcal{S}$, but this hypothesis does not belong to a linear class

Since $\Lambda(y)^2 = 1$ for all $y \in \mathcal{Y}$, the right-hand side of the above can be rewritten as $(h_{\boldsymbol{\theta}}(\mathbf{x}) - \Lambda(y))^2$. In the special case of linear models, where $h_{\boldsymbol{\theta}}(\mathbf{x}) := \boldsymbol{\theta}^\intercal \mathbf{x}$, the squared loss reduces to $\left(\boldsymbol{\theta}^\intercal \mathbf{x} - \Lambda(y)\right)^2$.

The General Case of Non-linear Classification Tasks

We now come back to a more general case and return to our standard multi-label classification problem, where $|\mathcal{Y}| \geq 2$ and h is defined to be a mapping from $\mathcal{X}$ to $\mathbb{R}^{|\mathcal{Y}|}$. In this particular case, linear classifiers are often too simple to correctly capture the ground-truth distribution $\mathcal{D}$. In general, and even without going as far as image classification, we may need non-linear hypothesis classes to correctly classify a set of points, as illustrated in Fig. 2.2 (right). In modern-day machine learning, one of the most popular hypothesis classes is the class of non-linear models known as *neural networks*. A typical neural network yields a hypothesis that is a composition of N non-linear parametric functions $h^{(1)}_{\boldsymbol{\vartheta}_\mathbf{1}}, \ldots, h^{(N)}_{\boldsymbol{\vartheta}_\mathbf{N}}$, where each sub-parameter $\boldsymbol{\vartheta}_\mathbf{i}$ belongs to a set $\Theta_i \subseteq \mathbb{R}^{d_i}$. Specifically, parameter $\boldsymbol{\theta}$ for a hypothesis is given by the tuple $(\boldsymbol{\vartheta}_\mathbf{1}, \ldots, \boldsymbol{\vartheta}_\mathbf{N})$, and the hypothesis class is given by

$$\mathcal{H} := \left\{ \mathbf{x} \mapsto h_{\boldsymbol{\theta}}(\mathbf{x}) := h^{(N)}_{\boldsymbol{\vartheta}_\mathbf{N}} \circ \cdots \circ h^{(1)}_{\boldsymbol{\vartheta}_\mathbf{1}}(\mathbf{x}) \;\middle|\; \boldsymbol{\vartheta}_\mathbf{i} \in \Theta_i, \forall i \in [N] \right\}. \tag{2.6}$$

The parameter space of this hypothesis class is given by $\Theta := \Theta_1 \times \cdots \times \Theta_N$.

As we explained in the case of binary classification with linear models, we can obtain an optimal model from this class of neural networks by solving the ERM problem. Similar to the binary case, the $0/1$ loss is not a good candidate for ℓ; hence the need for a surrogate loss. In the context of neural networks, a common surrogate loss is the cross-entropy loss defined as follows:

$$\ell_{\text{cross-entropy}}(h_{\boldsymbol{\theta}}(\mathbf{x}), y) := -\sum_{k=1}^{|\mathcal{Y}|} \mathbb{1}\{\Lambda(y) = k\} \log \left(\frac{\exp\left([h_{\boldsymbol{\theta}}(\mathbf{x})]_k\right)}{\sum_{j=1}^{K} \exp\left([h_{\boldsymbol{\theta}}(\mathbf{x})]_j\right)} \right). \tag{2.7}$$

Simplification of the ERM

When using a parameterized hypothesis class, the ERM problem reduces to finding an optimal parameter $\boldsymbol{\theta}^*$ that solves the following optimization problem:

$$\min_{\boldsymbol{\theta} \in \Theta} \mathcal{L}(\boldsymbol{\theta}), \quad \text{where} \quad \mathcal{L}(\boldsymbol{\theta}) := \frac{1}{m} \sum_{i=1}^{m} \ell\left(h_{\boldsymbol{\theta}}(\mathbf{x}_\mathbf{i}), y_i\right). \tag{2.8}$$

In the remainder of this book, for simplicity, we assume that $\Theta = \mathbb{R}^d$, where d is called the *model size* or *model dimension*. Furthermore, because we mainly focus on the optimization part of machine learning algorithms, we promote solvability of the ERM problem and assume that ℓ is a differentiable *surrogate loss function*. Note that we just presented two of them, but in general the choice is not limited to either the square or the cross-entropy loss. In this book, we most often consider the generic

problem (without specifying the loss) and only consider the mapping $(\boldsymbol{\theta}, \mathbf{x}, y) \mapsto \ell(h_{\boldsymbol{\theta}}(\mathbf{x}), y)$ to be differentiable, with respect to the parameters $\boldsymbol{\theta}$. Furthermore, we assume that empirical loss function $\mathcal{L}(\boldsymbol{\theta})$ admits a minimum, i.e., there exists $\boldsymbol{\theta}^* \in \mathbb{R}^d$ such that $\mathcal{L}(\boldsymbol{\theta}^*) \leq \mathcal{L}(\boldsymbol{\theta})$ for all $\boldsymbol{\theta} \in \mathbb{R}^d$. Under these assumptions, the iterative gradient-based optimization algorithm can produce a good enough approximate solution to the ERM problem (2.8). The quality of approximation depends on additional assumptions we could make on the loss function $\mathcal{L}(\boldsymbol{\theta})$.

2.2 Standard Optimization Algorithms

We now turn our attention to the numerical technique of *gradient descent* for solving the ERM problem defined in (2.8). The gradient-descent methods, and their derivatives, represent the most widely used optimization algorithms in machine learning. These methods are iterative algorithms that all rely on the same guiding principle: At each iteration, the model parameters are updated as per a suitable *descent direction* of the empirical loss function. One such descent direction is determined by the *gradient* of the loss function $\mathcal{L}$, which we denote by $\nabla\mathcal{L}$. Specifically, updating the model parameters $\boldsymbol{\theta} \in \mathbb{R}^d$ along a vector facing negatively to the loss gradient, e.g., $-\nabla\mathcal{L}(\boldsymbol{\theta}) \in \mathbb{R}^d$, decrements the loss function.

Why the Gradient?
A good justification for using $-\nabla\mathcal{L}(\boldsymbol{\theta})$ as a descent direction is given by Taylor's theorem. According to this theorem, the change in the value of a differentiable function is linear in the change of the function's arguments, if the latter is sufficiently small. Moreover, the linear (or *first-order*) coefficient is given by the gradient of the function. Specifically, suppose that the current model parameters are given by $\boldsymbol{\theta} \in \mathbb{R}^d$. We would like to update the model parameters, say to $\boldsymbol{\theta}'$, with a smaller loss. By first-order Taylor polynomial approximation of function $\mathcal{L}$, we have

$$\mathcal{L}(\boldsymbol{\theta}') = \mathcal{L}(\boldsymbol{\theta}) + \left\langle \nabla\mathcal{L}(\boldsymbol{\theta}),\, \boldsymbol{\theta}' - \boldsymbol{\theta} \right\rangle + \psi\left(\boldsymbol{\theta}' - \boldsymbol{\theta}\right), \tag{2.9}$$

where $\psi\left(\boldsymbol{\theta}' - \boldsymbol{\theta}\right)$ is a function of $\boldsymbol{\theta}' - \boldsymbol{\theta}$ that is negligible when $\left\|\boldsymbol{\theta}' - \boldsymbol{\theta}\right\|$ is small. Specifically, $\frac{\psi(\boldsymbol{\theta}' - \boldsymbol{\theta})}{\left\|\boldsymbol{\theta}' - \boldsymbol{\theta}\right\|} \to 0$ as $\left\|\boldsymbol{\theta}' - \boldsymbol{\theta}\right\| \to 0$. Then, choosing $\boldsymbol{\theta}' = \boldsymbol{\theta} - \gamma\nabla\mathcal{L}(\boldsymbol{\theta})$, where γ is a positive real value, from (2.9), we have

$$\mathcal{L}(\boldsymbol{\theta}') = \mathcal{L}(\boldsymbol{\theta}) - \gamma \left\|\nabla\mathcal{L}(\boldsymbol{\theta})\right\|^2 + \psi\left(\gamma\nabla\mathcal{L}(\boldsymbol{\theta})\right).$$

Then, due to the fact that $\frac{\psi(\gamma\nabla\mathcal{L}(\boldsymbol{\theta}))}{\gamma\left\|\nabla\mathcal{L}(\boldsymbol{\theta})\right\|} \to 0$ as $\gamma \left\|\nabla\mathcal{L}(\boldsymbol{\theta})\right\| \to 0$, for sufficient small γ, we have $\mathcal{L}(\boldsymbol{\theta}') \leq \mathcal{L}(\boldsymbol{\theta})$. The inequality can be made strict when $\left\|\nabla\mathcal{L}(\boldsymbol{\theta})\right\| > 0$,[2]

which is true when $\boldsymbol{\theta}$ is suboptimal, i.e., $\mathcal{L}(\boldsymbol{\theta}) > \mathcal{L}(\boldsymbol{\theta}^*)$, where $\boldsymbol{\theta}^*$ denotes the minimum of the loss function.

Under a minimal set of assumptions regarding the *smoothness* of the loss function, the gradient-descent method produces a *stationary point*, i.e., $\boldsymbol{\theta}_{\mathbf{stat}}$ such that $\nabla\mathcal{L}(\boldsymbol{\theta}_{\mathbf{stat}}) = \mathbf{0} \in \mathbb{R}^d$. Note that a stationary point satisfies the first-order *necessary condition* of optimality. Specifically, $\boldsymbol{\theta}^*$ is a minimum of $\mathcal{L}$ on $\mathbb{R}^d$ *only if* $\nabla\mathcal{L}(\boldsymbol{\theta}^*) = \mathbf{0}$. The stationarity of a point is also *sufficient* for optimality when the loss function is *convex*, i.e., $\boldsymbol{\theta}_{\mathbf{stat}}$ is a minimum of a convex loss function. In which case, the gradient-descent approach yields an optimal solution. This, however, is not true in general, i.e., a stationary point need not be optimal when the loss function is *non-convex*. Nevertheless, a stationary point yields satisfactory model performance in many practical machine learning settings.

Gradient Computations

The success of a gradient-descent method depends on the quality of gradient computations. Computing an accurate gradient of the loss function is however not an easy operation. Recall, from (2.8), that $\mathcal{L}(\boldsymbol{\theta}) := \frac{1}{m}\sum_{i=1}^{m} \ell\left(h_{\boldsymbol{\theta}}(\mathbf{x_i}), y_i\right)$. In order to compute $\nabla\mathcal{L}(\boldsymbol{\theta})$, we have to compute gradients (with respect to $\boldsymbol{\theta}$) of the point-wise loss function $\ell\left(h_{\boldsymbol{\theta}}(\mathbf{x_i}), y_i\right)$ at all the m training data points $(\mathbf{x_i}, y_i) \in \mathcal{S}$. Computing these gradients might reduce to mere substitution of the training samples in a closed-form formula when the hypothesis class $\mathcal{H}$ is simple, such as the linear class described in (2.4). However, computing gradients is a computationally expensive operation when dealing with more sophisticated models such as neural networks. In fact, in the particular case of neural networks, we need a specialized algorithm called *backpropagation* for computing gradients. To enhance the practicality of gradient descent, instead of computing the exact value of $\nabla\mathcal{L}(\boldsymbol{\theta})$ on the whole training sample $\mathcal{S}$, we only estimate the gradient value on a randomly chosen data point. The resulting algorithm is called *stochastic* gradient descent. We can also sample multiple data points, in which case the algorithm is called *mini-batch* gradient descent. We present and discuss the performance of these methods in the remainder of this section.

Remark 2.2 The above intuition of first-order Taylor polynomial approximation can be extended to obtain other higher-order update schemes. For example, a second-order Taylor polynomial approximation explains how to construct second-order optimization schemes, of which Newton's method is a typical example. However, applying second-order schemes requires computing the Hessian of $\mathcal{L}$, which in general has computational complexity $O_d(d^2)$.[3] This renders second-order methods impractical for training models that are commonly used these days, since they have a large number of trainable parameters.

[2] **Hint:** When $\|\nabla\mathcal{L}(\boldsymbol{\theta})\| > 0$, there exists $\delta > 0$ such that $\frac{\psi(\gamma\nabla\mathcal{L}(\boldsymbol{\theta}))}{\gamma\|\nabla\mathcal{L}(\boldsymbol{\theta})\|} < \frac{1}{2}\|\nabla\mathcal{L}(\boldsymbol{\theta})\|$ for all γ such that $\gamma\|\nabla\mathcal{L}(\boldsymbol{\theta})\| < \delta$.

[3] We use the Bachmann–Landau notation $O_a()$ for describing the infinite behavior of a function when $a \to +\infty$. See section "Notation" for more details on notation.

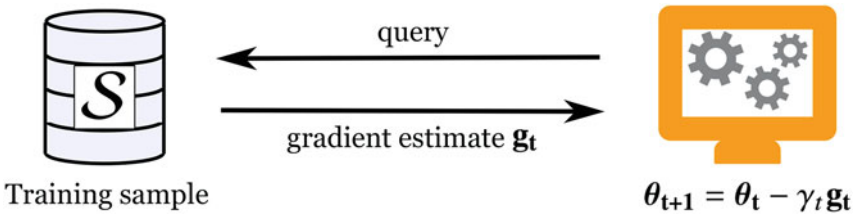

Fig. 2.3 Schematic of an iteration of a gradient-descent method. At iteration t, the current model parameters are given by $\boldsymbol{\theta_t}$. The algorithm queries the training sample $\mathcal{S}$ to compute an estimate $\mathbf{g_t}$ of the actual gradient of the loss function $\nabla\mathcal{L}(\boldsymbol{\theta_t})$. This gradient estimate is used to update the model parameters, specifically $\boldsymbol{\theta_{t+1}} := \boldsymbol{\theta_t} - \gamma_t \mathbf{g_t}$. For more details on this procedure, see Algorithms 1, 2, and 3

2.2.1 Gradient-Descent Methods

We now list the three main algorithms we focus on: *gradient descent*, *stochastic gradient descent*, and *mini-batch gradient descent*. These are optimization methods where the model is updated iteratively using the gradient estimates computed using the training sample. Any iteration of a gradient-descent method can be summarized according to Fig. 2.3.

Gradient Descent
This is a prototypical optimization scheme that is seldom used in practical machine learning settings. The gradient-descent technique, detailed in Algorithm 1 below, is the direct application of the intuition presented above in (2.9).

Algorithm 1 Gradient descent (GD)

Choose an initial model $\boldsymbol{\theta_1} \in \mathbb{R}^d$ for the model. The choice can be made at random or by using expert knowledge of the target model. In each iteration $t \geq 1$, given the current model $\boldsymbol{\theta_t}$, update the vector to

$$\boldsymbol{\theta_{t+1}} := \boldsymbol{\theta_t} - \gamma_t \nabla\mathcal{L}(\boldsymbol{\theta_t}) \,,$$

where γ_t is a positive real value referred as the *learning rate* at iteration t.

Although gradient descent is the method of choice for solving many optimization problems, it suffers from critical computational limitations and is not suitable for large-scale machine learning tasks. Indeed, recall that

$$\nabla\mathcal{L}(\boldsymbol{\theta_t}) := \frac{1}{m}\sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}}\, \ell\left(h_{\boldsymbol{\theta_t}}(\mathbf{x_i}), y_i\right) \,.$$

Computing the full gradient of $\mathcal{L}$, at each step t of the algorithm, involves computing m different gradients for the parameter $\boldsymbol{\theta}_\mathbf{t}$. In modern machine learning, the training sample $\mathcal{S}$ is often very large and cannot be processed entirely at every step of the learning procedure. The most standard way to circumvent this limitation is through the celebrated stochastic gradient-descent (SGD) algorithm.

Stochastic Gradient Descent

The rationale of SGD is to reduce the cost of gradient computations by using only one element of the training sample at a time. As described in Algorithm 2, SGD minimizes the loss function $\mathcal{L}$ using *stochastic estimates* of the gradient $\nabla\mathcal{L}$.

Algorithm 2 Stochastic gradient descent (SGD)

The initialization step is identical to GD. In each iteration $t \geq 1$, and given a current model $\boldsymbol{\theta}_\mathbf{t}$, compute a stochastic estimate $\mathbf{g_t}$ of the gradient $\nabla\mathcal{L}(\boldsymbol{\theta}_\mathbf{t})$. Specifically,

$$\mathbf{g_t} \coloneqq \nabla_{\boldsymbol{\theta}}\, \ell\left(h_{\boldsymbol{\theta}_\mathbf{t}}(\mathbf{x}),\, y\right) ,$$

where $(\mathbf{x},\, y)$ is drawn uniformly at random from the training sample $\mathcal{S}$. Upon computing $\mathbf{g_t}$, update the current model parameters $\boldsymbol{\theta}_\mathbf{t}$ to

$$\boldsymbol{\theta}_{\mathbf{t+1}} \coloneqq \boldsymbol{\theta}_\mathbf{t} - \gamma_t \mathbf{g_t} ,$$

where γ_t is the *learning rate* at iteration t.

Mini-batch Gradient Descent

Stochastic gradient estimation, while it enhances the practicality of the optimization algorithm, creates other issues due to concomitant *noise* in gradient computations. The algorithm can yield large numerical uncertainties when gradient noise is large, especially when the optimization problem being solved (i.e., the ERM problem in our case) is *ill-conditioned*. A natural way to reduce gradient noise is to sample multiple elements from $\mathcal{S}$ at a time, instead of just one. This is the driving principle behind the mini-batch gradient-descent (mini-batch GD) method, described in Algorithm 3.

Algorithm 3 Mini-batch gradient descent (mini-batch GD)

The initialization step is identical to GD. In each iteration $t \geq 1$, given a current model $\boldsymbol{\theta}_\mathbf{t}$, sample a batch of elements S_t uniformly at random from $\mathcal{S}$ (without replacement), and compute an estimate $\mathbf{g_t}$ of $\nabla\mathcal{L}(\boldsymbol{\theta}_\mathbf{t})$ as follows:

$$\mathbf{g_t} \coloneqq \frac{1}{|S_t|} \sum_{(\mathbf{x},y)\in S_t} \nabla_{\boldsymbol{\theta}_\mathbf{t}} \ell\big(h_{\boldsymbol{\theta}_\mathbf{t}}(\mathbf{x}),\, y\big) . \tag{2.10}$$

Then, update the model parameters similarly to SGD, i.e., obtain

$$\boldsymbol{\theta}_{\mathbf{t+1}} \coloneqq \boldsymbol{\theta}_\mathbf{t} - \gamma_t \mathbf{g_t} ,$$

where γ_t is the *learning rate* at iteration t.

Mini-batch GD is a strict generalization of GD and SGD. Indeed, mini-batch SGD reduces to SGD when $|S_t| = 1$ for all t and to GD when $|S_t| = m$ for all t. In general, when the data batch-sizes are not trivially set to 1 or m, mini-batch GD combines the best features of both SGD and GD. It provides an estimate $\mathbf{g_t}$ that is a better approximation of the true gradient of the loss function while still preserving the reasonable computational cost of the algorithm. In fact, when mini-batch GD is used with a fixed batch-size of b, i.e., $|S_t| = b$ for all t, the number of iterations needed for the algorithm to find a good model is divided by the b (compared to SGD). This helps reduce the learning time of the algorithm, since the mini-batch gradient computations can be parallelized for up to a certain value of b. Mini-batch GD is nowadays the backbone of optimization methods for large-scale machine learning. Owing to its simplicity and adaptive efficiency, mini-batch GD is implemented as a default optimizer in several machine learning frameworks, such as PyTorch [14] and TensorFlow [1], especially when considering deep neural network applications. In the remainder of this book, we mainly focus on mini-batch GD as a representative of gradient-descent schemes. We, however, provide side notes to adapt the results presented to both GD and SGD.

2.2.2 Convergence Rates

The number of iterations it takes for a gradient-descent method to produce a good approximate solution to the empirical risk minimization problem in (2.8) depends upon the additional assumption we could make about the loss function $\mathcal{L}$, specifically regarding its convexity. In this book, we focus on *non-convex* and *strongly convex* loss functions. In the following, we present an overview of these classes and the respective convergence guarantees of the gradient-descent methods described above. While we do not present proofs of these convergence guarantees for now, these can be derived from the general analysis presented later in Sect. 3.2. Specifically, the distributed methods presented in Sect. 3.1.2 reduce to the gradient-descent methods discussed above when $n = 1$.

Remark 2.3 The rationale for ignoring the class of *convex* functions, and instead of focusing on *strongly* convex functions, is the following. Similarly to [6], we argue that this choice better reflects the current practices in machine learning. In particular, in cases when the loss function is convex, e.g., training linear models, we add a strongly convex *regularizer* to the loss function to facilitate (or expedite) the optimization procedure. Moreover, results for convex (but not strongly convex) problems can be derived as extensions of the results presented in this book. This book aims to present the fundamentals of robust machine learning. Hence, the presented results constitute a starting point for building a more general theory encompassing other classes for the loss function.

Strongly Convex Case

We first discuss the case of strongly convex loss functions. The loss function $\mathcal{L}$ is said to be μ-strongly convex if, for any $\boldsymbol{\theta},\ \boldsymbol{\theta}' \in \mathbb{R}^d$, the following holds true.

$$\mathcal{L}(\boldsymbol{\theta}') \geq \mathcal{L}(\boldsymbol{\theta}) + \left\langle \nabla \mathcal{L}(\boldsymbol{\theta}),\, \boldsymbol{\theta}' - \boldsymbol{\theta} \right\rangle + \frac{\mu}{2} \left\| \boldsymbol{\theta}' - \boldsymbol{\theta} \right\|^2 , \tag{2.11}$$

where the μ, a.k.a., the *convexity parameter*, is a positive real value. Essentially, there exists a quadratic lower bound on the growth of the loss function. Note that a strongly convex function is also convex, specifically (2.11) implies the convexity condition: $\mathcal{L}(\boldsymbol{\theta}') \geq \mathcal{L}(\boldsymbol{\theta}) + \left\langle \nabla \mathcal{L}(\boldsymbol{\theta}),\, \boldsymbol{\theta}' - \boldsymbol{\theta} \right\rangle$ for all $\boldsymbol{\theta},\ \boldsymbol{\theta}' \in \mathbb{R}^d$. The converse, however, is not true, i.e., a convex function need not be strongly convex. Recall from the discussion in Sect. 2.2 that the convexity of $\mathcal{L}$ means that if $\boldsymbol{\theta}^*$ is a minimum of $\mathcal{L}$, then $\nabla \mathcal{L}(\boldsymbol{\theta}^*) = \mathbf{0}$. This particular observation, in conjunction with (2.11), implies that function $\mathcal{L}$ has a *unique minimum point*. Specifically, if $\boldsymbol{\theta}^*$ is a minimum of $\mathcal{L}$, then $\mathcal{L}(\boldsymbol{\theta}) > \mathcal{L}(\boldsymbol{\theta}^*)$ for all $\boldsymbol{\theta} \neq \boldsymbol{\theta}^*$. An example of a strongly convex function mapping $\mathbb{R}^2$ to $\mathbb{R}$ is shown below in Fig. 2.4.

In the context of gradient-descent methods, when the loss function $\mathcal{L}$ is strongly convex, we can bound the *suboptimality gap*, i.e., the difference between the actual value of the loss function and its minimum value, by a decreasing function of the number of iterations. Specifically, given a gradient-descent method discussed above, there exists a decreasing positive function $\phi : \mathbb{R} \to \mathbb{R}$ such that upon T iterations of the method, we have

$$\mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta}_{\mathbf{T+1}}) - \mathcal{L}^*\right] \leq \phi(T)\,, \tag{2.12}$$

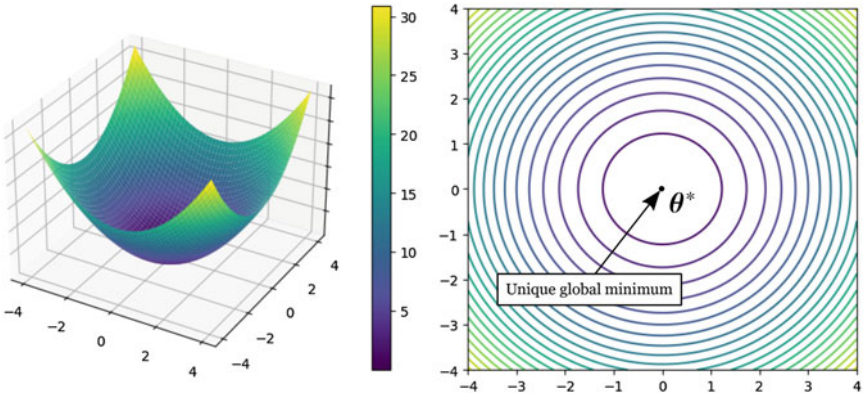

Fig. 2.4 Illustration of the strongly convex function $(z_1, z_2) \mapsto z_1^2 + z_2^2$ mapping $(-4, 4)^2$ to $\mathbb{R}$. On the left: 3D representation of the function, with a color bar indicating the values of the function. On the right: contour plot of the function with the same color bar. As shown, the function attains the minimum value at $(0, 0)$

Table 2.1 The order of ϕ for GD, SGD and mini-batch GD (with fixed batch-size b), with respect to the number of iterations T. This holds for a class of learning problems where $\mathcal{L}$ is smooth and strongly convex, assuming that the stochastic estimates of the gradients have bounded covariance trace

	GD	SGD	Mini-batch GD
$\phi(T)$	$O_T\left(e^{-T}\right)$	$O_T\left(\frac{1}{T}\right)$	$O_T\left(\frac{1}{bT}\right)$

where $\mathcal{L}^* := \min_{\boldsymbol{\theta}\in\mathbb{R}^d} \mathcal{L}(\boldsymbol{\theta})$ and the expectation $\mathbb{E}[\cdot]$ is taken over the (possible) randomness of the algorithm. The value of $\phi(T)$ decreases as T increases and approaches 0 when T approaches ∞. The function ϕ measures the *rate of convergence* of the algorithm, i.e., how slow or fast does the model $\boldsymbol{\theta}_{\mathbf{T+1}}$ approach the minimum point of $\mathcal{L}$. Specifically, we define the rate of convergence to be $\mathrm{R}(\phi) = \lim_{T\to\infty} \frac{\phi(T+1)}{\phi(T)}$. Note that, as ϕ is a positive decreasing function, $\mathrm{R}(\phi) \in (0, 1]$. We commonly say that the algorithm converges *linearly* when $\mathrm{R}(\phi) < 1$, and *sublinearly* when $\mathrm{R}(\phi) = 1$. In Table 2.1, we present a simplified (order) version of the mapping ϕ corresponding to GD, SGD, and mini-batch GD, in the strongly convex case. Note that while GD converges linearly, both SGD and mini-batch GD converge sublinearly. The deterioration of convergence rate is due to the presence of gradient noise in the case of SGD and mini-batch GD. We observe an "acceleration" of a constant factor b (i.e., the batch-size) in mini-batch GD due to the reduction by a similar factor in the variance of gradient noise. Detailed convergence analyses of these methods can be derived from the analyses presented in Sect. 3.2.3, upon substituting $n = 1$.

Non-convex Case

Although reasoning on convex loss functions greatly helps us measure the quality of an optimization algorithm, in practice, however, the loss function $\mathcal{L}$ need not be convex. For example, when training deep neural networks for image classification. In that case, solving the ERM problem (2.8) is NP-hard in general. A more reasonable goal is to compute a stationary point of $\mathcal{L}$, i.e., $\boldsymbol{\theta}_{\mathbf{stat}}$ such that $\nabla\mathcal{L}(\boldsymbol{\theta}_{\mathbf{stat}}) = \mathbf{0}$.

In theory, a stationary point can represent a local minimum, a local maximum, or a saddle point of the loss function. Verifying that $\boldsymbol{\theta}_{\mathbf{stat}}$ is a local minimum, in general, requires the second-order information of the loss function. Specifically, $\boldsymbol{\theta}_{\mathbf{stat}}$ is a local minimum of $\mathcal{L}$ if the gradient $\nabla\mathcal{L}(\boldsymbol{\theta}_{\mathbf{stat}}) = \mathbf{0}$ and the Hessian $\nabla^2\mathcal{L}(\boldsymbol{\theta}_{\mathbf{stat}}) \succcurlyeq \mathbf{0}$ (i.e., *positive semi-definite*).[4] Nevertheless, in the context of gradient descent, especially in the stochastic case, it is unusual for the method to converge to a local maximum or a saddle point. Specifically, local maxima and saddle points are *unstable* points. Accordingly, if the algorithm converges to a region in the space where the gradients are small, this region arguably encompasses a local

[4] This condition is referred as the *second-order sufficient condition* for optimality.

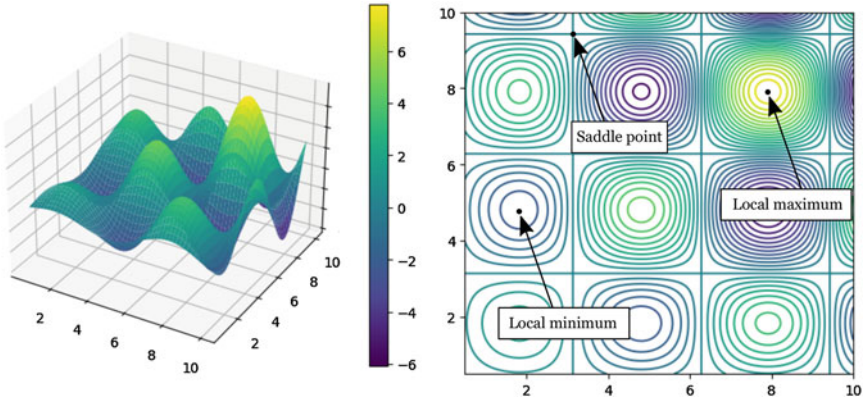

Fig. 2.5 Illustration of the non-convex function $(z_1, z_2) \mapsto \sqrt{|z_1 z_2|}\sin(z_1)\sin(z_2)$ mapping $(0, 10)^2$ to $\mathbb{R}$. On the left: 3D representation of the function, with a color bar indicating the values of the function. On the right: contour plot of the function with the same color bar

minimum. Figure 2.5 presents an example of a non-convex function with input space $\mathbb{R}^2$ and output space $\mathbb{R}$.

When the loss function is non-convex and smooth, we can guarantee a bound on the *stationarity gap*, measured by the (Euclidean) norm of the gradient at the model produced by the training scheme. Specifically, there exists a decreasing (positive) function $\phi : \mathbb{R} \to \mathbb{R}$ such that if the algorithm outputs $\hat{\boldsymbol{\theta}}$ at the completion of T iterations, we have

$$\mathbb{E}\left[\|\nabla \mathcal{L}\left(\hat{\boldsymbol{\theta}}\right)\|^2\right] \leq \phi(T)\,, \tag{2.13}$$

where the expectation $\mathbb{E}[\cdot]$ is taken over the (possible) randomness of the algorithm. Similar to the strongly convex case, ϕ measures the speed, on average, at which the method can provide a stationary point. In Table 2.2, we present this mapping ϕ for GD, SGD, and mini-batch GD. Detailed convergence analyses of these methods can be derived from the analyses presented in Sect. 3.2.2 upon substituting $n = 1$.

Table 2.2 The order of ϕ in (2.13) for GD, SGD, and mini-batch GD (with fixed batch-size b). This holds for a class of learning problems where $\mathcal{L}$ is smooth, assuming that the stochastic gradients have bounded covariance trace

	GD	SGD	Mini-batch GD
$\phi(T)$	$O_T\left(\frac{1}{T}\right)$	$O_T\left(\sqrt{\frac{1}{T}}\right)$	$O_T\left(\sqrt{\frac{1}{bT}}\right)$

2.2.3 Training Error and Gradient Complexity

We now have at our disposal algorithms, specifically GD, SGD, and mini-batch GD, for solving the ERM problem (2.8). Each method is iterative in nature: We start with an arbitrary estimate of a "good model" and proceed to improve this estimate using gradients of the point-wise loss function on the training sample. The objective is to minimize the overall *training error*, i.e., the suboptimality or stationarity gap at the output model, depending upon the class of the loss function at hand. But, at the same time, we would like to minimize the *gradient complexity*, i.e., the computation power spent in computing the gradients during the training procedure. We primarily focus on gradient computations because they represent the most expensive computational operations in the implementation of gradient-descent methods. Expectedly, there exists a trade-off between the training error and gradient complexity, which is precisely characterized by the convergence function ϕ of the method, as we explain below using the example of mini-batch GD.

Recall, from Algorithm 3, that in each iteration of mini-batch GD we compute b gradients. This means that if we run T iterations of mini-batch GD, our gradient complexity is proportional to bT. Suppose that the loss function $\mathcal{L}$ is strongly convex. In that case, recall that there exists a function $\phi(T) \in O_T\left(\frac{1}{bT}\right)$ such that for the last updated model $\boldsymbol{\theta}_{\mathbf{T+1}}$, the suboptimality gap is bounded as follows:

$$\mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta}_{\mathbf{T+1}}) - \mathcal{L}^*\right] \leq \phi(T)\,.$$

Therefore, in order to obtain a training error lower than or equal to ε, i.e., $\phi(T) = \varepsilon$, we need $T \in O_{\frac{1}{\varepsilon}}\left(\frac{1}{b\varepsilon}\right)$ iterations. Hence, the gradient complexity is given by $O_{\frac{1}{\varepsilon}}\left(\frac{1}{\varepsilon}\right)$. This relationship between gradient complexity and training error applies directly to SGD as well. For GD (i.e., Algorithm 1), where in each iteration we compute $|\mathcal{S}|$ number of gradients ($\mathcal{S}$ being the whole training sample), as $\phi(T) \in O_T\left(e^{-T}\right)$, the gradient complexity is given by $O_{\frac{1}{\varepsilon}}\left(|\mathcal{S}| \log \frac{1}{\varepsilon}\right)$. It appears that the gradient complexity of GD is smaller than mini-batch GD or SGD for obtaining the same training error ε. However, this is usually not true in practical scenarios where the size of the training sample, i.e., $|\mathcal{S}|$, is quite large and dominates the dependence on the training error in gradient complexity. We can extend the above analysis on the trade-off between training error and gradient complexity to the non-convex case by replacing the suboptimality gap by the stationarity gap (i.e., the squared Euclidean norm of the loss function's gradient) at the output model.

2.3 Chapter Notes

We have presented in this chapter a brief overview of supervised classification to illustrate and formalize the main concepts of machine learning. After formalizing the problem, we have reviewed some of the standard optimization tools for solving this task. Specifically, we have presented some of the most prominent optimization schemes in modern machine learning, namely gradient-based methods.

The aim of this chapter is primarily to introduce the general optimization problem induced by supervised learning, rather than an exhaustive presentation of the field. In particular, we have not discussed topics such as *generalization*, *structural risk minimization*, *consistency*, and *loss function calibration*. For more details on the topic of supervised learning, we refer the reader to introductory books on machine learning such as [3, 9, 11, 15]. Those books present a more comprehensive overview of the field. In particular, they provide sufficient information on the concepts of generalization and structural risk minimization to enable a proper understanding on why (and when) the ERM (or a regularized version of the ERM) yields a satisfactory model on unseen data. For this specific point, we refer the interested reader to [11, Chapter 4] and [3, Chapter 4]. For a more specific introduction to calibration and consistency of loss functions, we refer the interested reader to [4, 16, 17].

For advanced optimization algorithms and further details on gradient-descent methods, we refer the reader to [5–7, 12, 13]. Lower bounds on the convergence rate for first-order gradient-descent methods, with respect to T, can be found in [2, 13]. Although the upper bound(s) for the stochastic (and mini-batch) gradient-descent method we presented in this chapter are tight in general, they can be further improved in the non-convex case using *momentum-based variance reduction*, upon additionally assuming the loss function to be smooth with respect to the data points, as shown in [8]. We present further interesting properties of momentum-based optimization in the context of robust machine learning in Chap. 5. For more details, including a tighter analysis, on the convergence of stochastic (and mini-batch) gradient descent in the non-convex case, we refer the interested reader to paper [10].

References

1. Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jia Y, Jozefowicz R, Kaiser L, Kudlur M, Levenberg J, Mané D, Monga R, Moore S, Murray D, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker P, Vanhoucke V, Vasudevan V, Viégas F, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y, Zheng X (2015) TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from: Tensorflow.org
2. Arjevani Y, Carmon Y, Duchi JC, Foster DJ, Srebro N, Woodworth B (2023) Lower bounds for non-convex stochastic optimization. Math Program 199(1–2):165–214
3. Bach F (2023) Learning theory from first principles. MIT Press (Draft), Cambridge

4. Bartlett PL, Jordan MI, McAuliffe JD (2006) Convexity, classification, and risk bounds. J Amer Stat Assoc 101(473):138–156
5. Bottou L (1999) On-line learning and stochastic approximations. In: On-line learning in neural networks. Saad D (ed).. Publications of the Newton Institute. Cambridge University Press, Cambridge, pp 9–42
6. Bottou L, Curtis FE, Nocedal J (2018) Optimization methods for large-scale machine learning. Siam Rev 60(2):223–311
7. Boyd S, Vandenberghe L (2004) Convex optimization. Cambridge University Press, Cambridge
8. Cutkosky A, Orabona F (2019) Momentum-based variance reduction in non-convex SGD. In: Advances in Neural Information Processing Systems, vol 32. Curran Associates, Red Hook, pp 15236–15245
9. Hastie T, Tibshirani R, Friedman J (2001) The elements of statistical learning. Springer series in statistics. Springer New York, New York
10. Khaled A, Richtárik P (2023) Better theory for SGD in the nonconvex world. Trans Mach Learn Res. Survey Certification. ISSN: 2835-8856. https://openreview.net/forum?id=AU4qHN2VkS
11. Mohri M, Rostamizadeh A, Talwalkar A (2018) Foundations of machine learning. MIT Press, Cambridge
12. Moulines E, Bach F (2011) Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In: Advances in Neural Information Processing Systems. Shawe-Taylor J, Zemel R, Bartlett P, Pereira F, Weinberger K, vol 24. Curran Associates, Red Hook
13. Nesterov Y et al. (2018) Lectures on convex optimization, vol 137. Springer, Berlin
14. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, Desmaison A, Kopf A, Yang E, DeVito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai J, Chintala S (2019) PyTorch: an imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems, vol 32. Curran Associates, Red Hook, pp 8024–8035
15. Shalev-Shwartz S, Ben-David S (2014) Understanding machine learning: From theory to algorithms. Cambridge University Press, Cambridge
16. Steinwart I (2007) How to compare different loss functions and their risks. Construct Approx 26(2):225–287
17. Zhang T (2004) Statistical behavior and consistency of classification methods based on convex risk minimization. Ann Statist 32(1):56–85

Chapter 3
Federated Machine Learning

Abstract While presenting the basics of machine learning in the previous chapter, we assumed that we had access to a training sample $\mathcal{S}$ comprising data points that are (independently) drawn from the ground-truth distribution $\mathcal{D}$. We did not, however, explicitly mention how this dataset was collected or stored. A dominant practice in machine learning for many years has been to collect and store *user-generated data* on a single machine, which then uses the data to train a model. Although this approach is effective in training accurate models, it raises serious *privacy* concerns about the utilization of user data. Furthermore, training the model *centrally*, i.e., on a single machine, is becoming increasingly challenging due to the growing size of the models and the amount of data involved in modern machine learning tasks. To circumvent these limitations, a new paradigm has emerged: *federated machine learning*. This learning scheme consists in having several machines train a model in collaboration while keeping their data local. The computational load is distributed over multiple machines, and users can retain control over their local data. This paradigm is also sometimes coined as *collaborative* or simply *distributed* learning. In this chapter, we formalize the problem of federated machine learning in a server-based framework and present how to adapt gradient-descent methods in this context.

3.1 Machine Learning as a Distributed Problem

In this section, we first present the typical mathematical framework that characterizes federated machine learning, which is essentially machine learning on a distributed computing architecture. Then, we show how standard gradient-descent methods (such as mini-batch GD and, a fortiori, GD, and SGD) can be adapted to solve the federated machine learning problem.

R. Guerraoui et al., *Robust Machine Learning*, Machine Learning: Foundations, Methodologies, and Applications, https://doi.org/10.1007/978-981-97-0688-4_3

3.1.1 Problem Statement

We consider a system with n machines, hereafter referred as *nodes*, represented by the set $[n] \coloneqq \{1, \ldots, n\}$. These nodes communicate either *indirectly*, through means of a central machine (a.k.a., *server*), or *directly*, in a peer-to-peer manner, over a network. Each node i has access to m training samples represented by set $\mathcal{S}_i$ drawn independently from a distribution $\mathcal{D}_i$ defined over $\mathcal{X} \times \mathcal{Y}$. The resulting local loss function for each node i, denoted by $\mathcal{L}_i : \mathbb{R}^d \to \mathbb{R}$, is given by

$$\mathcal{L}_i(\boldsymbol{\theta}) \coloneqq \frac{1}{m} \sum_{(\mathbf{x}, y) \in \mathcal{S}_i} \ell\left(h_{\boldsymbol{\theta}}(\mathbf{x}), y\right) . \tag{3.1}$$

The objective of federated machine learning is to design a *distributed algorithm* that enables each node to compute a common optimal model $\boldsymbol{\theta}^* \in \mathbb{R}^d$ that minimizes the average loss function for all the nodes, i.e., is a solution to

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^d} \mathcal{L}(\boldsymbol{\theta}) \ , \ \text{where} \ \ \mathcal{L}(\boldsymbol{\theta}) \coloneqq \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_i(\boldsymbol{\theta}) . \tag{3.2}$$

Remark 3.1 For simplicity, we assume that all the nodes have the same number of data points, i.e., m. In general, when nodes have a nonuniform number of training samples, instead of minimizing the average of their local loss functions, we minimize a weighted average, i.e., $\sum_{i=1}^{n} p_i \mathcal{L}_i(\boldsymbol{\theta})$, where p_i's are positive real values such that $\sum_{i=1}^{n} p_i = 1$. Usually, if each node i has m_i data points, we choose $p_i = m_i / \sum_{i=1}^{n} m_i$. Although we focus on the minimization of the average loss, the results and techniques presented in this book can be easily extended to weighted average loss minimization by simply modifying each local loss function $\mathcal{L}_i$ to $\widetilde{\mathcal{L}}_i = n p_i \mathcal{L}_i$.

Comparison to the Centralized Methodology

The above optimization problem can be solved by simply collecting the respective training samples held by the nodes at a single machine and then implementing one of the centralized machine learning algorithms on the collective training set, e.g., one of the gradient-descent methods presented in Sect. 2.2.1. However, this centralized approach imposes disproportionate computational workload on a single machine and requires nodes to compromise the ownership of their data, which could discourage many nodes from participating in the learning procedure. A distributed machine learning algorithm, presented shortly in Sect. 3.1.2, can solve the optimization problem (3.2) without having the nodes directly share their local training samples. Instead, in a federated machine learning approach, a node i is required to share only information about its local loss function $\mathcal{L}_i$ at a finite number of points in the parameter space. Federated learning, however, requires all the nodes to coordinate their local actions during the entire learning procedure. This is unlike

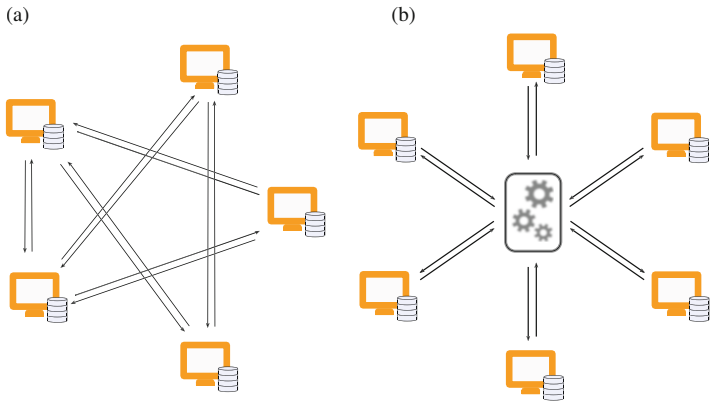

Fig. 3.1 Illustration of (**a**) peer-to-peer and (**b**) server-based system architectures

the centralized methodology, where the nodes need not participate in the learning procedure themselves, once they have exported their local training samples.

System Architectures
Two main types of distributed system architecture have been considered: *server-based* and *peer-to-peer*. We illustrate the two architectures in Fig. 3.1. In the server-based architecture, the nodes collaborate through the means of a central coordinator commonly referred to as the server, sometimes called the *parameter server*. Note that the server can also be considered as an abstraction of a cluster of several machines that communicate with the nodes. In the peer-to-peer architecture, the nodes collaborate by communicating directly with each other, on a given network topology. Note that, when $n = 1$, both the peer-to-peer and the server-based architectures can easily be reduced to the centralized case.[1] Accordingly, all the results we present in the remainder of this chapter immediately extend to the centralized setting. In the subsequent section, we present the simplest, and perhaps the most natural, distributed implementation of mini-batch GD in the server-based architecture. This implementation also easily extends to that of distributed GD and of distributed SGD.

Remark 3.2 Throughout the rest of the book, we mainly focus on the server-based architecture, owing to its wider utility in practical systems. Some discussion and references pertaining to the peer-to-peer architecture can be found in the chapter notes of Chaps. 3, 4, and 5.

[1] For the server-based architecture, reducing the problem to the centralized setting when $n = 1$ requires merging the server and the node into a single machine.

3.1.2 Distributed Gradient-Descent Methods

The basic operations of a gradient-descent method in the server-based architecture are very similar to those of the centralized setting, except that an iteration in the former proceeds in two phases: *the local phase* and *the global phase*. In each iteration t, the server initializes the local phase upon broadcasting its current model $\boldsymbol{\theta_t}$ to all the nodes. Each node i computes an estimate $\mathbf{g_t^{(i)}}$ of the gradient of its local loss function $\mathcal{L}_i$ at the current model $\boldsymbol{\theta_t}$. The nodes return their respective gradient estimates to the server, thus initiating the second phase. In the global phase, the server updates the current model $\boldsymbol{\theta_t}$ to $\boldsymbol{\theta_{t+1}}$ by linearly combining $\boldsymbol{\theta_t}$ with the average of the local gradients' estimates sent by the nodes. This distributed implementation of a gradient-descent method is illustrated in Fig. 3.2.

Distributed Mini-batch GD

The implementation of the local phase of a distributed gradient-descent method depends upon the algorithm we intend to implement. In the specific case of the *distributed mini-batch GD* (DMGD) algorithm, each node queries a batch of data points (at random) from its local dataset and computes the corresponding gradient estimate for the local loss function at the current model. Hence, this gradient is a *stochastic* estimate of its local loss function. We describe the complete scheme of DMGD below, in Algorithm 4.

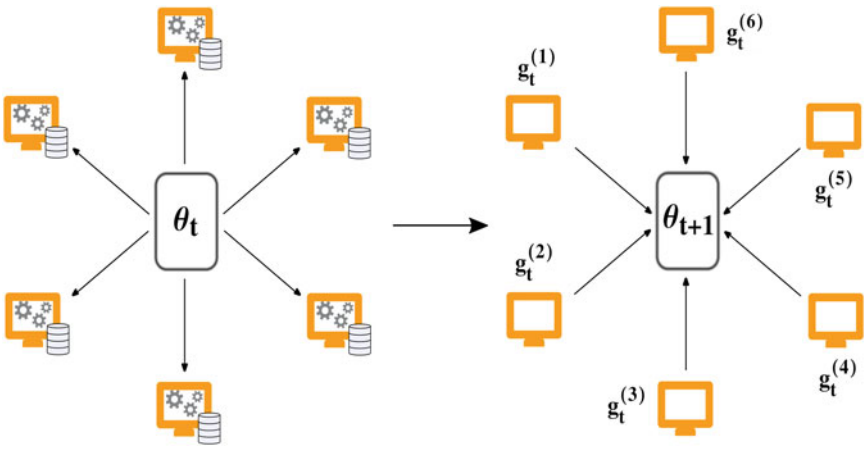

Fig. 3.2 An iteration of a distributed gradient-descent method with a server

Algorithm 4 Distributed mini-batch GD (DMGD)

The server initializes the procedure by choosing an initial model $\boldsymbol{\theta_1} \in \mathbb{R}^d$. Similar to the mini-batch GD method, this initial model can either be chosen arbitrarily or using some expert knowledge of the model being trained. In each iteration $t \geq 1$, the server maintains a model parameter $\boldsymbol{\theta_t}$ that is broadcast to all the nodes. The rest of the iteration is as follows:

1. **Local Phase.** Each node $i \in [n]$ independently and randomly samples a set of b data points, denoted by $S_t^{(i)}$, from the local training sample $\mathcal{S}_i$ (without replacement), and computes the local gradient estimate

$$\mathbf{g_t^{(i)}} \coloneqq \frac{1}{b} \sum_{(\mathbf{x},y)\in S_t^{(i)}} \nabla_{\boldsymbol{\theta}} \ell\big(h_{\boldsymbol{\theta_t}}(\mathbf{x}), y\big)\,. \tag{3.3}$$

2. **Global Phase.** Each node i returns to the server the gradient estimate $\mathbf{g_t^{(i)}}$. Upon receiving the gradients from all the nodes, the server computes their average

$$\mathbf{g_t} \coloneqq \frac{1}{n} \sum_{i=1}^{n} \mathbf{g_t^{(i)}}\,, \tag{3.4}$$

and updates the current model parameters as follows:

$$\boldsymbol{\theta_{t+1}} \coloneqq \boldsymbol{\theta_t} - \gamma_t\, \mathbf{g_t}\,, \tag{3.5}$$

where γ_t is the *learning rate* at iteration t.

Distributed GD and SGD

Recall from Chap. 2 that GD and SGD are two special instances of mini-batch GD where the gradient estimates are computed with data batches of sizes m and 1, respectively. Therefore, the implementations of these methods in the considered distributed environment are obtained from Algorithm 4 upon modifying the gradient computations in the local phases with $b = m$ and $b = 1$. In the remainder of this book, we present our convergence results for DMGD and describe how to adapt these results to *distributed GD* (DGD) and *distributed SGD* (DSGD). The results are accompanied with self-contained proofs. Although the reader can skip the proofs without loss of continuity, it is highly recommended to read them for acquiring an in-depth understanding of the topic.

3.2 Convergence of Distributed Mini-batch Gradient Descent (DMGD)

In this section, we present the convergence analysis for the distributed mini-batch GD (DMGD) method. A summary of the convergence rates of DMGD (consequently, DGD and DSGD) is presented in Table 3.1 (Sect. 3.3). We start by making some assumptions concerning the smoothness of the loss function and the

level of uncertainty in the gradient estimates. These assumptions have been observed to hold true for many learning problems.

3.2.1 Assumptions and Notation

First, as stated below, we assume that the point-wise loss function $\ell(h_{\boldsymbol{\theta}}(\mathbf{x}), y)$ is Lipschitz smooth with respect to the vector $\boldsymbol{\theta}$ at any data point $(\mathbf{x}, y)$. In the following, ∇ by default denotes the gradient with respect to $\boldsymbol{\theta}$.

Assumption 3.1 (Lipschitz Smoothness) There exists a real value $L > 0$ such that for all $\boldsymbol{\theta}, \boldsymbol{\theta}' \in \mathbb{R}^d$, the following holds true:

$$\left\| \nabla \ell(h_{\boldsymbol{\theta}}(\mathbf{x}), y) - \nabla \ell\big(h_{\boldsymbol{\theta}'}(\mathbf{x}), y\big) \right\| \leq L \left\| \boldsymbol{\theta} - \boldsymbol{\theta}' \right\| .$$

Second, as stated below, we assume that for each node i, the point-wise gradients, i.e., $\nabla \ell(h_{\boldsymbol{\theta}}(\mathbf{x}), y)$, $(\mathbf{x}, y) \in \mathcal{S}_i$, have a uniformly bounded covariance trace. Recall that, by definition of $\mathcal{L}_i(\boldsymbol{\theta})$ in (3.1), $\nabla \mathcal{L}_i(\boldsymbol{\theta}) = \frac{1}{m} \sum_{(\mathbf{x}, y) \in \mathcal{S}_i} \nabla \ell(h_{\boldsymbol{\theta}}(\mathbf{x}), y)$ for all $\boldsymbol{\theta}$.

Assumption 3.2 (Bounded Covariance Trace) There exists a real value σ such that for all $i \in [n]$ and $\boldsymbol{\theta} \in \mathbb{R}^d$, the following holds true:

$$\frac{1}{m} \sum_{(\mathbf{x}, y) \in \mathcal{S}_i} \left\| \nabla \ell(h_{\boldsymbol{\theta}}(\mathbf{x}), y) - \nabla \mathcal{L}_i(\boldsymbol{\theta}) \right\|^2 \leq \sigma^2 .$$

When imposing these assumptions, we consider two main types of loss functions for which we analyze the convergence of DMGD below: non-convex (in Sect. 3.2.2) and strongly convex loss functions (in Sect. 3.2.3). Due to the stochastic nature of the update procedure when using random batches, we characterize the convergence of DMGD in expectation. Specifically, we denote by $\mathcal{P}_t$ the transcript or the history of the algorithm, until the t-th iteration. This history is defined as follows:

$$\mathcal{P}_t \coloneqq \{\boldsymbol{\theta_1}, \ldots, \boldsymbol{\theta_t}\} . \tag{3.6}$$

We denote by $\mathbb{E}_t[\cdot]$ the conditional expectation $\mathbb{E}[\cdot \,|\, \mathcal{P}_t]$. Thus, upon executing T iterations of DMGD, the expectation over the randomness of the algorithm, denoted by $\mathbb{E}[\cdot]$, is defined by the nested conditional expectations over the iterations, i.e., $\mathbb{E}[\cdot] = \mathbb{E}_1[\cdots \mathbb{E}_T[\cdot]]$.

3.2.2 Non-convex Loss Functions

We begin by presenting the case when the loss function might be non-convex, which is arguably quite often the case in machine learning. As mentioned in Chap. 2, when the global average loss function $\mathcal{L}(\boldsymbol{\theta})$ is non-convex, the best outcome for

the DMGD algorithm is a stationary point, i.e., $\boldsymbol{\theta}^*$ such that $\nabla\mathcal{L}(\boldsymbol{\theta}^*) = \mathbf{0}$. The following theorem shows that, after executing T iterations of the DMGD method with a sufficiently small learning rate, if the server chooses to output a random model from the set $\{\boldsymbol{\theta_1}, \ldots, \boldsymbol{\theta_T}\}$, then the algorithm approximates a stationary point of the loss function $\mathcal{L}$. Recall that $\mathcal{L}^* := \min_{\boldsymbol{\theta}\in\mathbb{R}^d} \mathcal{L}(\boldsymbol{\theta})$.

Theorem 3.1 *Suppose Assumptions 3.1 and 3.2. Consider T iterations of the DMGD algorithm with batch-size $b \in [1, m)$. If $\gamma_t = \gamma \in \left(0, \frac{1}{L}\right]$ for all $t \in [T]$, then the following holds true:*

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}(\boldsymbol{\theta_t})\|^2\right] \leq \frac{2\left(\mathcal{L}(\boldsymbol{\theta_1}) - \mathcal{L}^*\right)}{T\,\gamma} + \frac{L\sigma^2}{nb}\gamma\,. \tag{3.7}$$

Proof By the definition of $\mathcal{L}(\boldsymbol{\theta})$ in (3.2), we note that the Lipschitz smoothness of $\ell\left(h_{\boldsymbol{\theta}}(\mathbf{x}), y\right)$ with respect to $\boldsymbol{\theta}$, i.e., Assumption 3.1, implies Lipschitz smoothness for the global loss function $\mathcal{L}(\boldsymbol{\theta})$, with the same Lipschitz constant L. Consider an arbitrary $t \in [T]$. From the Lipschitz smoothness inequality (shown in Lemma A.1, Appendix A.1), we have

$$\mathcal{L}(\boldsymbol{\theta_{t+1}}) \leq \mathcal{L}(\boldsymbol{\theta_t}) + \langle\nabla\mathcal{L}(\boldsymbol{\theta_t}),\, \boldsymbol{\theta_{t+1}} - \boldsymbol{\theta_t}\rangle + \frac{L}{2}\|\boldsymbol{\theta_{t+1}} - \boldsymbol{\theta_t}\|^2\,.$$

Substituting $\boldsymbol{\theta_{t+1}}$ on the right-hand side from (3.5), then taking the expectation $\mathbb{E}_t[\cdot]$ on both sides, we obtain that

$$\mathbb{E}_t\left[\mathcal{L}(\boldsymbol{\theta_{t+1}})\right] \leq \mathcal{L}(\boldsymbol{\theta_t}) - \gamma_t\langle\nabla\mathcal{L}(\boldsymbol{\theta_t}),\, \mathbb{E}_t[\mathbf{g_t}]\rangle + \frac{L}{2}\gamma_t^2\,\mathbb{E}_t\left[\|\mathbf{g_t}\|^2\right]. \tag{3.8}$$

For all $i \in [n]$, by definition of $\mathbf{g_t^{(i)}}$ in (3.3), $\mathbb{E}_t\left[\mathbf{g_t^{(i)}}\right] = \nabla\mathcal{L}_i(\boldsymbol{\theta_t})$. Therefore, by definition of $\mathbf{g_t}$ in (3.4) and $\mathcal{L}(\boldsymbol{\theta})$ in (3.2), we have

$$\mathbb{E}_t[\mathbf{g_t}] = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}_t\left[\mathbf{g_t^{(i)}}\right] = \frac{1}{n}\sum_{i=1}^{n}\nabla\mathcal{L}_i(\boldsymbol{\theta_t}) = \nabla\mathcal{L}(\boldsymbol{\theta_t})\,. \tag{3.9}$$

Substituting from (3.9) in (3.8), we have

$$\mathbb{E}_t\left[\mathcal{L}(\boldsymbol{\theta_{t+1}})\right] \leq \mathcal{L}(\boldsymbol{\theta_t}) - \gamma_t\|\nabla\mathcal{L}(\boldsymbol{\theta_t})\|^2 + \frac{L}{2}\gamma_t^2\,\mathbb{E}_t\left[\|\mathbf{g_t}\|^2\right]. \tag{3.10}$$

(continued)

We now obtain an upper bound for $\mathbb{E}_t\left[\|\mathbf{g_t}\|^2\right]$. Note that

$$\mathbb{E}_t\left[\|\mathbf{g_t} - \nabla\mathcal{L}(\boldsymbol{\theta_t})\|^2\right] = \mathbb{E}_t\left[\left\|\frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{g_t^{(i)}} - \nabla\mathcal{L}_i(\boldsymbol{\theta_t})\right)\right\|^2\right]. \tag{3.11}$$

Furthermore, as each node samples its data points independently, for any pair of non-identical nodes $(i,\ j) \in [n] \times [n]$, we have

$$\begin{aligned}&\mathbb{E}_t\left[\left\langle \mathbf{g_t^{(i)}} - \nabla\mathcal{L}_i(\boldsymbol{\theta_t}),\ \mathbf{g_t^{(j)}} - \nabla\mathcal{L}_j(\boldsymbol{\theta_t})\right\rangle\right]\\ =&\left\langle \mathbb{E}_t\left[\mathbf{g_t^{(i)}}\right] - \nabla\mathcal{L}_i(\boldsymbol{\theta_t}),\ \mathbb{E}_t\left[\mathbf{g_t^{(j)}}\right] - \nabla\mathcal{L}_j(\boldsymbol{\theta_t})\right\rangle = 0\,.\end{aligned}$$

Therefore, we have

$$\mathbb{E}_t\left[\left\|\frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{g_t^{(i)}} - \nabla\mathcal{L}_i(\boldsymbol{\theta_t})\right)\right\|^2\right] = \frac{1}{n^2}\sum_{i=1}^{n}\mathbb{E}_t\left[\left\|\mathbf{g_t^{(i)}} - \nabla\mathcal{L}_i(\boldsymbol{\theta_t})\right\|^2\right]. \tag{3.12}$$

Recall that, for each $i \in [n]$,

$$\mathbf{g_t^{(i)}} = \frac{1}{b}\sum_{(\mathbf{x},y)\in S_t^{(i)}} \nabla_{\boldsymbol{\theta}}\,\ell\big(h_{\boldsymbol{\theta_t}}(\mathbf{x}),\,y\big)\,,$$

and where $S_t^{(i)}$ is a set of b data points sampled randomly (without replacement) from $\mathcal{S}_i$. Then, by definition of the expectation $\mathbb{E}_t\,[\cdot]$, we have[a]

$$\begin{aligned}&\mathbb{E}_t\left[\left\|\mathbf{g_t^{(i)}} - \nabla\mathcal{L}_i(\boldsymbol{\theta_t})\right\|^2\right]\\ &\quad= \frac{m-b}{b(m-1)}\left(\frac{1}{m}\sum_{(\mathbf{x},y)\in\mathcal{S}_i}\left\|\nabla\ell\big(h_{\boldsymbol{\theta_t}}(\mathbf{x}),\,y\big) - \nabla\mathcal{L}_i(\boldsymbol{\theta_t})\right\|^2\right).\end{aligned}$$

[a] This equality is a standard sampling result, e.g., see [4, Chapter 7].

(continued)

As $\frac{m-b}{m-1} \leq 1$, using Assumption 3.2 in the above implies that, for all $i \in [n]$,

$$\mathbb{E}_t \left[\left\| \mathbf{g_t^{(i)}} - \nabla \mathcal{L}_i(\boldsymbol{\theta_t}) \right\|^2 \right] \leq \frac{\sigma^2}{b} .$$

Using this in (3.12) and subsequently in (3.11), we obtain that

$$\mathbb{E}_t \left[\|\mathbf{g_t} - \nabla \mathcal{L}(\boldsymbol{\theta_t})\|^2 \right] \leq \frac{\sigma^2}{nb} . \tag{3.13}$$

As $\langle \mathbb{E}_t [\mathbf{g_t}] - \nabla \mathcal{L}(\boldsymbol{\theta_t}),\ \nabla \mathcal{L}(\boldsymbol{\theta_t}) \rangle = 0$ (see (3.9)), due to (3.13), the following holds true.

$$\mathbb{E}_t \left[\|\mathbf{g_t}\|^2 \right] = \mathbb{E}_t \left[\|\mathbf{g_t} - \nabla \mathcal{L}(\boldsymbol{\theta_t})\|^2 \right] + \|\nabla \mathcal{L}(\boldsymbol{\theta_t})\|^2 \leq \frac{\sigma^2}{nb} + \|\nabla \mathcal{L}(\boldsymbol{\theta_t})\|^2 .$$

Substituting from the above in (3.10), taking total expectation $\mathbb{E}[\cdot]$ on both sides, and rearranging the terms yield the following:

$$\mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta_{t+1}})\right] \leq \mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta_t})\right] - \gamma_t \left(1 - \frac{L}{2}\gamma_t\right) \mathbb{E}\left[\|\nabla \mathcal{L}(\boldsymbol{\theta_t})\|^2\right] + \frac{L\sigma^2}{2nb}\gamma_t^2 , \tag{3.14}$$

which can equivalently be written as (recalling that $\gamma_t > 0$)

$$\left(1 - \frac{L}{2}\gamma_t\right) \mathbb{E}\left[\|\nabla \mathcal{L}(\boldsymbol{\theta_t})\|^2\right] \leq \frac{\mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta_t})\right] - \mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta_{t+1}})\right]}{\gamma_t} + \frac{L\sigma^2}{2nb}\gamma_t .$$

Lastly, recall that for all $t \in [T]$, we have $\gamma_t = \gamma \leq \frac{1}{L}$, hence $1 - \frac{L}{2}\gamma_t \geq \frac{1}{2}$. Thus, upon substituting $\gamma_t = \gamma$ in the above and summing both sides from $t = 1$ to $t = T$, we obtain that

$$\sum_{t=1}^{T} \mathbb{E}\left[\|\nabla \mathcal{L}(\boldsymbol{\theta_t})\|^2\right] \leq \frac{2\,\mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta_1}) - \mathcal{L}(\boldsymbol{\theta_{T+1}})\right]}{\gamma} + \frac{L\sigma^2}{nb}\gamma T . \tag{3.15}$$

Note that as $\mathcal{L}(\boldsymbol{\theta}) \geq \mathcal{L}^*$ for all $\boldsymbol{\theta} \in \mathbb{R}^d$, we have $\mathcal{L}(\boldsymbol{\theta_1}) - \mathcal{L}(\boldsymbol{\theta_{T+1}}) = \mathcal{L}(\boldsymbol{\theta_1}) - \mathcal{L}^* + \mathcal{L}^* - \mathcal{L}(\boldsymbol{\theta_{T+1}}) \leq \mathcal{L}(\boldsymbol{\theta_1}) - \mathcal{L}^*$. Using this in (3.15), and dividing both sides by T, concludes the proof. □

The convergence result shown in Theorem 3.1 tells us that, upon selecting an appropriate learning rate γ, the DMGD algorithm generates a sequence of models

that on average can have an arbitrarily small stationarity gap, i.e., the gradient norm for global loss $\mathcal{L}$. Specifically, we obtain the following corollary of Theorem 3.1.

Corollary 3.1 *Suppose Assumptions 3.1 and 3.2. Consider T iterations of the DMGD algorithm with batch-size $b \in [1, m)$. Let Δ be a real value such that $2\left(\mathcal{L}(\boldsymbol{\theta_1}) - \mathcal{L}^*\right) \leq \Delta$, and set*

$$\gamma_t = \gamma = \min\left\{\frac{1}{L}, \sqrt{\frac{nb\Delta}{\sigma^2 LT}}\right\}, \forall t \in [T]\,. \tag{3.16}$$

Then, as soon as $T \geq \frac{nb\Delta}{\sigma^2}$, the following holds true:

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}(\boldsymbol{\theta_t})\|^2\right] \leq 2\sqrt{\frac{\Delta L\sigma^2}{nbT}}\,. \tag{3.17}$$

Proof Note that by definition of γ, we have $\gamma_t = \gamma \leq \frac{1}{L}$ for all $t \in [T]$. Thus, by Theorem 3.1, we have

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}(\boldsymbol{\theta_t})\|^2\right] \leq \frac{2\left(\mathcal{L}(\boldsymbol{\theta_1}) - \mathcal{L}^*\right)}{T\gamma} + \frac{L\sigma^2}{nb}\gamma\,.$$

Let us assume that $T \geq \frac{nb\Delta}{\sigma^2}$. In this case, $\frac{1}{L} \geq \sqrt{\frac{nb\Delta}{\sigma^2 LT}}$, hence $\gamma = \sqrt{\frac{nb\Delta}{\sigma^2 LT}}$. Plugging this γ in the above, we obtain that

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}(\boldsymbol{\theta_t})\|^2\right] \leq \frac{2\sqrt{L\sigma^2}\left(\mathcal{L}(\boldsymbol{\theta_1}) - \mathcal{L}^*\right)}{\sqrt{\Delta nbT}} + \sqrt{\frac{\Delta L\sigma^2}{nbT}}\,.$$

Recall that $2\left(\mathcal{L}(\boldsymbol{\theta_1}) - \mathcal{L}^*\right) \leq \Delta$. This concludes the proof. □

The corollary implies that if we choose a model $\hat{\boldsymbol{\theta}}$ randomly from $\{\boldsymbol{\theta_1}, \ldots, \boldsymbol{\theta_T}\}$, then (in expectation) the model we select approximates a stationary point of $\mathcal{L}$ with an error in $O_T\left(\sqrt{\frac{\sigma^2}{nbT}}\right)$. Specifically, under the conditions specified in Corollary 3.1,

$$\mathbb{E}\left[\left\|\nabla\mathcal{L}\left(\hat{\boldsymbol{\theta}}\right)\right\|^2\right] \in O_T\left(\sqrt{\frac{\sigma^2}{nbT}}\right),$$

where the expectation in the above is also applied over the choice of $\hat{\boldsymbol{\theta}}$, i.e., $\mathbb{E}[\cdot] = \mathbb{E}_{\hat{\boldsymbol{\theta}}}[\mathbb{E}_1[\cdots \mathbb{E}_T[\cdot]]]$. This shows that the primary advantage of DMGD over its centralized counterpart is its increased computational efficiency. Indeed, to obtain the same convergence guarantees as its centralized counterpart, DMGD divides the gradient complexity, i.e., the computation cost for computing the total gradients, of each node by the total size of the system n. Specifically, for training error of ε, i.e., $\mathbb{E}\left[\left\|\nabla \mathcal{L}\left(\hat{\boldsymbol{\theta}}\right)\right\|^2\right] \leq \varepsilon$, DMGD uses $T \in O_{\frac{1}{\varepsilon}}\left(\frac{\sigma^2}{nb\varepsilon^2}\right)$ iterations in contrast to $O_{\frac{1}{\varepsilon}}\left(\frac{\sigma^2}{b\varepsilon^2}\right)$ iterations used by mini-batch GD. This makes sense intuitively because when every node computes b point-wise gradients, the whole system effectively computes nb point-wise gradients, which should expedite the learning procedure, i.e., reduce the total iterations needed to ensure the desired training error. In short, implementing DMGD in the server-based setting is equivalent to running mini-batch GD with batch-size nb in a centralized environment, i.e., in a system with one node doing all the computations. By reducing the gradient computational workload for each node, DMGD makes training of large complex models efficient.

3.2.3 Strongly Convex Loss Functions

We now turn our attention to the strongly convex case, i.e., when the loss function $\mathcal{L}$ is a priori known to be strongly convex. As we previously observed in Chap. 2, although studying a non-convex function is more general and holds true for a larger set of learning problems, taking a look at strongly convex functions enables us to obtain faster convergence rates and provides guarantees directly on the suboptimality gap, i.e., the value of the loss function, instead of its gradient's norm. To study the strongly convex case, in this book, we consider a class of functions that is slightly more general than that of strongly convex functions. Specifically, we study functions that satisfy the *Polyak–Lojasiewicz* (PL) inequality, stated below.

Assumption 3.3 (PL Inequality) There exists a real value $\mu > 0$ such that for all $\boldsymbol{\theta} \in \mathbb{R}^d$,

$$\|\nabla \mathcal{L}(\boldsymbol{\theta})\|^2 \geq 2\mu \left(\mathcal{L}(\boldsymbol{\theta}) - \mathcal{L}^*\right),$$

where $\mathcal{L}^* := \min_{\boldsymbol{\theta} \in \mathbb{R}^d} \mathcal{L}(\boldsymbol{\theta})$.

The PL inequality is satisfied by all strongly convex functions, i.e., functions satisfying condition (2.11). However, this inequality is more general and can also be satisfied by some non-convex functions, e.g., see [6]. The following theorem considers the case when the loss function $\mathcal{L}$ satisfies Assumption 3.3 and characterizes the rate of change in the values of the loss function over the sequence of the models generated by the DMGD method.

Theorem 3.2 *Suppose Assumptions 3.1, 3.2, and 3.3. Consider an arbitrary iteration t of the DMGD algorithm with batch-size $b \in [1, m)$ and learning rate $\gamma_t \leq \frac{1}{L}$. Then, the following holds true:*

$$\mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta_{t+1}}) - \mathcal{L}^*\right] \leq (1 - \mu\gamma_t)\,\mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta_t}) - \mathcal{L}^*\right] \; + \; \frac{L\sigma^2}{2nb}\gamma_t^2\,. \tag{3.18}$$

Proof Similar to (3.14) in the proof of Theorem 3.1, we obtain that

$$\mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta_{t+1}})\right] \leq \mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta_t})\right] - \gamma_t\left(1 - \frac{L}{2}\,\gamma_t\right)\mathbb{E}\left[\|\nabla\mathcal{L}(\boldsymbol{\theta_t})\|^2\right] + \frac{L\sigma^2}{2nb}\,\gamma_t^2\,.$$

Let us denote $U_t \coloneqq \mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta_t}) - \mathcal{L}^*\right]$. Then, the above is equivalently written as

$$U_{t+1} \leq U_t - \gamma_t\left(1 - \frac{L}{2}\,\gamma_t\right)\mathbb{E}\left[\|\nabla\mathcal{L}(\boldsymbol{\theta_t})\|^2\right] + \frac{L\sigma^2}{2nb}\,\gamma_t^2\,.$$

As $0 < \gamma_t \leq \frac{1}{L}$, we have $1 - \frac{L}{2}\gamma_t \geq \frac{1}{2}$. Using this in the above, we obtain that

$$U_{t+1} \leq U_t - \frac{\gamma_t}{2}\,\mathbb{E}\left[\|\nabla\mathcal{L}(\boldsymbol{\theta_t})\|^2\right] + \frac{L\sigma^2}{2nb}\,\gamma_t^2\,.$$

Due to Assumption 3.3, we have $\mathbb{E}\left[\|\nabla\mathcal{L}(\boldsymbol{\theta_t})\|^2\right] \geq \mathbb{E}\left[2\mu\left(\mathcal{L}(\boldsymbol{\theta_t}) - \mathcal{L}^*\right)\right] = 2\mu U_t$. Hence, from the above, we obtain that

$$U_{t+1} \leq (1 - \mu\gamma_t)\,U_t \; + \; \frac{L\sigma^2}{2nb}\,\gamma_t^2\,.$$

This concludes the proof. □

Theorem 3.2 characterizes the dynamics of the learning process when using DMGD on a strongly convex problem. The result suggests that for a well-chosen sequence of learning rates, the algorithm can converge *exponentially* fast. Let us first consider the case of constant learning rate (similarly to the non-convex case). Specifically, suppose that $\gamma_t = \gamma \leq \frac{1}{L}$ for all t. Note that, when Assumptions 3.1 and 3.3 are imposed simultaneously, we have $\mu \leq L$ (formal derivation for this observation is deferred to Appendix A.2). Therefore, when $\gamma < \frac{1}{L}$, we have

$(1 - \mu\gamma) \in [0, 1)$. Then, upon executing the DMGD algorithm for T iterations, applying Theorem 3.2 recursively from $t = T$ to $t = 1$, we get

$$\mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta}_{\mathbf{T+1}}) - \mathcal{L}^*\right] \leq (1 - \mu\gamma)^T \left(\mathcal{L}(\boldsymbol{\theta}_{\mathbf{1}}) - \mathcal{L}^*\right) + \sum_{k=0}^{T-1} (1 - \mu\gamma)^k \frac{L\sigma^2}{2nb}\gamma^2 . \tag{3.19}$$

As $(1 - \mu\gamma) \in [0, 1)$, the sum of the geometric series on the right-hand side of (3.19), i.e., $\sum_{k=0}^{T-1}(1 - \mu\gamma)^k$, is upper bounded by $\frac{1}{\mu\gamma}$. Using this in the above, we obtain that

$$\mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta}_{\mathbf{T+1}}) - \mathcal{L}^*\right] \leq (1 - \mu\gamma)^T \left(\mathcal{L}(\boldsymbol{\theta}_{\mathbf{1}}) - \mathcal{L}^*\right) \; + \; \frac{L\sigma^2}{2\mu nb}\gamma . \tag{3.20}$$

Although the above inequality provides a first characterization of the convergence rate of DMGD in the strongly convex case, we note that as T increases, the right-hand side of (3.20) does not go to 0, as might be expected. Specifically, we have a residual error of $\frac{L\sigma^2}{2\mu nb}$, even when the number of iterations T approaches infinity. This means that we cannot guarantee convergence of the algorithm to the optimal model $\boldsymbol{\theta}^* \coloneqq \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$. To circumvent this shortcoming, we use a more refined scheduling of the learning rates, derived with the help of Lemma 3.1 presented below. The proof of Lemma 3.1 can be found in Appendix B.

Lemma 3.1 *Let $\alpha_1, \alpha_2, \alpha_3$ be real values such that $\alpha_1 > 0$, $\alpha_2 \geq 0$, and $\alpha_3 \geq 0$. Let $T > 2$ be an even number. Consider two positive real-valued sequences $(\gamma_1, \ldots, \gamma_T)$ and $(U_1, \ldots, U_{T+1})$ such that for all $t \in [T]$,*

$$\gamma_t = \begin{cases} \gamma_o & , \; t < T/2 \\ \dfrac{2}{\alpha_1\left(s_o + t - \frac{T}{2}\right)} & , \; t \geq T/2 \end{cases}, \quad \text{and} \quad U_{t+1} \leq (1 - \alpha_1\gamma_t)\, U_t + \alpha_2\gamma_t^2 + \alpha_3\gamma_t ,$$

where γ_o and s_o are positive real values such that $\gamma_o = \frac{2}{\alpha_1 s_o} < \frac{1}{\alpha_1}$. Then,

$$U_{T+1} \leq (s_o - 1)^2 U_1 \frac{4e^{-\frac{\alpha_1\gamma_o}{2}(T-2)}}{T^2} + \frac{4(\alpha_2\gamma_o + \alpha_3)(s_o - 1)^2}{\alpha_1 T^2} + \frac{8\alpha_2}{\alpha_1^2 T} + \frac{2\alpha_3}{\alpha_1} .$$

We use Lemma 3.1 to define an appropriate sequence of learning rates for DMGD and combine that with the observation on the dynamics of DMGD in Theorem 3.2 to obtain the following result for the convergence rate of DMGD to the minimum point of the loss function.

Corollary 3.2 *Suppose Assumptions 3.1, 3.2, and 3.3. Let $T > 2$ be an even number. Consider T iterations of the DMGD algorithm with batch-size $b \in [1, m)$. Set the learning rate*

$$\gamma_t = \begin{cases} \gamma_o & , \ t < T/2 \\ \frac{2}{\mu\left(s_o + t - \frac{T}{2}\right)} & , \ t \geq T/2 \end{cases},$$

where γ_o and s_o are positive real values such that $\gamma_o = \frac{2}{\mu s_o} < \frac{1}{L}$. Then,

$$\mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta}_{\mathbf{T+1}}) - \mathcal{L}^*\right] \leq \Delta_o \frac{e^{-\frac{\mu\gamma_o}{2}(T-2)}}{T^2} + \frac{2K_{\mathcal{L}}\gamma_o(s_o-1)^2}{T^2}\frac{\sigma^2}{nb} + \frac{4K_{\mathcal{L}}}{\mu T}\frac{\sigma^2}{nb},$$

where Δ_o is a real value such that $4(s_o-1)^2\left(\mathcal{L}(\boldsymbol{\theta}_{\mathbf{1}}) - \mathcal{L}^*\right) \leq \Delta_o$ and $K_{\mathcal{L}} \coloneqq \frac{L}{\mu}$ is referred as the condition number of the loss function $\mathcal{L}$.

Proof As $\gamma_o < \frac{1}{L}$ and $\gamma_t \leq \gamma_o$ for all t, we have $\gamma_t \leq \gamma_0 < \frac{1}{L}$ for all $t \in [T]$. Therefore, owing to Theorem 3.2, for an arbitrary $t \in [T]$, we have

$$\mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta}_{\mathbf{t+1}}) - \mathcal{L}^*\right] \leq (1-\mu\gamma_t)\,\mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta}_{\mathbf{t}}) - \mathcal{L}^*\right] + \frac{L\sigma^2}{2nb}\gamma_t^2.$$

Setting $U_t \coloneqq \mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta}_{\mathbf{t}}) - \mathcal{L}^*\right]$, the above can be equivalently written as

$$U_{t+1} \leq (1-\mu\gamma_t)U_t + \frac{L\sigma^2}{2nb}\gamma_t^2. \tag{3.21}$$

At this point, we would like to invoke Lemma 3.1 to obtain a convergence result for the sequence $(U_t)_{t\in[T+1]}$. For doing so, we need to verify that all the conditions of the lemma hold true. We choose $\alpha_1 = \mu, \alpha_2 = \frac{L\sigma^2}{2nb}$, and $\alpha_3 = 0$. First, note that the condition for the sequence $(U_t)_{t\in[T+1]}$ in (3.21) reduces to that in Lemma 3.1. Second, since $\mu = \alpha_1$ and $\mu \leq L$ when Assumptions 3.1 and 3.3 hold true simultaneously (shown in Appendix A.2), we have

$$\gamma_o = \frac{2}{\mu s_o} = \frac{2}{\alpha_1 s_o} < \frac{1}{L} \leq \frac{1}{\mu} = \frac{1}{\alpha_1}.$$

(continued)

Therefore, the sequence $(\gamma_t)_{t\in[T]}$ also satisfies the respective condition of Lemma 3.1. Hence, upon substituting the above values of $\alpha_1, \alpha_2, \alpha_3$ in Lemma 3.1, we obtain that

$$U_{T+1} \leq (s_o-1)^2 U_1 \frac{4e^{-\frac{\mu\gamma_o}{2}(T-2)}}{T^2} + \frac{4L\sigma^2\gamma_o(s_o-1)^2}{2nb\mu T^2} + \frac{8L\sigma^2}{2nb\mu^2 T}.$$

From the above, upon recalling that $4(s_o-1)^2U_1 = 4(s_o-1)^2\left(\mathcal{L}(\boldsymbol{\theta_1})-\mathcal{L}^*\right) \leq \Delta_o$ and $\frac{L}{\mu} = K_{\mathcal{L}}$, and simplifying the fractions, we obtain that

$$\mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta_{T+1}})-\mathcal{L}^*\right] \leq \Delta_0 \frac{e^{-\frac{\mu\gamma_o}{2}(T-2)}}{T^2} + \frac{2K_{\mathcal{L}}\gamma_o(s_o-1)^2}{T^2}\frac{\sigma^2}{nb} + \frac{4K_{\mathcal{L}}}{\mu T}\frac{\sigma^2}{nb}.$$

This concludes the proof. □

From Corollary 3.2, we can draw similar conclusions as in the non-convex case. The suboptimality error rate of DMGD is equivalent to that of centralized mini-batch GD with batch-size nb, but with increased computational efficiency. Specifically, after completing T iterations of DMGD, the server produces a model $\boldsymbol{\theta_{T+1}}$ such that

$$\mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta_{T+1}})-\mathcal{L}^*\right] \in O_T\left(K_{\mathcal{L}}\frac{\sigma^2}{nb}\frac{1}{T}\right).$$

Remark 3.3 While we could also use a simpler diminishing learning rate of $\gamma_t := \frac{1}{t}$, it yields a convergence rate that is in $O_T\left(\frac{\sigma^2}{nb}\frac{\log T}{T}\right)$, which is not tight in T. The scheduled learning rate, in Corollary 3.2, helps eliminate the logarithmic factor.

3.2.4 Analyzing Gradient and Stochastic Gradient Descents

Now that we have analyzed the convergence of DMGD, both in the non-convex and the strongly convex setting, we present the extension of these results to analyze other gradient-descent methods, specifically distributed gradient descent and distributed stochastic gradient descent.

Distributed SGD

Adapting the analysis of DMGD to distributed SGD (DSGD) is straightforward. This is because DSGD is equivalent to DMGD with batch-size $b = 1$. Hence, to

derive the convergence of DSGD, it suffices to take the results in Corollaries 3.1 and 3.2 and set $b = 1$. With this substitution, we obtain the following result.

Corollary 3.3 *Suppose Assumptions 3.1 and 3.2. Consider T iterations of the DSGD algorithm. Then, the following assertions hold true:*

1. ***Non-convex case:*** *There exists a sequence of learning rates $(\gamma_t)_{t\in[T]}$ for which, after T iterations, we have*

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}(\boldsymbol{\theta_t})\|^2\right] \in O_T\left(\sqrt{\frac{\sigma^2}{nT}}\right).$$

2. ***Strongly convex case:*** *Suppose Assumption 3.3 as well and that T is an even number greater than 2. Then, there exists a sequence of learning rates $(\gamma_t)_{t\in[T]}$ for which after T iterations, we have*

$$\mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta_{T+1}}) - \mathcal{L}^*\right] \in O_T\left(K_{\mathcal{L}}\frac{\sigma^2}{nT}\right).$$

Proof The proof immediately follows from Corollaries 3.1 and 3.2, after noting that DSGD is equivalently defined as DMGD with $b = 1$. The exact forms of the learning rates can be respectively obtained from these corollaries. □

Distributed GD

For distributed GD (DGD), we can make a similar analysis, by noting that DGD is simply DMGD with $b = m$. Specifically, for all $i \in [n]$ and $t \in [T]$, we have $\mathbf{g_t^{(i)}} = \nabla\mathcal{L}_i(\boldsymbol{\theta_t})$. Note, however, that we purposely avoided to present a result when $b = m$ in Corollaries 3.2 and 3.1. This is because, when $b = m$, the gradient estimate at each iteration of the algorithm is not stochastic anymore, which is equivalent to setting $\sigma^2 = 0$ in the analysis of DMGD. Then, we can easily derive the convergence rate for DGD that considerably improves upon DMGD for which $b < m$. While we only state the orders of the convergence rates in the statement of the corollary below, the reader can find the precise bounds (along with some more details) in the proof.

Corollary 3.4 *Suppose Assumptions 3.1 and 3.2. Consider T iterations of the DGD algorithm. If $\gamma_t = \gamma \leq \frac{1}{L}$ for all $t \in [T]$, then the following assertions hold true:*

1. ***Non-convex case:*** *We have*

$$\frac{1}{T}\sum_{t=1}^{T}\|\nabla\mathcal{L}(\boldsymbol{\theta_t})\|^2 \in O_T\left(\frac{1}{\gamma T}\right).$$

2. ***Strongly convex case:*** *Under Assumption 3.3, we have*

$$\mathcal{L}(\boldsymbol{\theta}_{\mathbf{T+1}}) - \mathcal{L}^* \in O_T\left(e^{-\mu\gamma T}\right).$$

Substituting $\gamma = \frac{1}{L}$ *above, we obtain the best possible convergence guarantee, i.e.,* $\mathcal{L}(\boldsymbol{\theta}_{\mathbf{T+1}}) - \mathcal{L}^* \in O_T\left(e^{-\frac{1}{K_{\mathcal{L}}}T}\right)$. *Recall that* $K_{\mathcal{L}} := \frac{L}{\mu}$ *is the condition number of the loss function* $\mathcal{L}$.

Proof We divide this proof into two parts. In the first part, we analyze the non-convex case. In the second part, we consider the strongly convex case:

1. **Non-convex case:** Consider an arbitrary $t \in [T]$. Similar to (3.8) in the proof of Theorem 3.1, noting that $\boldsymbol{\theta}_{\mathbf{t+1}} = \boldsymbol{\theta}_{\mathbf{t}} - \gamma\mathcal{L}(\boldsymbol{\theta}_{\mathbf{t}})$, we obtain that

$$\mathcal{L}(\boldsymbol{\theta}_{\mathbf{t+1}}) \leq \mathcal{L}(\boldsymbol{\theta}_{\mathbf{t}}) - \gamma\langle\nabla\mathcal{L}(\boldsymbol{\theta}_{\mathbf{t}}),\, \nabla\mathcal{L}(\boldsymbol{\theta}_{\mathbf{t}})\rangle + \frac{L}{2}\gamma^2\,\|\mathcal{L}(\boldsymbol{\theta}_{\mathbf{t}})\|^2 .$$

Rearranging the terms on the right-hand side yields the following:

$$\mathcal{L}(\boldsymbol{\theta}_{\mathbf{t+1}}) \leq \mathcal{L}(\boldsymbol{\theta}_{\mathbf{t}}) - \gamma\left(1 - \frac{L}{2}\gamma\right)\|\nabla\mathcal{L}(\boldsymbol{\theta}_{\mathbf{t}})\|^2 , \tag{3.22}$$

which can equivalently be written as

$$\left(1 - \frac{L}{2}\gamma\right)\|\nabla\mathcal{L}(\boldsymbol{\theta}_{\mathbf{t}})\|^2 \leq \frac{\mathcal{L}(\boldsymbol{\theta}_{\mathbf{t}}) - \mathcal{L}(\boldsymbol{\theta}_{\mathbf{t+1}})}{\gamma} .$$

As $\gamma \leq \frac{1}{L}$, we have $1 - \frac{L}{2}\gamma \geq \frac{1}{2}$. Using this in the above, and then summing both sides from $t = 1$ to $t = T$, we obtain that

$$\sum_{t=1}^{T}\|\nabla\mathcal{L}(\boldsymbol{\theta}_{\mathbf{t}})\|^2 \leq \frac{2\left(\mathcal{L}(\boldsymbol{\theta}_{\mathbf{1}}) - \mathcal{L}(\boldsymbol{\theta}_{\mathbf{T+1}})\right)}{\gamma} .$$

Dividing by T on both sides, we finally get

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}(\boldsymbol{\theta}_{\mathbf{t}})\|^2\right] \leq \frac{2\left(\mathcal{L}(\boldsymbol{\theta}_{\mathbf{1}}) - \mathcal{L}(\boldsymbol{\theta}_{\mathbf{T+1}})\right)}{\gamma T} \leq \frac{2\left(\mathcal{L}(\boldsymbol{\theta}_{\mathbf{1}}) - \mathcal{L}^*\right)}{\gamma T} .$$

(continued)

The last inequality follows from the fact that $\mathcal{L}(\boldsymbol{\theta}_{\mathbf{T+1}}) - \mathcal{L}^* \geq 0$. This concludes the proof for the non-convex case.

2. **Strongly convex case:** Recall, from (3.22) above, that for all $t \in [T]$,

$$\mathcal{L}(\boldsymbol{\theta}_{\mathbf{t+1}}) \leq \mathcal{L}(\boldsymbol{\theta}_{\mathbf{t}}) - \gamma \left(1 - \frac{L}{2}\gamma\right) \|\nabla \mathcal{L}(\boldsymbol{\theta}_{\mathbf{t}})\|^2 \ .$$

Under Assumption 3.3, $\|\nabla \mathcal{L}(\boldsymbol{\theta}_{\mathbf{t}})\|^2 \geq 2\mu \left(\mathcal{L}(\boldsymbol{\theta}_{\mathbf{t}}) - \mathcal{L}^*\right)$. Using this in the above, and also recalling that $1 - \frac{L}{2}\gamma \geq \frac{1}{2}$, yields

$$\mathcal{L}(\boldsymbol{\theta}_{\mathbf{t+1}}) \leq \mathcal{L}(\boldsymbol{\theta}_{\mathbf{t}}) - \gamma\mu \left(\mathcal{L}(\boldsymbol{\theta}_{\mathbf{t}}) - \mathcal{L}^*\right) \ .$$

Upon subtracting $\mathcal{L}^*$ from both sides in the above, we obtain that

$$\mathcal{L}(\boldsymbol{\theta}_{\mathbf{t+1}}) - \mathcal{L}^* \leq (1 - \mu\gamma) \left(\mathcal{L}(\boldsymbol{\theta}_{\mathbf{t}}) - \mathcal{L}^*\right) \ .$$

Recall that $\mu \leq L$ when Assumptions 3.1 and 3.3 hold true simultaneously. This implies that, as $\gamma \leq \frac{1}{L}$, $1 - \mu\gamma \geq 1 - \frac{\mu}{L} \geq 0$. Therefore, upon applying the above inequality recursively from $t = T$ to $t = 1$, we obtain that

$$\mathcal{L}(\boldsymbol{\theta}_{\mathbf{T+1}}) - \mathcal{L}^* \leq (1 - \mu\gamma)^T \left(\mathcal{L}(\boldsymbol{\theta}_{\mathbf{1}}) - \mathcal{L}^*\right) \ .$$

As $(1 - \mu\gamma)^T \leq e^{-\mu\gamma T}$, the above implies that

$$\mathcal{L}(\boldsymbol{\theta}_{\mathbf{T+1}}) - \mathcal{L}^* \leq e^{-\mu\gamma T} \left(\mathcal{L}(\boldsymbol{\theta}_{\mathbf{1}}) - \mathcal{L}^*\right) \ .$$

The above concludes the proof.

Remark 3.4 For the bounding factor $e^{-\mu\gamma T}$ to attain the lowest possible value, for a given T, we must use γ to be as large as possible, i.e., $\gamma = \frac{1}{L}$. The resulting convergence rate is then given by $\exp\left(-\frac{T}{K_{\mathcal{L}}}\right)$, where recall that $K_{\mathcal{L}} := \frac{L}{\mu}$ is the condition number of the loss function $\mathcal{L}$. Expectedly, we observe that smaller the condition number faster the gradient-descent method can converge to the minimum, and vice versa.

□

Note that, unlike DSGD and DMGD, the convergence rate of DGD is identical to that of its centralized counterpart, i.e., we do not have an acceleration when using multiple nodes. This comes from the fact that, at every step in DGD, each node computes, and sends to the server, the full gradient of its local loss function. Hence, we do not obtain the same "variance reduction" property as in (3.13). However, DGD still reduces the computational burden per node compared to the centralized

GD. In particular, at every step, each node computes only m point-wise gradient locally, instead of mn, if the data points were collected at a single machine. Thus, the computational cost per node is indeed divided by n.

3.3 Chapter Notes

We have presented in this chapter the framework of (server-based) federated machine learning and have introduced the prominent methods for learning in a federated manner, i.e., distributed gradient-descent methods. Specifically, we have explained the underlying principle of mini-batch distributed gradient descent by analyzing in detail the cases of non-convex and strongly convex loss functions. We have derived training errors in both cases and have presented mathematical tools that lay the foundation for further analysis of gradient-descent methods. Most of the proofs that we have presented in this chapter are standard and can essentially be considered as simplified versions of the results one may find in [13]. Furthermore, the convergence analysis we have presented under strong convexity, using a scheduled learning rate, is inspired by the results presented in [7, 15]. The training errors for the different methods that we discussed in this chapter are summarized below in Table 3.1.

Although the analysis that we have presented in this chapter is self-contained and sufficient to introduce the concepts useful for the subsequent chapters in this book, it does not constitute a complete introduction to the fields of distributed algorithms and distributed optimization. For a more comprehensive treatment of these topics, the interested reader can refer to [3, 10] and [1, 2], respectively. It is also important to notice that, for pedagogical reasons, we have presented only simple distributed gradient-descent methods. However, federated machine learning is a versatile new paradigm that is growing rapidly and can offer a number of other solutions, including *local updates*, *node sub-sampling,* or *adaptive methods*. We refer the interested reader to [5] for a good overview of the field, including open problems and possible future advances. Finally, we chose to not cover the analysis of *decentralized* gradient-descent methods on a peer-to-peer network. Although the

Table 3.1 Convergence rates for DGD, DSGD, and DMGD (with batch-size b). These hold for a class of learning problems where $\mathcal{L}$ is L-smooth, assuming that the point-wise gradients have bounded (local) covariance trace of σ^2. Recall that, in the strongly convex case, $K_{\mathcal{L}} := \frac{L}{\mu}$ denotes the condition number of the loss function $\mathcal{L}$

	DGD	DSGD	DMGD
Non-convex	$\mathcal{O}_T\left(\frac{1}{T}\right)$	$\mathcal{O}_T\left(\sqrt{\frac{\sigma^2}{nT}}\right)$	$\mathcal{O}_T\left(\sqrt{\frac{\sigma^2}{nbT}}\right)$
μ-Strongly convex	$\mathcal{O}_T\left(e^{-\frac{1}{K_{\mathcal{L}}}T}\right)$	$\mathcal{O}_T\left(K_{\mathcal{L}}\frac{\sigma^2}{nT}\right)$	$\mathcal{O}_T\left(K_{\mathcal{L}}\frac{\sigma^2}{nbT}\right)$

analysis of decentralized algorithms on peer-to-peer architecture shares important similarities with the federated algorithms we consider, these have some important peculiarities, especially regarding the impact of *network topology* and *asynchronous communications* on the training error. In addition to the references mentioned above, we would also like to point out some notable work on the topic of *decentralized optimization*, specifically [8, 9, 12, 14].

The distributed gradient-descent methods that we have presented in this chapter provide a concrete foundation for the robustness schemes (introduced in the subsequent chapters) in the context of federated machine learning. We provide some references on works that address robustness in other variants of distributed gradient descent that involve multiple local updates and node sub-sampling (such as in *FederatedAveraging* [11]) and peer-to-peer architecture, at the end of Chap. 4.

References

1. Bertsekas D, Tsitsiklis J (2015) Parallel and distributed computation: numerical methods. Athena Scientific, Nashua
2. Boyd S, Parikh N, Chu E, Peleato B, Eckstein J (2011) Distributed optimization and statistical learning via the alternating direction method of multipliers. Found Trends Mach Learn 3(1):1–122. ISSN: 1935-8237
3. Cachin C, Guerraoui R, Rodrigues L (2011) Introduction to reliable and secure distributed programming. Springer Science & Business Media, Cham
4. Freund JE (1962) Mathematical statistics. Prentice-Hall, Englewood Cliffs
5. Kairouz P, McMahan HB, Avent B, Bellet A, Bennis M, Bhagoji AN, Bonawitz K, Charles Z, Cormode G, Cummings R, D'Oliveira RGL, Eichner H, Rouayheb SE, Evans D, Gardner J, Garrett Z, Gascón A, Ghazi B, Gibbons PB, Gruteser M, Harchaoui Z, He C, He L, Huo Z, Hutchinson B, Hsu J, Jaggi M, Javidi T, Joshi G, Khodak M, Konecný J, Korolova A, Koushanfar F, Koyejo S, Lepoint T, Liu Y, Mittal P, Mohri M, Nock R, Özgür A, Pagh R, Qi H, Ramage D, Raskar R, Raykova M, Song D, Song W, Stich SU, Sun Z, Suresh AT, Tramèr F, Vepakomma P, Wang J, Xiong L, Xu Z, Yang Q, Yu FX, Yu H, Zhao S (2021) Advances and open problems in federated learning. Found Trends®Mach Learn 14(1–2):1–210. ISSN: 1935-8237
6. Karimi H, Nutini J, Schmidt M (2016) Linear convergence of gradient and proximal-gradient methods under the Polyak-Lojasiewicz condition. In: Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2016, Riva del Garda, Italy, September 19–23, 2016, Proceedings, Part I 16. Springer, Berlin, pp. 795–811
7. Khaled A, Richtárik P (2023) Better theory for SGD in the nonconvex world. Trans Mach Learn Res . Survey Certification. ISSN: 2835-8856. https://openreview.net/forum?id=AU4qHN2VkS
8. Koloskova A, Loizou N, Boreiri S, Jaggi M, Stich SU (2020) A unified theory of decentralized SGD with changing topology and local updates. In: Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13–18 July 2020, Virtual Event, vol. 119. Proceedings of Machine Learning Research. PMLR, pp. 5381–5393
9. Lian X, Zhang W, Zhang C, Liu J (2018) Asynchronous decentralized parallel stochastic gradient descent. In: ICML, pp. 3049–3058
10. Lynch NA (1996) Distributed algorithms. Elsevier, Amsterdam
11. McMahan B, Moore E, Ramage D, Hampson S, Arcas BAY (2017) Communication-efficient learning of deep networks from decentralized data. In: Proceedings of the 20th International Conference on Artificial Intelligence and Statistics. Singh A, Zhu J (eds), vol. 54. Proceedings of Machine Learning Research. PMLR, pp. 1273–1282

12. Nedic A, Ozdaglar A (2009) Distributed subgradient methods for multi-agent optimization. IEEE Trans Autom Control 54(1):48–61
13. Nesterov Y et al (2018) Lectures on convex optimization, vol 137. Springer, Berlin
14. Scaman K, Bach F, Bubeck S, Lee YT, Massoulié L (2017) Optimal algorithms for smooth and strongly convex distributed optimization in networks. In: Proceedings of the 34th International Conference on Machine Learning. Precup D, Teh YW (eds), vol 70. Proceedings of Machine Learning Research. PMLR, pp 3027–3036
15. Stich SU (2019) Unified optimal analysis of the (stochastic) gradient method. In: arXiv preprint arXiv:1907.04232

Chapter 4
Fundamentals of Robust Machine Learning

Abstract As explained in the previous chapter, distributing the learning procedure significantly simplifies the task of training complex models on a large amount of data. The workload for each node is divided essentially by the total size of the network, while the nodes retain the control over their local data. However, the perks of federated machine learning rest upon the unrealistic assumption that each node *correctly* executes the prescribed algorithm and all its data are trustworthy. This assumption need not hold true in practice. In this chapter, we revisit the problem of federated machine learning in the case when some of the participating nodes *deviate* from the set of instructions prescribed to them. As discussed in the introduction of this book, this deviation can be caused by bad data, software bugs, hardware failures, or even malicious attackers controlling some of the nodes. Under the presence of such *adversarial* nodes, traditional distributed-learning methods fail to guarantee good accuracy. We explain in this chapter how the traditional server-based gradient-descent methods can be rendered robust against a *minority* of adversarial nodes, and we analyze the *training error* of the resulting *robust gradient-descent* algorithm.

4.1 Getting Started with Robustness

Due to the presence of multiple nodes running independent operations, federated machine learning is prone to having misbehaving nodes. In the parlance of distributed computing, misbehaving (hereafter called *adversarial*) nodes that behave arbitrarily are commonly referred to as *Byzantine*. The distributed gradient-descent methods we presented until now are not robust to adversarial nodes. In fact, a single adversarial node can significantly compromise the efficacy of the learning procedure by sending erroneous local gradients (or updates) to the server.

R. Guerraoui et al., *Robust Machine Learning*, Machine Learning: Foundations, Methodologies, and Applications, https://doi.org/10.1007/978-981-97-0688-4_4

4.1.1 Fragility of Distributed Gradient Descent

The main point of vulnerability of distributed gradient-descent methods is the *averaging operation* at the server in the global phase (see Algorithm 4). Basically, if a node $i \in [n]$ is adversarial, then the estimate of the local gradient $\mathbf{g}_t^{(i)}$ sent by node i to the server in the global phase of the algorithm need not be a good representation of its true local gradient. When $\mathbf{g}_t^{(i)}$ is not computed as per the algorithm originally prescribed, the learning procedure could be compromised. Indeed, in the worst-case scenario, a single adversarial node can arbitrarily manipulate the outcome of the averaging operation in the global phase, which could trivially prevent the server from reaching a good machine learning model, as illustrated in Fig. 4.1. Note that while in Fig. 4.1 we illustrate the fragility of distributed gradient descent considering a small system of only 3 nodes, in practice we could have hundreds (or even thousands) of nodes. Surprisingly, even in a system comprising a large number of nodes, it is sufficient for a single node to be adversarial to severely degrade the performance of conventional distributed gradient descent.

The very fact that the learning procedure can be arbitrarily manipulated in the face of an attack from a malicious entity (as depicted in Fig. 4.1), which can either poison the data of a node or take complete control of the node, is quite worrisome.

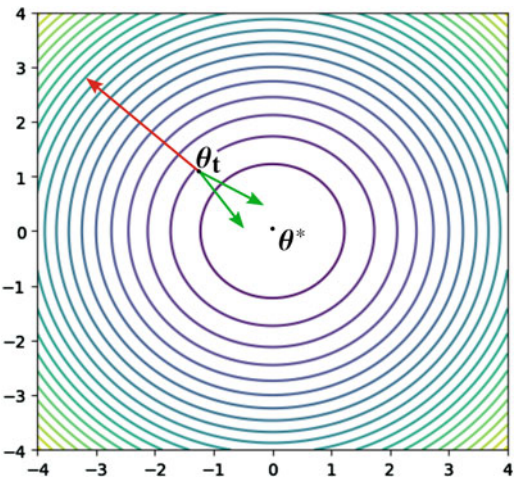

Fig. 4.1 Consider a server-based system architecture comprising 3 nodes. Suppose that node 3 is adversarial. If the server implements the distributed mini-batch GD (DMGD) algorithm presented in Chap. 3, then at each iteration it computes an update by averaging the gradients it receives. At any iteration t, if the adversarial node 3 sends $\mathbf{g}_t^{(3)} = -\left(\mathbf{g}_t^{(1)} + \mathbf{g}_t^{(2)}\right)$, instead of following the procedure defined in (3.3), then node 3 can entirely prevent the server from updating the model

Although the worst-case scenario is not necessarily the one that occurs most often with software or hardware bugs, analyzing the correctness of a machine learning scheme in the worst case highlights the catastrophic impact that an unfortunate event could have on the learning procedure. Arguably, even a random shift in the direction of the gradient can have a significant impact on the learning procedure. It has been documented that in some cases even a simple *poisoning of local data*, e.g., random flipping of their labels, is sufficient to hinder learning. Refer the notes presented at the end of this chapter, in Sect. 4.5, for a list of various adversarial behaviors that have been considered so far. In this chapter, we present general robustness schemes that provably protect the efficacy of distributed gradient descent against a fraction of adversarial nodes that can behave *arbitrarily bad*, including data poisoning and even model poisoning attacks.

An essential step in improving the robustness of a distributed gradient-descent method is to replace the simple averaging rule with a more sophisticated aggregation rule that would protect the model updates from incorrect gradients that could be sent by adversarial nodes. In the next section, we present a natural extension of the distributed mini-batch GD (DMGD) algorithm that replaces the averaging operation of the global phase with a so-called *robust aggregation rule*.

Remark 4.1 While the robustness techniques presented subsequently in this chapter (and Chap. 5) are analyzed in the context of *distributed* gradient descent, the techniques are also relevant in the conventional *centralized* machine learning setting. In particular, we could replace the classic arithmetic averaging by a robust aggregation, presented shortly in Sect. 4.1.3, to compute the gradients in each iteration of the mini-batch GD method (see Algorithm 3). This would prevent the machine learning algorithm from becoming corrupted due to poisonous training samples that might have been collected from unreliable sources.

4.1.2 Robustifying Distributed Gradient Descent

As discussed above, to impart robustness to DMGD, it is quite natural to replace the averaging operation in the global phase by a *robust aggregation rule* that can diminish any detrimental effects of “incorrect gradients" that might be sent by the adversarial nodes. What classifies as an incorrect gradient is, however, a difficult question to answer, especially when the *honest* (i.e., non-adversarial) nodes hold different training samples. The analysis of how the adversarial nodes influence the learning procedure is deferred to Sect. 4.4.

The robust implementation of DMGD, or *Robust DMGD*, relies on adapting the global phase of the algorithm by introducing a general aggregation function $F : \left(\mathbb{R}^d\right)^n \to \mathbb{R}^d$ for merging the gradients. Note that, in general, the number of vectors

we aggregate need not be fixed to be n; rather it equals the number of gradients the server chooses to aggregate at any given iteration. Furthermore, as the indices can be assigned arbitrarily to the nodes, the output of an aggregation rule is usually oblivious to the order of the input vectors $\mathbf{g_t^{(1)}}, \ldots, \mathbf{g_t^{(n)}}$. In other words, for any permutation mapping $\tau : [n] \to [n]$, we have

$$F\left(\mathbf{g_t^{(1)}}, \ldots, \mathbf{g_t^{(n)}}\right) = F\left(\mathbf{g_t^{(\tau(1))}}, \ldots, \mathbf{g_t^{(\tau(n))}}\right). \tag{4.1}$$

Lastly, note that, because adversarial nodes might deviate from the algorithm, the instructions of Robust DMGD only apply to honest nodes. Throughout this chapter, we denote the set of honest nodes by $H \subseteq [n]$. It is important to note that the identity of H (i.e., the honest nodes) remains a priori unknown to the server; otherwise, the problem of robustness is rendered trivial. We present in Algorithm 5 the schematic of Robust DMGD with a generic aggregation rule F.

Algorithm 5 Robust DMGD

The server initializes the procedure by choosing an initial model $\boldsymbol{\theta_1} \in \mathbb{R}^d$, which can be done in a manner similar to the DMGD method described in Chap. 3. In each iteration $t \in \{1, 2, \ldots\}$, the server maintains a model parameter $\boldsymbol{\theta_t}$ that is broadcast to all the nodes. The rest of the iteration is as follows:

1. **Local Phase.** Each **honest** node $i \in H$ independently and randomly samples a set of b data points, denoted by $S_t^{(i)}$, from the local training sample $\mathcal{S}_i$ (without replacement), and computes the local gradient estimate

$$\mathbf{g_t^{(i)}} \coloneqq \frac{1}{b} \sum_{(\mathbf{x}, y) \in S_t^{(i)}} \nabla_{\boldsymbol{\theta}} \ell\big(h_{\boldsymbol{\theta_t}}(\mathbf{x}), y\big). \tag{4.2}$$

2. **Global Phase.** Each node i returns to the server a gradient $\mathbf{g_t^{(i)}}$. If node i is **honest**, then $\mathbf{g_t^{(i)}}$ is computed as per (4.2). Otherwise, $\mathbf{g_t^{(i)}}$ can be an arbitrary vector in $\mathbb{R}^d$. Upon receiving the gradients from all the nodes, the server computes their aggregate

$$\widehat{\mathbf{g_t}} \coloneqq F\left(\mathbf{g_t^{(1)}}, \ldots, \mathbf{g_t^{(n)}}\right). \tag{4.3}$$

   Using the aggregate gradient $\widehat{\mathbf{g_t}}$, the server updates the current model $\boldsymbol{\theta_t}$ to

$$\boldsymbol{\theta_{t+1}} \coloneqq \boldsymbol{\theta_t} - \gamma_t \widehat{\mathbf{g_t}},$$

   where γ_t is the learning rate at iteration t.

Robust DGD and DSGD
In the remainder of this book, we consider Robust DMGD as the representative of all three gradient-descent methods presented in the previous chapter. Recall that the only difference between these methods is their local phase, which remains unchanged in the robust implementation. Therefore, to obtain a robust implementation of either DGD or DSGD, it suffices to replace their global phase with that of Algorithm 5. Essentially, the modification always consists in changing the averaging to another aggregation rule $F : \left(\mathbb{R}^d\right)^n \to \mathbb{R}^d$. Whenever suitable, we provide remarks on how to extend the results pertaining to DMGD to the other two methods. We present next in Sect. 4.1.3 some common robust aggregation rules.

4.1.3 Robust Aggregation

The idea behind a robust aggregation rule is to provide some insensitivity to changes in some input gradients. For example, consider the scalar case where we are given n (scalar) gradient values $v_1, \ldots, v_n \in \mathbb{R}$. Suppose that the aggregation rule, denoted F, is defined to be a linear combination of the input values, i.e.,

$$F(v_1, \ldots, v_n) = \sum_{i=1}^{n} a_i v_i, \text{ where } a_i > 0.$$

Then, as we just saw in the example given in Fig. 4.1, the outcome of the aggregate can be changed arbitrarily by changing a single gradient value. In other words, by changing v_i to v_i' for any i, the output of $F(v_1, \ldots, v_n)$ changes by $a_i(v_i - v_i')$ that is independent of the other gradients, hence can be made arbitrarily large. Now, suppose that F is defined to be the *median* of the input values, i.e.,

$$F(v_1, \ldots, v_n) = \text{median}(v_1, \ldots, v_n).$$

Then, even if we change v_i to v_i', the change in the output of F is bounded by the maximum absolute difference between any two unchanged input values. This observation holds regardless of the magnitude of the change we make to v_i. In particular, this means that the median cannot be arbitrarily manipulated by a single input. It is thus more robust than a mere linear combination of the inputs. To seek *more robustness*, let us consider F to be the *trimmed mean* of the input values.[1] The trimmed mean is a standard robust adaptation of averaging, based on outlier filtering. Given n scalar values $v_1, \ldots, v_n \in \mathbb{R}$, and a *trimming parameter* $q < \frac{n}{2}$,

[1] The quality of robustness achieved by an aggregation rule, which gives substance to the notion of *more robustness* (or less robustness), is formalized later in Sect. 4.3.

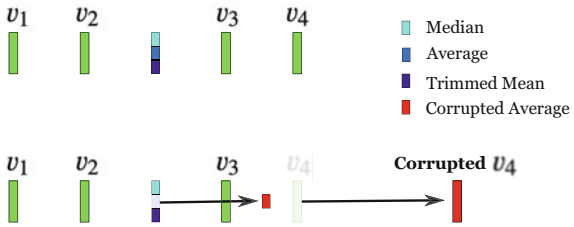

Fig. 4.2 Illustration of the value output by average, median, and trimmed mean (with $q = 1$) over a set of four scalar values, before and after corrupting a value. In the first row, all three aggregations are identical. In the second row, one of the input values (specifically, v_4) is corrupted and shifts to the right, while the remaining three values remain unchanged. Although the values of median and trimmed mean have not changed, the average has shifted to the right, moving away from the majority of the uncorrupted values. Arguably, after corruption, the median and trimmed mean provide a better representation of the set of uncorrupted values

the trimmed mean, denoted by TM_q, is defined to be

$$\mathrm{TM}_q\left(v_1, \ldots, v_n\right) = \frac{1}{n-2q} \sum_{i=q+1}^{n-q} v_{(i)}, \tag{4.4}$$

where $v_{(i)}, \ldots, v_{(n)}$ denote the order statistics of $v_1, \ldots, v_n$, i.e., the sorting of the values in non-decreasing order (with ties broken arbitrarily).[2] In essence, TM_q computes the average of the $n - 2q$ values that remain after eliminating the q largest and q smallest values. Similar to the median, if we change an input v_i to v_i', the change in the final value of the aggregation is also bounded by the maximum absolute difference between any two unchanged input values. The change in trimmed mean is further reduced (compared to median) by a factor of $n - 2q$, thanks to the averaging operation after the elimination of extreme values. A trimmed mean is thus clearly less vulnerable to manipulation than standard averaging. We illustrate this phenomenon in Fig. 4.2.

The driving principle behind the design of a robust aggregation is to reproduce for high-dimensional vectors the insensitivity to the changes we just discussed in the scalar case. For now, we defer a formalism of this driving principle to Sect. 4.3, and we list below some commonly used robust aggregation rules, specifically *coordinate-wise median* (CWMed), *geometric median* (GeoMed), *coordinate-wise trimmed mean* (CWTM), and *(Multi-)Krum*. We do not attempt to provide an exhaustive list of methods. Instead, we present some notable schemes that were analyzed by several works in recent years; these are therefore good representatives of the trends and the advancements in this field of robust federated machine learning. As we would see later in Sect. 4.3, these aggregation rules are "robust," as long as

[2] More formally, let τ be the permutation on $[n]$ such that $v_{\tau(1)} \leq \cdots \leq v_{\tau(n)}$. The i-th order statistic of $v_1, \ldots, v_n$ is simply $v_{(i)} = v_{\tau(i)}$ for all $i \in [n]$.

we assume that a majority of the nodes in the system are honest. Below, we consider a given set of n input vectors $\mathbf{v_1}, \ldots, \mathbf{v_n}$ from $\mathbb{R}^d$ and define our aggregation rule on this set. Recall that for a vector $\mathbf{v}$, $[\mathbf{v}]_k$ denotes its k-th element.

Coordinate-Wise Median (CWMed)
This aggregation rule is a straightforward vector implementation of the scalar median operator. Specifically, the coordinate-wise median CWMed $(\mathbf{v_1}, \ldots, \mathbf{v_n})$ is a vector in $\mathbb{R}^d$ whose k-th element, for all $k \in [d]$, is given by

$$[\text{CWMed}\,(\mathbf{v_1}, \ldots, \mathbf{v_n})]_k = \text{median}\,([\mathbf{v_1}]_k, \ldots, [\mathbf{v_n}]_k)\ . \tag{4.5}$$

Clearly, in the scalar case, i.e., when $d = 1$, the coordinate-wise median reduces to the classic median that is more robust than the averaging (as we just discussed). Computing the coordinate-wise median is very time-efficient. Indeed, by using the *BFPRT algorithm* [8], a median for each coordinate can be computed in a linear amount of operations (with respect to n). Accordingly, CWMed has an overall time complexity $O_n(nd)$, which matches the time complexity of computing the average of d-dimensional vectors.

Geometric Median (GeoMed)
This aggregation rule is a generalization of median to high-dimensional spaces. Specifically, the geometric median of a set of input vectors is defined to be a vector that minimizes the sum of the Euclidean distances to the input vectors, i.e.,

$$\text{GeoMed}\,(\mathbf{v_1}, \ldots, \mathbf{v_n}) \in \underset{\mathbf{v}\in\mathbb{R}^d}{\text{argmin}} \sum_{i=1}^{n} \|\mathbf{v} - \mathbf{v_i}\|\ . \tag{4.6}$$

In the special case of scalar inputs, (i) when n is odd, the traditional median is equivalent to the geometric median, and (ii) when n is even, the geometric median need not be unique, but the traditional median is a valid geometric median. This means that, in the scalar case, the geometric median and the coordinate-wise median basically coincide. But, when $d > 1$, this equivalence need not hold anymore. In the general high-dimensional space of $\mathbb{R}^d$, the geometric median is the Euclidean *barycenter* of the input vectors. However, the coordinate-wise median does not enjoy this geometric property. Hence, we can expect geometric median to have better robustness (against input corruptions) than coordinate-wise median. More details on this comparison are presented later in Sect. 4.3.

Remark 4.2 Although the geometric median has a reasonable robustness property (against input corruptions), in general, there exists no closed-form solution for this operation. Existing methods compute an approximation of geometric median using iterative schemes, e.g., *the Weiszfeld's method*. We refer the interested reader to [11] for more discussion on efficient approximation methods for geometric median.

Coordinate-Wise Trimmed Mean (CWTM)
This aggregation rule is a vector implementation of the trimmed mean rule, which we discussed earlier for scalars. Given a non-negative integer $q < \frac{n}{2}$ (a.k.a. the trimming parameter), $\text{CWTM}_q(\mathbf{v_1}, \ldots, \mathbf{v_n})$ is defined to be a vector whose k-th element, for all $k \in [d]$, is given by

$$\left[\text{CWTM}_q(\mathbf{v_1}, \ldots, \mathbf{v_n})\right]_k = \text{TM}_q([\mathbf{v_1}]_k, \ldots, [\mathbf{v_n}]_k) , \tag{4.7}$$

where the mapping TM_q is as defined earlier in (4.4). Clearly, in the scalar case, the coordinate-wise trimmed mean reduces to the classic trimmed mean. When $q = \lceil n/2 \rceil - 1$, a coordinate-wise trimmed mean is equivalent to the coordinate-wise median operation. Similarly to the coordinate-wise median, this aggregation rule is quite efficient in terms of worst-case algorithmic complexity. The time complexity of sorting vectors in each dimension is in $O_n(n \log n)$ (e.g., using the *mergesort* algorithm), and averaging the selected value in each dimension is in $O_n((n-2q)d)$. Overall, the time complexity of the CWTM aggregation rule is in $O_n(dn \log n)$, which is only a $\log n$ factor more than that of the average.

Multi-Krum
This particular aggregation rule induces two parameters, $q_1 \in [n-2]$ and $q_2 \in [n-q_1]$. Multi-Krum consists in computing for each input vector a set of other input vectors that are the closest to the input vector. Specifically, for each $i \in [n]$ and $j \in [n-1]$, we denote by $i_j \in [n] \setminus \{i\}$ the index of the j-th closest input vector from $\mathbf{v_i}$, i.e., we have $\left\|\mathbf{v_i} - \mathbf{v_{i_1}}\right\| \leq \ldots \leq \left\|\mathbf{v_i} - \mathbf{v_{i_{n-1}}}\right\|$, with ties broken arbitrarily. We also define $S_i(q_1) := \{i_1, \ldots, i_{n-q_1-1}\}$. Using these notations, Multi-Krum can be defined in two steps. First, for each $i \in [n]$, compute the score

$$\text{Score}(i) = \sum_{j \in S_i(q_1)} \left\|\mathbf{v_i} - \mathbf{v_j}\right\|^2 .$$

Then, the output of the aggregation is defined to be the average of q_2 input vectors with the smallest scores, i.e.,

$$\text{Multi-Krum}_{q_1,q_2}(\mathbf{v_1}, \ldots, \mathbf{v_n}) = \frac{1}{q_2} \sum_{i \in Z(q_2)} \mathbf{v_i} , \tag{4.8}$$

where $Z(q_2) \subset [n]$ such that $|Z(q_2)| = q_2$ and for all $i \in Z(q_2)$ and $j \in [n] \setminus Z(q_2)$, $\text{Score}(i) \leq \text{Score}(j)$. Because Multi-Krum is not an exhaustive search, rather a greedy algorithm, it is not very expensive from a computational viewpoint. Indeed, the time complexity of computing all pair-wise distances between the vectors is in $O_n\left(dn^2\right)$, and the complexity of computing the score of each input vector is in $O_n(q_1 n \log(n))$. Lastly, the time complexity for sorting of the vectors according to their scores and the computation of the average over the q_2 smallest-scored vectors is in $O_n(n \log(n) + q_2 d)$. Overall, the time complexity of Multi-Krum is in $O_n\left(n^2 d + q_1 n \log(n)\right)$. Hence, although more expensive than coordinate-wise

methods, the computational cost of Multi-Krum need not be prohibitive. Note that when $q_2 = 1$, i.e., we output a vector with the smallest score, the resulting Multi-Krum rule is simply called Krum.

4.2 Measuring Robustness

We formalize in this section the objective of robustness in the context of server-based distributed machine learning. The formalism is not specific to Robust DMGD and can be applied to other algorithms. However, we later use this formalism to define key metrics for characterizing the robustness in the specific case of gradient-descent methods. We consider the following system setup.

System Setup
We consider a scenario where up to f out of n total nodes might be *adversarial*. Alternately, we aim to design a distributed algorithm that can resist up to f adversarial nodes in the system. An adversarial node need not correctly follow the prescribed algorithm and can send arbitrary messages to the server during the learning procedure. The identity of adversarial nodes, although a priori unknown to the server and the remaining honest (i.e., non-adversarial) nodes, remains fixed throughout the execution of the algorithm. In classic distributed computing, adversarial nodes are commonly referred as *Byzantine*. In the specific case of Robust DMGD (i.e., Algorithm 5), if a node $i \in [n]$ is honest, it sends gradients to the server that are computed as per the procedure defined in (4.2). However, an adversarial node can choose to send arbitrary vectors for its gradients to the server. Typically, the fraction of adversarial nodes, i.e., $\frac{f}{n}$, is assumed to be less than $\frac{1}{2}$. Indeed, we cannot achieve any meaningful robustness when a majority of the nodes are adversarial.

It is important to notice that we do not know a priori the number of adversarial nodes present in the system. This means that f is usually difficult to estimate in practice. When designing a robust machine learning algorithm, f is simply a *robustness parameter* that specifies the maximum number of adversarial nodes that the algorithm can tolerate. The algorithm delivers a model of specified training error when executed with less than (or equal to) f adversarial nodes in a system. However, if there are in fact more than f adversarial nodes, then the algorithm cannot guarantee an accurate model. This point becomes more clear once we introduce below, in Sect. 4.2.1, the definition of *resilience*, which formalizes the notion of robustness in the context of federated machine learning.

Remark 4.3 The results, concerning the robustness of Robust DMGD, presented in this chapter hold true even when the identity of adversarial nodes is *dynamic*, i.e., can change from one iteration to another, as long as the total number of adversarial nodes is bounded by f (the chosen robustness parameter) in all iterations. However, when the identity of the adversarial nodes is fixed, i.e., an honest node cannot become adversarial during the learning procedure, we can significantly improve the robustness guarantee (compared to the Robust DMGD method) using advanced

schemes such as *variance reduction* or *distributed gradient momentum*. The latter scheme is introduced and analyzed in Chap. 6.

4.2.1 Definition of Resilience

In general, regardless of the distributed machine learning algorithm we use, as the adversarial nodes need not share credible information about their local loss functions, seeking a solution to the original optimization problem (3.2), i.e., minimizing the global average loss $\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_i(\boldsymbol{\theta})$, is generally vacuous. A more reasonable goal, in the presence of adversarial nodes, is to have the server compute a model that minimizes the average loss function for the honest nodes. Specifically, we characterize the robustness of a distributed machine learning algorithm through the notion of (f, ε)-*resilience*, where recall that f represents the maximum number of adversarial nodes that the system intends to tolerate and ε is the training error over the training samples held by the honest nodes. For stating (f, ε)-resilience, we introduce the following notation. For a non-empty set of nodes S, we denote its average loss by

$$\mathcal{L}_S(\boldsymbol{\theta}) \coloneqq \frac{1}{|S|}\sum_{i\in S}\mathcal{L}_i(\boldsymbol{\theta}).$$

Then, $\mathcal{L}_S^*$ denotes the minimum value of $\mathcal{L}_S(\boldsymbol{\theta})$, i.e., $\mathcal{L}_S^* = \min_{\boldsymbol{\theta}\in\mathbb{R}^d}\mathcal{L}_S(\boldsymbol{\theta})$.

Before we formally express the property of resilience, we remark that the robustness parameter f determines the *maximum number* of adversarial nodes the algorithm can tolerate. It is however possible that in an execution of the algorithm the *actual number* of adversarial nodes is less than f. As we cannot know a prior this actual number, the robustness guarantee only depends on (the chosen limit) f and not on the actual number of adversarial nodes. Otherwise, the robustness guarantee is rendered unverifiable.

Definition 4.1 ((f, ε)-Resilience) A distributed algorithm Π is said to be (f, ε)-*resilient* if, despite the presence of up to f adversarial nodes (out of n), Π enables the server to output a model $\hat{\boldsymbol{\theta}}$ such that, for every $H \subseteq [n]$ comprising $n - f$ honest nodes (that correctly follow Π), the following holds true:

$$\mathcal{L}_H\left(\hat{\boldsymbol{\theta}}\right) - \mathcal{L}_H^* \leq \varepsilon. \tag{4.9}$$

The property of (f, ε)-resilience ensures that the server delivers a model that is ε-*suboptimal* with respect to the training samples held by the honest nodes, even when the system is infiltrated by up to f adversarial nodes that might behave arbitrarily. Recall from our discussion earlier, the original DMGD algorithm (and other gradient-descent methods) presented in Chap. 3 can be (f, ε)-resilient if and only if $f = 0$, i.e., it cannot even tolerate a single adversarial node in the system.

Although the objective of (f, ε)-resilience can be achieved when the loss function is convex, the optimization guarantee (4.9) is difficult to obtain when the loss function $\mathcal{L}_H$ is non-convex (even if all the nodes are honest, as discussed in Chap. 3). In the context of gradient-descent methods, considering the non-convex case, we aim for a stationary point, i.e., we relax resilience to (f, ε)-*stationary resilience*.

Definition 4.2 ((f, ε)-Stationary Resilience) A distributed algorithm Π, in a server-based architecture, is said to be (f, ε)-*stationary resilient* if, despite the presence of up to f adversarial nodes, Π enables the server to output $\hat{\boldsymbol{\theta}}$ such that, for every set H comprising $n - f$ nodes that correctly follow Π, the following holds true:

$$\left\| \nabla \mathcal{L}_H \left(\hat{\boldsymbol{\theta}} \right) \right\|^2 \leq \varepsilon \, . \tag{4.10}$$

Similar to the non-adversarial setting, when studying randomized algorithms, such as DMGD, we relax the deterministic optimization guarantees (4.9) and (4.10) by their stochastic variants. Typically, we apply expectation over the randomness introduced in the algorithm, e.g., sampling of data points in DMGD by the nodes for computing local stochastic gradients.

Remark 4.4 It is natural to wonder whether we can obtain (f, ε)-resilience (or stationary resilience) for any pair (f, ε). The answer is no. In general, we cannot tolerate a majority of adversarial nodes, i.e., if $f \geq \frac{n}{2}$, then we cannot ensure $\varepsilon < \infty$. Furthermore, even when $f < \frac{n}{2}$, the training error ε (over honest nodes' data) can be bounded away from zero. In other words, ε cannot be arbitrarily small, and the lower bound is determined by the *heterogeneity* of the data held by the honest nodes. We only consider a particular class of heterogeneity, specifically, the *gradient dissimilarity*, introduced shortly in Sect. 4.4. The lower bound on ε under the considered gradient dissimilarity condition is presented later in Sect. 5.3.2. We refer the interested reader to papers [20, 26, 32] for additional details on robustness limitations in the context of distributed convex optimization, which also generally apply to the federated machine learning problem we consider.

4.2.2 Breakdown Point

Recall from our discussion in Chaps. 2 and 3 that we want our learning algorithm to yield a small training error and at the same time require a small number of gradient computations. In addition to these qualities, in robust machine learning we also want the algorithm to tolerate as many number of adversarial nodes as possible. We can formalize these three elements of robust machine learning, using the notion of resilience introduced above. For simplicity, in the following we assume that the loss function $\mathcal{L}_H$ is convex. The formalism extends immediately to the non-convex case,

simply by replacing resilience with its stationary counterpart. Consider a distributed machine learning algorithm Π.

Breakdown Point. The breakdown point of Π, denoted by BP_Π, is defined to be the smallest fraction of adversarial nodes sufficient to render Π ineffective. Specifically, BP_Π is the value for which Π cannot be (q, ε)-resilient unless $q < \mathrm{BP}_\Pi\, n$ or $\varepsilon = \infty$. This implies that if the fraction of adversarial nodes are greater than or equal to BP_Π, then there exists a valid instance of the federated machine learning problem for which the adversarial nodes can force the algorithm Π to make an arbitrary large training error over honest nodes' collective training sample. Note that in some cases, as discussed shortly in Sect. 4.4, the robustness parameter f determines the breakdown point.
Training Error. Suppose that the robustness parameter of Π is set to be f. Then, the training error of Π, denoted by ε_Π, is a lower bound on the value of ε for which Π is (f, ε)-resilient. That is to say, Π is (f, ε)-resilient for all $\varepsilon \geq \varepsilon_\Pi$. Usually, we characterize training error of an algorithm for a meaningful class of federated machine learning problems, e.g., problems represented by smooth and convex point-wise loss functions. More on this point is discussed in Sect. 4.4.
Gradient Complexity. For a given robustness parameter f and training error ε, the gradient complexity of Π represents the time complexity of computing the gradients by a single honest node during the entire execution of Π. Similar to the training error, gradient complexity also generally depends on the class of learning problems being considered, e.g., the class of strongly convex loss functions with a fixed range of convexity coefficients.

Recall that there exists a trade-off between training error and gradient complexity, as discussed in Sect. 2.2.3. In short, smaller training error leads to larger gradient complexity and vice versa. In the context of robust machine learning, we have an additional trade-off between the breakdown point and training error. Basically, for many distributed algorithms we can trade the breakdown point for a smaller training error. This point becomes clearer once we present the analysis on the resilience (and stationary resilience) of the Robust DMGD method in the subsequent sections. We begin by presenting below a key property, called *robust averaging*, that quantifies the ability of a robust aggregation rule to resist corrupted gradients that could be sent by the adversarial nodes to the server.

4.3 Robust Averaging: A Primitive for Robust Aggregation

To analyze the aforementioned resilience properties of Robust DMGD (i.e., Algorithm 5), we make use of the following criterion of *robust averaging*. This criterion characterizes the ability of the robust aggregation rule, i.e., $F : \left(\mathbb{R}^d\right)^n \to \mathbb{R}^d$, to protect against erroneous gradients that could be sent to the server.

Definition 4.3 ((q, κ)-Robust Averaging) For a non-negative integer $q < n$, we say that an aggregation rule $F : \left(\mathbb{R}^d\right)^n \to \mathbb{R}^d$ is a (q, κ)-*robust averaging* if there exists a (non-negative) real value κ such that, for any set of n vectors $\mathbf{v_1}, \ldots, \mathbf{v_n} \in \mathbb{R}^d$ and any subset $S \subseteq [n]$ of size $n - q$, the following holds true:

$$\|F(\mathbf{v_1}, \ldots, \mathbf{v_n}) - \overline{\mathbf{v}}_\mathbf{S}\|^2 \leq \frac{\kappa}{|S|} \sum_{i \in S} \|\mathbf{v_i} - \overline{\mathbf{v}}_\mathbf{S}\|^2 , \tag{4.11}$$

where $\overline{\mathbf{v}}_\mathbf{S} \coloneqq \frac{1}{|S|} \sum_{i \in S} \mathbf{v_i}$. The parameter κ is referred to as the *robustness coefficient*.

The definition tells us that if an aggregation rule F is (q, κ)-robust averaging, then its output cannot be altered arbitrarily by corrupting at most q (out of n) input vectors. In the context of the Robust DMGD method, if the aggregation rule F satisfies the property of (f, κ)-robust averaging, then the aggregate of the n gradients received by the server is an approximation of the average of the gradients sent by $n - f$ honest nodes, even if up to f adversarial nodes send corrupted gradients. The approximation error is determined by (1) their *empirical covariance trace*, i.e.,

$$\frac{1}{|H|} \sum_{i \in H} \|\mathbf{v_i} - \overline{\mathbf{v}}_\mathbf{H}\|^2 ,$$

where H denotes any set comprising $n - f$ honest nodes and (2) the robustness coefficient κ. The robustness coefficient can be perceived as a quantification of the resistance provided by the aggregation rule against the adversarial nodes. The smaller the robustness coefficient, the better the accuracy for estimating the average of the correct nodes. Often, the robustness coefficient κ of an (q, κ)-robust averaging rule degrades with the expected fraction of input corruptions, i.e., $\frac{q}{n}$. In other words, as the fraction of expected corruptions increases, the accuracy of a robust averaging rule in estimating the correct average deteriorates. We present shortly, in Sect. 4.3.2, upper bounds on the robustness coefficients for the aggregation rules described in Sect. 4.1.3. But first, let us highlight an important property of robust averaging, specifically, the consistency with the majority rule, which also serves as a sanity check for the robust averaging criterion.

4.3.1 Consistency with Majority

Consider an aggregation rule $F : \left(\mathbb{R}^d\right)^n \to \mathbb{R}^d$. Suppose that F satisfies the criterion of (q, κ)-robust averaging for some κ, as long as $q < \frac{n}{2}$. In this case, F is consistent with the classic *majority rule*. More precisely, if the majority of the input vectors in the set $\mathbf{v_1}, \ldots, \mathbf{v_n}$ are identical, let us say equal to $\mathbf{v}$, then $F(\mathbf{v_1}, \ldots, \mathbf{v_n}) = \mathbf{v}$. Although consistency with the majority rule is not sufficient

to guarantee robust averaging, it is however necessary. The essential nature of this consistency is demonstrated below through means of an example.

Consider a specific federated machine learning setting, where all the nodes a priori hold the same training samples; hence, the average loss function for the honest nodes is identical to each of their local loss functions. Suppose that the loss function has a unique critical point that can be found efficiently. In this particular setting, we can easily obtain the optimal point of the honest average loss function, i.e., $\mathcal{L}_H$, as long as the adversarial nodes are in minority. The solution is as follows. At each step of the learning procedure, each honest node computes the (full-batch) gradient of its local loss function and sends it to the server. Then, because the honest nodes are in majority and all send the same gradient at each step, the server can use the gradient sent by the majority of the nodes to do the global update. This procedure essentially reduces to gradient descent, which will converge to the critical point of the loss function. Note, however, that we cannot replicate this solution if we use an aggregation rule F that does not comply with the majority rule. This is simply because, even when the current model $\boldsymbol{\theta}_\mathbf{t}$ maintained by the server in an iteration t is optimal, i.e., the gradient $\mathbf{g}_\mathbf{t}^{(\mathbf{i})}$ sent by each honest node $i \in H$ is equal to the zero vector, the aggregate $\widehat{\mathbf{g}_\mathbf{t}} = F\left(\mathbf{g}_\mathbf{t}^{(\mathbf{1})}, \ldots, \mathbf{g}_\mathbf{t}^{(\mathbf{n})}\right)$ need not equal the zero vector and the server might move away from $\boldsymbol{\theta}_\mathbf{t}$ to a suboptimal point. Such a trivial divergence from the optimal point does not occur if the aggregation respects the majority rule.

In short, implementing an aggregation rule that does not respect the majority rule enables the adversarial nodes to prevent the server from producing a critical (or an optimal) point of the honest average loss function.

4.3.2 Robustness Coefficients

The criterion of (q, κ)-robust averaging encompasses several robust aggregation rules. We summarize, in Table 4.1, the characterization of this property for the aggregation rules presented earlier in Sect. 4.1.3. Formal derivations for the values of their respective robustness coefficients, i.e., κ, are deferred to Appendix C. We would like to have the parameter $\frac{q}{n}$ as large as possible to provide resistance against arbitrary corruption of a large fraction of input vectors, thereby facilitating tolerance against a large number of adversarial nodes when implementing the Robust DMGD method. However, it is generally impossible to set $q \geq \frac{n}{2}$, i.e., an aggregation rule cannot guarantee (q, κ)-robust averaging for any value of κ when $q \geq \frac{n}{2}$. Furthermore, even when $q < \frac{n}{2}$, an aggregation rule cannot satisfy the property of (q, κ)-robust averaging with $\kappa \leq \frac{q}{n-2q}$. The interested reader can find formal proofs for both these limitations in Appendix C.1. This lower bound means that while we can design a robust averaging scheme whose error expectedly decreases with $\frac{q}{n}$, i.e., the fraction of corrupted inputs, the error cannot be made arbitrary small.

In light of the above constraints on q and κ, we remark that CWTM_q have near-optimal (q, κ)-robust averaging guarantees. Specifically, suppose that there exists a

Table 4.1 List of some robust aggregation rules that are a (q, κ)-robust averaging for the shown robustness coefficients κ, as long as $q < \frac{n}{2}$. The last column represents the order of κ when the number of adversarial nodes q is negligible compared to n

Aggregation rule	κ for any $q < \frac{n}{2}$	κ when $q \in O_n(1)$
CWMed	$4\left(1+\frac{q}{n-2q}\right)^2$	$O_n(1)$
GeoMed	$4\left(1+\frac{q}{n-2q}\right)^2$	$O_n(1)$
CWTM$_q$	$\frac{6q}{n-2q}\left(1+\frac{q}{n-2q}\right)$[a]	$O_n\left(\frac{1}{n}\right)$
Multi-Krum$_{q,1}$	$6\left(1+\frac{q}{n-2q}\right)$	$O_n(1)$

[a] CWTM rules can achieve better robustness (i.e., smaller robustness coefficients) for a weaker tolerance (i.e., smaller q), as explained in the text

positive real-valued constant c (independent of q and n) such that

$$\frac{q}{n} \leq \frac{1}{2}\left(\frac{c}{1+c}\right) < \frac{1}{2}. \tag{4.12}$$

Then, the robustness coefficient for CWTM$_q$ is given by $\kappa = \frac{3(2+c)q}{n-2q}$ (see Table 4.1).[3] These upper bounds on the robustness coefficient match the aforementioned lower bound up to some constant factors. For instance, upon substituting $c = 1$, we obtain that CWTM$_q$ is (q, κ)-robust averaging with $\kappa = \frac{6q}{n-2q}$ and $\kappa = \frac{9q}{n-2q}$, respectively, for all $q \leq \frac{n}{4}$. We remark that $\kappa \in O_n(1)$ for the median-based rules, i.e., CWMed and GeoMed, and Multi-Krum$_{q,1}$, for any parameter $q < \frac{n}{2}$. We later present a *pre-aggregation scheme*, in Chap. 5, that can confer order-optimal robustness (in terms of robust averaging) to the median-based rules, but at the cost of some additional computation.

Remark 4.5 In certain scenarios, e.g., when deriving the robustness property of an aggregation rule, it is easier to work with the following equivalent alternative of the robustness condition (4.11):

$$\left\| F\left(\mathbf{v_1}, \ldots, \mathbf{v_n}\right) - \bar{\mathbf{v}}_S \right\|^2 \leq \frac{\kappa}{2\,|S|^2} \sum_{i,j \in S} \left\| \mathbf{v_i} - \mathbf{v_j} \right\|^2.$$

The above follows from the fact that $2\,|S| \sum_{i \in S} \left\| \mathbf{v_i} - \bar{\mathbf{v}}_S \right\|^2 = \sum_{i,j \in S} \left\| \mathbf{v_i} - \mathbf{v_j} \right\|^2$.
Hint: $\left\| \mathbf{v_i} - \mathbf{v_j} \right\|^2 = \left\| \mathbf{v_i} - \bar{\mathbf{v}}_S \right\|^2 + \left\| \mathbf{v_j} - \bar{\mathbf{v}}_S \right\|^2 - 2\left\langle \mathbf{v_i} - \bar{\mathbf{v}}_S,\ \mathbf{v_j} - \bar{\mathbf{v}}_S \right\rangle$. For any $i \in S$, we have $\sum_{j \in S} \left\langle \mathbf{v_i} - \bar{\mathbf{v}}_S,\ \mathbf{v_j} - \bar{\mathbf{v}}_S \right\rangle = \left\langle \mathbf{v_i} - \bar{\mathbf{v}}_S,\ \sum_{j \in S} (\mathbf{v_j} - \bar{\mathbf{v}}_S) \right\rangle = \left\langle \mathbf{v_i} - \bar{\mathbf{v}}_S,\ \mathbf{0} \right\rangle = 0$
.

[3] **Hint:** If $\frac{q}{n} \leq \frac{1}{2}\left(\frac{c}{1+c}\right)$, then $\frac{q}{n-2q} \leq \frac{c}{2}$.

4.4 Resilience of Robust DMGD

In this section, we analyze the resilience property for the Robust DMGD algorithm when the aggregation rule corresponds to one of the schemes introduced above. We start by analyzing the generic convergence of the Robust DMGD algorithm, with the aggregation rule F satisfying the property of (f, κ)-robust averaging. We consider two cases, specifically, the average loss function for the honest nodes being either strongly convex or non-convex.

Assumptions
For the convergence analysis, we make the same assumptions on the smoothness of the point-wise loss function and the covariance trace of the local gradient estimates, i.e., Assumptions 3.1 and 3.2, respectively. Moreover, we characterize the heterogeneity among the data held by the nodes through means of a bound on their *gradient dissimilarities*. In general, it is impossible to obtain any meaningful robustness guarantee when no such bound is assumed, as shown later in Sect. 5.3.2.

Assumption 4.1 (ζ-Gradient Dissimilarity) Let H be a subset of $[n]$ comprising $n - f$ honest nodes, i.e., $|H| = n - f$. Then, there exists a real value ζ such that for all $\boldsymbol{\theta} \in \mathbb{R}^d$, the following holds true:

$$\frac{1}{n-f} \sum_{i \in H} \|\nabla \mathcal{L}_i(\boldsymbol{\theta}) - \nabla \mathcal{L}_H(\boldsymbol{\theta})\|^2 \ \leq \ \zeta \, .$$

Recall that $\mathcal{L}_H(\boldsymbol{\theta}) \coloneqq \frac{1}{n-f} \sum_{i \in H} \mathcal{L}_i(\boldsymbol{\theta})$.

In the subsequent section, we study the deviation of the Robust DMGD algorithm from the classic DMGD algorithm (presented earlier) when the aggregation rule F is assumed to be (f, κ)-robust.

Remark 4.6 The heterogeneity condition presented in Assumption 4.1 is quite restrictive, in general. Indeed, Assumption 4.1 need not be satisfied for many machine learning problems where the model parameters are cross-coupled with data points, e.g., regression models and neural networks. Nevertheless, we analyze the robustness of gradient-descent methods under this notion of gradient dissimilarity for pedagogical reasons. It enables us to obtain meaningful insights on the performance of the learning algorithm under data heterogeneity.

4.4.1 Deviation from Traditional DMGD

Before proceeding to the specific cases of strongly convex and non-convex loss functions, we make an observation regarding the deviation of the Robust DMGD algorithm from its original version, i.e., the DMGD method presented in Algorithm 4. Recall that in Robust DMGD, i.e., Algorithm 5, in each iteration t the

server updates the current model $\boldsymbol{\theta_t}$ to

$$\boldsymbol{\theta_{t+1}} = \boldsymbol{\theta_t} - \gamma_t \, \widehat{\mathbf{g}_\mathbf{t}} \,, \tag{4.13}$$

where recall from (4.3) that $\widehat{\mathbf{g}_\mathbf{t}} \coloneqq F\left(\mathbf{g_t^{(1)}}, \ldots, \mathbf{g_t^{(n)}}\right)$. Furthermore, recall that if node i is honest, then $\mathbf{g_t^{(i)}}$ is computed as per (3.3). Otherwise, $\mathbf{g_t^{(i)}}$ could be an arbitrary vector in $\mathbb{R}^d$. Let H be a subset of $[n]$ comprising $n - f$ honest nodes. We define

$$\boldsymbol{\delta_t} \coloneqq F\left(\mathbf{g_t^{(1)}}, \ldots, \mathbf{g_t^{(n)}}\right) - \mathbf{g_t^H} \,, \tag{4.14}$$

where $\mathbf{g_t^H}$ denotes the average of the local gradients for the honest nodes in H, i.e., $\mathbf{g_t^H} \coloneqq \frac{1}{|H|} \sum_{i \in H} \mathbf{g_t^{(i)}}$. Then, we can rewrite the update rule (4.13) as follows:

$$\boldsymbol{\theta_{t+1}} = \boldsymbol{\theta_t} - \gamma_t \, \mathbf{g_t^H} - \gamma_t \boldsymbol{\delta_t} \,. \tag{4.15}$$

In other words, we can treat the update rule of Robust DMGD as a perturbed variant of the original DMGD algorithm (introduced in Chap. 3) being executed by the honest nodes in H. The perturbation term $\boldsymbol{\delta_t}$ is bounded by the local gradients' covariance trace σ^2 and dissimilarity ζ when the aggregation rule F is an (f, κ)-robust averaging, as shown in Lemma 4.1. Recall, from Chap. 3, that $\mathbb{E}_t\,[\cdot]$ denotes the conditional expectation $\mathbb{E}\,[\cdot \,|\, \mathcal{P}_t]$, where

$$\mathcal{P}_t \coloneqq \left\{ (\boldsymbol{\theta_1}, \ldots, \boldsymbol{\theta_t}) \;,\; \left(\mathbf{g_1^{(i)}}, \ldots, \mathbf{g_{t-1}^{(i)}}\right)_{i \in [n]} \right\} .$$

Note that in the case of the Robust DMGD method, the expectation $\mathbb{E}\,[\cdot \,|\, \mathcal{P}_t]$ is over (1) the random sampling of data points by the honest nodes (for computing their local gradient estimates) at iteration t and (2) the distribution used by the adversarial nodes for sending vectors to the server (disguised as their local gradient estimates) at iteration t. While the former source of randomness was also present in the classic DMGD algorithm, the latter is specific to its robust counterpart.

Lemma 4.1 *Suppose Assumptions 3.2 and 4.1. Consider the Robust DMGD algorithm with batch-size $b \in [1, m)$, where the aggregation F is an (f, κ)-robust averaging. Then, for each iteration t, we have*

$$\mathbb{E}_t \left[\|\boldsymbol{\delta_t}\|^2 \right] \leq 4\kappa \, \frac{\sigma^2}{b} + \kappa \, \zeta \,.$$

Proof Consider an arbitrary iteration t of the Robust DMGD algorithm. Recall that H is a subset of $[n]$ comprising $n - f$ honest nodes. From (4.14), we have

$$\|\boldsymbol{\delta}_{\mathbf{t}}\|^2 = \left\| F\left(\mathbf{g}_{\mathbf{t}}^{(\mathbf{1})}, \ldots, \mathbf{g}_{\mathbf{t}}^{(\mathbf{n})}\right) - \mathbf{g}_{\mathbf{t}}^{\mathbf{H}} \right\|^2 .$$

As $|H| = n - f$, by the definition of (f, κ)-robust averaging, i.e., Definition 4.3, we have

$$\|\boldsymbol{\delta}_{\mathbf{t}}\|^2 \leq \frac{\kappa}{n-f} \sum_{i \in H} \left\| \mathbf{g}_{\mathbf{t}}^{(\mathbf{i})} - \mathbf{g}_{\mathbf{t}}^{\mathbf{H}} \right\|^2 . \tag{4.16}$$

For each $i \in H$, we have that

$$\begin{aligned} \left\| \mathbf{g}_{\mathbf{t}}^{(\mathbf{i})} - \mathbf{g}_{\mathbf{t}}^{\mathbf{H}} \right\|^2 = \Big\| \mathbf{g}_{\mathbf{t}}^{(\mathbf{i})} - \nabla \mathcal{L}_i(\boldsymbol{\theta}_{\mathbf{t}}) + \nabla \mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t}}) - \mathbf{g}_{\mathbf{t}}^{\mathbf{H}} \\ + \nabla \mathcal{L}_i(\boldsymbol{\theta}_{\mathbf{t}}) - \nabla \mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t}}) \Big\|^2 . \end{aligned}$$

As $\|\mathbf{v}\|^2 = \langle \mathbf{v}, \mathbf{v} \rangle$, we obtain that $\left\| \mathbf{g}_{\mathbf{t}}^{(\mathbf{i})} - \mathbf{g}_{\mathbf{t}}^{\mathbf{H}} \right\|^2 = A_t + B_t$, where

$$\begin{aligned} A_t \coloneqq \left\| \mathbf{g}_{\mathbf{t}}^{(\mathbf{i})} - \nabla \mathcal{L}_i(\boldsymbol{\theta}_{\mathbf{t}}) \right\|^2 + \left\| \nabla \mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t}}) - \mathbf{g}_{\mathbf{t}}^{\mathbf{H}} \right\|^2 \\ + \|\nabla \mathcal{L}_i(\boldsymbol{\theta}_{\mathbf{t}}) - \nabla \mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t}})\|^2 , \end{aligned} \tag{4.17}$$

and

$$\begin{aligned} B_t \coloneqq & 2\left\langle \mathbf{g}_{\mathbf{t}}^{(\mathbf{i})} - \nabla \mathcal{L}_i(\boldsymbol{\theta}_{\mathbf{t}}),\ \nabla \mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t}}) - \mathbf{g}_{\mathbf{t}}^{\mathbf{H}} \right\rangle \\ & + 2\left\langle \nabla \mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t}}) - \mathbf{g}_{\mathbf{t}}^{\mathbf{H}},\ \nabla \mathcal{L}_i(\boldsymbol{\theta}_{\mathbf{t}}) - \nabla \mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t}}) \right\rangle \\ & + 2\left\langle \mathbf{g}_{\mathbf{t}}^{(\mathbf{i})} - \nabla \mathcal{L}_i(\boldsymbol{\theta}_{\mathbf{t}}),\ \nabla \mathcal{L}_i(\boldsymbol{\theta}_{\mathbf{t}}) - \nabla \mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t}}) \right\rangle . \end{aligned}$$

Applying the conditional expectation $\mathbb{E}_t[\cdot]$, we obtain that

$$\mathbb{E}_t\left[\left\| \mathbf{g}_{\mathbf{t}}^{(\mathbf{i})} - \mathbf{g}_{\mathbf{t}}^{\mathbf{H}} \right\|^2 \right] = \mathbb{E}_t[A_t] + \mathbb{E}_t[B_t] . \tag{4.18}$$

We obtain next an upper bound on $\mathbb{E}_t[B_t]$ in terms of $\mathbb{E}_t[A_t]$.

(continued)

Recall that, for any honest node $i \in H$, $\mathbb{E}_t\left[\mathbf{g_t^{(i)}}\right] = \nabla \mathcal{L}_i(\boldsymbol{\theta_t})$. Therefore, as $\boldsymbol{\theta_t}$ is part of the history $\mathcal{P}_t$,

$$\begin{aligned}&\mathbb{E}_t\left[\left\langle \mathbf{g_t^{(i)}} - \nabla \mathcal{L}_i(\boldsymbol{\theta_t}),\ \nabla \mathcal{L}_i(\boldsymbol{\theta_t}) - \nabla \mathcal{L}_H(\boldsymbol{\theta_t})\right\rangle\right]\\ &= \left\langle \mathbb{E}_t\left[\mathbf{g_t^{(i)}}\right] - \nabla \mathcal{L}_i(\boldsymbol{\theta_t}),\ \nabla \mathcal{L}_i(\boldsymbol{\theta_t}) - \nabla \mathcal{L}_H(\boldsymbol{\theta_t})\right\rangle = 0\,.\end{aligned}$$

Similarly, as $\mathbb{E}_t\left[\mathbf{g_t^H}\right] = \nabla \mathcal{L}_H(\boldsymbol{\theta_t})$, we have

$$\mathbb{E}_t\left[\left\langle \nabla \mathcal{L}_H(\boldsymbol{\theta_t}) - \mathbf{g_t^H},\ \nabla \mathcal{L}_i(\boldsymbol{\theta_t}) - \nabla \mathcal{L}_H(\boldsymbol{\theta_t})\right\rangle\right] = 0\,.$$

Consequently,

$$\mathbb{E}_t\left[B_t\right] = 2\mathbb{E}_t\left[\left\langle \mathbf{g_t^{(i)}} - \nabla \mathcal{L}_i(\boldsymbol{\theta_t}),\ \nabla \mathcal{L}_H(\boldsymbol{\theta_t}) - \mathbf{g_t^H}\right\rangle\right]. \tag{4.19}$$

By Cauchy–Schwarz inequality and the fact that $2ab \leq a^2 + b^2$, we obtain that

$$\begin{aligned}&2\left\langle \mathbf{g_t^{(i)}} - \nabla \mathcal{L}_i(\boldsymbol{\theta_t}),\ \nabla \mathcal{L}_H(\boldsymbol{\theta_t}) - \mathbf{g_t^H}\right\rangle \leq 2\left\|\mathbf{g_t^{(i)}} - \nabla \mathcal{L}_i(\boldsymbol{\theta_t})\right\|\left\|\nabla \mathcal{L}_H(\boldsymbol{\theta_t}) - \mathbf{g_t^H}\right\|\\ &\leq \left\|\mathbf{g_t^{(i)}} - \nabla \mathcal{L}_i(\boldsymbol{\theta_t})\right\|^2 + \left\|\nabla \mathcal{L}_H(\boldsymbol{\theta_t}) - \mathbf{g_t^H}\right\|^2.\end{aligned}$$

Substituting from the above in (4.19), we obtain that

$$\mathbb{E}_t\left[B_t\right] \leq \mathbb{E}_t\left[\left\|\mathbf{g_t^{(i)}} - \nabla \mathcal{L}_i(\boldsymbol{\theta_t})\right\|^2 + \left\|\nabla \mathcal{L}_H(\boldsymbol{\theta_t}) - \mathbf{g_t^H}\right\|^2\right].$$

As $\left\|\mathbf{g_t^{(i)}} - \nabla \mathcal{L}_i(\boldsymbol{\theta_t})\right\|^2 + \left\|\nabla \mathcal{L}_H(\boldsymbol{\theta_t}) - \mathbf{g_t^H}\right\|^2 = A_t - \|\nabla \mathcal{L}_i(\boldsymbol{\theta_t}) - \nabla \mathcal{L}_H(\boldsymbol{\theta_t})\|^2$, the above implies that

$$\mathbb{E}_t\left[B_t\right] \leq \mathbb{E}_t\left[A_t\right] - \mathbb{E}_t\left[\|\nabla \mathcal{L}_i(\boldsymbol{\theta_t}) - \nabla \mathcal{L}_H(\boldsymbol{\theta_t})\|^2\right].$$

Substituting from the above in (4.18) yields

$$\mathbb{E}_t\left[\left\|\mathbf{g_t^{(i)}} - \mathbf{g_t^H}\right\|^2\right] \leq 2\mathbb{E}_t\left[A_t\right] - \mathbb{E}_t\left[\|\nabla \mathcal{L}_i(\boldsymbol{\theta_t}) - \nabla \mathcal{L}_H(\boldsymbol{\theta_t})\|^2\right].$$

(continued)

Upon substituting A_t (from (4.17)) in the above, we obtain that

$$\mathbb{E}_t\left[\left\|\mathbf{g_t^{(i)}}-\mathbf{g_t^H}\right\|^2\right] \leq 2\mathbb{E}_t\left[\left\|\mathbf{g_t^{(i)}}-\nabla\mathcal{L}_i(\boldsymbol{\theta_t})\right\|^2\right]+2\mathbb{E}_t\left[\left\|\nabla\mathcal{L}_H-\mathbf{g_t^H}\right\|^2\right] + \mathbb{E}_t\left[\|\nabla\mathcal{L}_i(\boldsymbol{\theta_t})-\nabla\mathcal{L}_H(\boldsymbol{\theta_t})\|^2\right].$$

Recall, from the proof of Theorem 3.1, that under Assumption 3.2, we have $\mathbb{E}_t\left[\left\|\mathbf{g_t^{(i)}}-\nabla\mathcal{L}_i(\boldsymbol{\theta_t})\right\|^2\right] \leq \frac{\sigma^2}{b}$. Consequently, as the honest nodes compute independently their local gradients, and $|H| = n - f$, we have $\mathbb{E}_t\left[\left\|\nabla\mathcal{L}_H-\mathbf{g_t^H}\right\|^2\right] \leq \frac{\sigma^2}{(n-f)b} \leq \frac{\sigma^2}{b}$. Using these observations in the above, we obtain that

$$\mathbb{E}_t\left[\left\|\mathbf{g_t^{(i)}}-\mathbf{g_t^H}\right\|^2\right] \leq \frac{4\sigma^2}{b}+\mathbb{E}_t\left[\|\nabla\mathcal{L}_i(\boldsymbol{\theta_t})-\nabla\mathcal{L}_H(\boldsymbol{\theta_t})\|^2\right].$$

The above holds true for each node $i \in H$. Therefore,

$$\frac{\sum_{i\in H}\mathbb{E}_t\left[\left\|\mathbf{g_t^{(i)}}-\mathbf{g_t^H}\right\|^2\right]}{n-f} \leq \frac{4\sigma^2}{b}+\frac{\sum_{i\in H}\mathbb{E}_t\left[\|\nabla\mathcal{L}_i(\boldsymbol{\theta_t})-\nabla\mathcal{L}_H(\boldsymbol{\theta_t})\|^2\right]}{n-f}.$$

Due to Assumption 4.1, $\frac{1}{n-f}\sum_{i\in H}\mathbb{E}_t\left[\|\nabla\mathcal{L}_i(\boldsymbol{\theta_t})-\nabla\mathcal{L}_H(\boldsymbol{\theta_t})\|^2\right] \leq \zeta$. Thus, from above we obtain that

$$\frac{1}{n-f}\sum_{i\in H}\mathbb{E}_t\left[\left\|\mathbf{g_t^{(i)}}-\mathbf{g_t^H}\right\|^2\right] \leq \frac{4\sigma^2}{b}+\zeta.$$

Applying the conditional expectation $\mathbb{E}_t[\cdot]$ on both sides in (4.16) and then substituting from the above conclude the proof. □

The above upper bound for $\boldsymbol{\delta_t}$ plays a crucial role in the following analyses on the convergence of robust DSGD. We begin by considering the case where the loss function is a non-convex function, as we did in Chap. 3. Some of the results derived for this case later prove useful, when dealing with strongly convex loss function.

4.4.2 Case of Non-convex Loss Function

In this section, we consider the Robust DMGD algorithm with robustness parameter f, i.e., we assume that $n - f$ (out of n) nodes are honest. We first analyze, in Theorem 4.1, below the convergence of the algorithm to a stationary point of the average loss function for the honest nodes. Then, we will deduce the (f, ε)-stationary resilience property obtained at the completion of T iterations of the method, corresponding to the specific aggregation rules described in Sect. 4.1.3. Recall that, for a non-empty set $S \subseteq [n]$, $\mathcal{L}_S$ denotes the average loss function for S, i.e.,

$$\mathcal{L}_S(\boldsymbol{\theta}) \coloneqq \frac{1}{|S|} \sum_{i \in H} \mathcal{L}_i(\boldsymbol{\theta}) .$$

We let $\mathcal{L}_S^* \coloneqq \min_{\boldsymbol{\theta} \in \mathbb{R}^d} \mathcal{L}_S(\boldsymbol{\theta})$. Recall also that $\mathbb{E}[\cdot]$ denotes the total expectation over the randomness in Algorithm 5 after T iterations, i.e., $\mathbb{E}[\cdot] = \mathbb{E}_1[\cdots \mathbb{E}_T[\cdot]]$.

Theorem 4.1 *Suppose Assumptions 3.1, 3.2, and 4.1. Consider T iterations of the Robust DMGD algorithm with batch-size $b \in [1, m)$ and a constant learning rate, i.e., $\gamma_t = \gamma \leq \frac{1}{4L}$ for all $t \in [T]$. If the aggregation F is an (f, κ)-robust averaging, then, for any $H \subseteq [n]$ comprising $n - f$ honest nodes, the following holds true:*

$$\frac{1}{T} \sum_{t=1}^{T} \mathbb{E}\left[\|\nabla \mathcal{L}_H(\boldsymbol{\theta_t})\|^2\right] \leq \frac{4\left(\mathcal{L}_H(\boldsymbol{\theta_1}) - \mathcal{L}_H^*\right)}{\gamma T} + \frac{4L\sigma^2}{(n-f)b}\gamma + 3\kappa\left(\frac{4\sigma^2}{b} + \zeta\right) .$$

Proof Consider an arbitrary $t \in [T]$, and an arbitrary set $H \subseteq [n]$ comprising $n - f$ honest nodes. Similar to (3.8) in the proof of Theorem 3.1, Lipschitz smoothness of the point-wise loss function, i.e., Assumption 3.1, in conjunction with the definition of $\boldsymbol{\theta_{t+1}}$ in Robust DMGD, implies that

$$\mathbb{E}_t\left[\mathcal{L}_H(\boldsymbol{\theta_{t+1}})\right] \leq \mathcal{L}_H(\boldsymbol{\theta_t}) - \gamma_t \langle \nabla \mathcal{L}_H(\boldsymbol{\theta_t}),\, \mathbb{E}_t[\widehat{\mathbf{g}}_\mathbf{t}] \rangle + \frac{L}{2}\gamma_t^2\, \mathbb{E}_t\left[\|\widehat{\mathbf{g}}_\mathbf{t}\|^2\right] . \tag{4.20}$$

Recall, from (4.14), that

$$\widehat{\mathbf{g}}_\mathbf{t} = F\left(\mathbf{g}_\mathbf{t}^{(1)}, \ldots, \mathbf{g}_\mathbf{t}^{(n)}\right) = \mathbf{g}_\mathbf{t}^\mathbf{H} + \boldsymbol{\delta_t} . \tag{4.21}$$

(continued)

As $\mathbb{E}_t\left[\mathbf{g_t^{(i)}}\right] = \nabla\mathcal{L}_i(\boldsymbol{\theta_t})$ for each honest node $i \in H$, we have $\mathbb{E}_t\left[\mathbf{g_t^H}\right] = \nabla\mathcal{L}_H(\boldsymbol{\theta_t})$. Thus,

$$\mathbb{E}_t\left[\widehat{\mathbf{g_t}}\right] = \nabla\mathcal{L}_H(\boldsymbol{\theta_t}) + \mathbb{E}_t\left[\boldsymbol{\delta_t}\right] . \tag{4.22}$$

Next, using the triangle inequality in (4.21), and the fact that for real values a_1, a_2, we have $(a_1 + a_2)^2 \leq 2a_1^2 + 2a_2^2$,[a] we obtain that

$$\mathbb{E}_t\left[\|\widehat{\mathbf{g_t}}\|^2\right] \leq 2\mathbb{E}_t\left[\left\|\mathbf{g_t^H}\right\|^2\right] + 2\mathbb{E}_t\left[\|\boldsymbol{\delta_t}\|^2\right] . \tag{4.23}$$

Substituting from (4.22) and (4.23) in (4.20), we obtain that

$$\begin{aligned}\mathbb{E}_t\left[\mathcal{L}_H(\boldsymbol{\theta_{t+1}})\right] \leq & \mathcal{L}_H(\boldsymbol{\theta_t}) - \gamma_t \|\nabla\mathcal{L}_H(\boldsymbol{\theta_t})\|^2 + L\gamma_t^2\,\mathbb{E}_t\left[\left\|\mathbf{g_t^H}\right\|^2\right] \\ & - \gamma_t\langle\nabla\mathcal{L}_H(\boldsymbol{\theta_t}),\, \mathbb{E}_t\left[\boldsymbol{\delta_t}\right]\rangle + L\,\gamma_t^2\,\mathbb{E}_t\left[\|\boldsymbol{\delta_t}\|^2\right] . \end{aligned} \tag{4.24}$$

Upper Bound on $\mathbb{E}_t\left[\left\|\mathbf{g_t^H}\right\|^2\right]$. Note that

$$\mathbb{E}_t\left[\left\|\mathbf{g_t^H}\right\|^2\right] = \mathbb{E}_t\left[\left\|\mathbf{g_t^H} - \nabla\mathcal{L}_H(\boldsymbol{\theta_t})\right\|^2\right] + \|\nabla\mathcal{L}_H(\boldsymbol{\theta_t})\|^2 .$$

Recall from the proof of Lemma 4.1 above that under Assumption 3.2, $\mathbb{E}_t\left[\left\|\mathbf{g_t^H} - \nabla\mathcal{L}_H(\boldsymbol{\theta_t})\right\|^2\right] \leq \frac{\sigma^2}{(n-f)b}$. Therefore,

$$\mathbb{E}_t\left[\left\|\mathbf{g_t^H}\right\|^2\right] \leq \frac{\sigma^2}{(n-f)b} + \|\nabla\mathcal{L}_H(\boldsymbol{\theta_t})\|^2 .$$

Substituting from the above in (4.24), we get

$$\begin{aligned}\mathbb{E}_t\left[\mathcal{L}_H(\boldsymbol{\theta_{t+1}})\right] \leq & \mathcal{L}_H(\boldsymbol{\theta_t}) - \gamma_t(1 - L\gamma_t)\,\|\nabla\mathcal{L}_H(\boldsymbol{\theta_t})\|^2 + \gamma_t^2\,\frac{L\sigma^2}{(n-f)b} \\ & - \gamma_t\langle\nabla\mathcal{L}_H(\boldsymbol{\theta_t}),\, \mathbb{E}_t\left[\boldsymbol{\delta_t}\right]\rangle + L\,\gamma_t^2\,\mathbb{E}_t\left[\|\boldsymbol{\delta_t}\|^2\right] . \end{aligned} \tag{4.25}$$

Lower Bound on $\langle\nabla\mathcal{L}_H(\boldsymbol{\theta_t}),\, \mathbb{E}_t\left[\boldsymbol{\delta_t}\right]\rangle$. Due to the Cauchy–Schwarz inequality, and the fact that for real values a_1, a_2 we have $2a_1a_2 \leq a_1^2 + a_2^2$, we obtain

(continued)

that

$$\langle \nabla \mathcal{L}_H(\boldsymbol{\theta_t}),\, \mathbb{E}_t\left[\boldsymbol{\delta_t}\right]\rangle \geq -\left\|\nabla \mathcal{L}_H(\boldsymbol{\theta_t})\right\| \left\|\mathbb{E}_t\left[\boldsymbol{\delta_t}\right]\right\| \geq -\frac{\left\|\nabla \mathcal{L}_H(\boldsymbol{\theta_t})\right\|^2 + \left\|\mathbb{E}_t\left[\boldsymbol{\delta_t}\right]\right\|^2}{2}\,.$$

Substituting from the above in (4.25), we get

$$\mathbb{E}_t\left[\mathcal{L}_H(\boldsymbol{\theta_{t+1}})\right] \leq \mathcal{L}_H(\boldsymbol{\theta_t}) - \gamma_t\left(\frac{1}{2} - L\gamma_t\right)\left\|\nabla \mathcal{L}_H(\boldsymbol{\theta_t})\right\|^2 + \gamma_t^2\,\frac{L\sigma^2}{(n-f)b} + \frac{\gamma_t}{2}\left\|\mathbb{E}_t\left[\boldsymbol{\delta_t}\right]\right\|^2 + L\,\gamma_t^2\,\mathbb{E}_t\left[\left\|\boldsymbol{\delta_t}\right\|^2\right].$$

Note that, due to the triangle inequality, $\left\|\mathbb{E}_t\left[\boldsymbol{\delta_t}\right]\right\| \leq \mathbb{E}_t\left[\left\|\boldsymbol{\delta_t}\right\|\right]$. Combining this with the Jensen's inequality, we have $\left\|\mathbb{E}_t\left[\boldsymbol{\delta_t}\right]\right\|^2 \leq \mathbb{E}_t\left[\left\|\boldsymbol{\delta_t}\right\|^2\right]$. Using this in the above, we obtain that

$$\mathbb{E}_t\left[\mathcal{L}_H(\boldsymbol{\theta_{t+1}})\right] \leq \mathcal{L}_H(\boldsymbol{\theta_t}) - \gamma_t\left(\frac{1}{2} - L\gamma_t\right)\left\|\nabla \mathcal{L}_H(\boldsymbol{\theta_t})\right\|^2 + \gamma_t^2\,\frac{L\sigma^2}{(n-f)b} + \gamma_t\left(\frac{1}{2} + L\gamma_t\right)\mathbb{E}_t\left[\left\|\boldsymbol{\delta_t}\right\|^2\right]. \tag{4.26}$$

From Lemma 4.1, we recall that when the aggregation F is an (f,κ)-robust averaging, and Assumptions 3.2 and 4.1 hold true, $\mathbb{E}_t\left[\left\|\boldsymbol{\delta_t}\right\|^2\right] \leq 4\kappa\,\frac{\sigma^2}{b} + \kappa\,\zeta$. Using this in (4.26), we obtain that

$$\mathbb{E}_t\left[\mathcal{L}_H(\boldsymbol{\theta_{t+1}})\right] \leq \mathcal{L}_H(\boldsymbol{\theta_t}) - \gamma_t\left(\frac{1}{2} - L\gamma_t\right)\left\|\nabla \mathcal{L}_H(\boldsymbol{\theta_t})\right\|^2 + \gamma_t^2\,\frac{L\sigma^2}{(n-f)b} + \gamma_t\left(\frac{1}{2} + L\gamma_t\right)\left(4\kappa\,\frac{\sigma^2}{b} + \kappa\,\zeta\right). \tag{4.27}$$

Concluding Step. Upon taking the total expectation $\mathbb{E}\left[\cdot\right]$ on both the sides in (4.27), and rearranging the terms, we obtain that

$$\left(\frac{1}{2} - L\,\gamma_t\right)\mathbb{E}\left[\left\|\nabla \mathcal{L}_H(\boldsymbol{\theta_t})\right\|^2\right] \leq \frac{\mathbb{E}\left[\mathcal{L}_H(\boldsymbol{\theta_t})\right] - \mathbb{E}\left[\mathcal{L}_H(\boldsymbol{\theta_{t+1}})\right]}{\gamma_t} + \gamma_t\,\frac{L\sigma^2}{(n-f)b} + \left(\frac{1}{2} + L\gamma_t\right)\left(4\kappa\,\frac{\sigma^2}{b} + \kappa\,\zeta\right).$$

(continued)

As $\gamma_t = \gamma \leq \frac{1}{4L}$ for all t, we have $\frac{1}{2} - L\,\gamma_t \geq \frac{1}{4}$. Moreover, $\left(\frac{1}{2} + L\gamma_t\right) \leq \left(\frac{1}{2} + \frac{L}{4L}\right) = \frac{3}{4}$. Thus, we have

$$\frac{1}{4}\,\mathbb{E}\left[\|\nabla \mathcal{L}_H(\boldsymbol{\theta_t})\|^2\right] \leq \frac{\mathbb{E}\left[\mathcal{L}_H(\boldsymbol{\theta_t})\right] - \mathbb{E}\left[\mathcal{L}_H(\boldsymbol{\theta_{t+1}})\right]}{\gamma_t} + \gamma_t\,\frac{L\sigma^2}{(n-f)b} + \frac{3}{4}\left(4\kappa\,\frac{\sigma^2}{b} + \kappa\,\zeta\right).$$

Upon substituting $\gamma_t = \gamma$ in the above, and then taking average on both sides over $t = 1$ to $t = T$, we obtain that

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla \mathcal{L}_H(\boldsymbol{\theta_t})\|^2\right] \leq \frac{4\left(\mathbb{E}\left[\mathcal{L}_H(\boldsymbol{\theta_1}) - \mathcal{L}_H(\boldsymbol{\theta_{T+1}})\right]\right)}{\gamma T} + \frac{4L\sigma^2}{(n-f)b}\gamma + 3\left(4\kappa\,\frac{\sigma^2}{b} + \kappa\,\zeta\right).$$

As $\mathcal{L}_H(\boldsymbol{\theta_1}) - \mathcal{L}_H(\boldsymbol{\theta_{T+1}}) \leq \mathcal{L}_H(\boldsymbol{\theta_1}) - \mathcal{L}_H^*$,[b] and the initial model $\boldsymbol{\theta_1}$ is given to us, we have $\mathbb{E}\left[\mathcal{L}_H(\boldsymbol{\theta_1}) - \mathcal{L}_H(\boldsymbol{\theta_{T+1}})\right] \leq \mathbb{E}\left[\mathcal{L}_H(\boldsymbol{\theta_1}) - \mathcal{L}_H^*\right] = \mathcal{L}_H(\boldsymbol{\theta_1}) - \mathcal{L}_H^*$. This concludes the proof.

□

[a] We can reason the same using the Jensen's inequality on the square function $(\cdot)^2$.

[b] Refer to the proof of Theorem 3.1 for the reasoning.

Stationary Resilience Property

Upon selecting an appropriate value for the learning rate γ in Theorem 4.1, we obtain the following corollary, which consequently enables us to characterize the stationary resilience property of Robust DMGD.

Corollary 4.1 *Suppose Assumptions 3.1, 3.2 and 4.1. Consider T iterations of the Robust DMGD algorithm with batch-size $b \in [1, m)$ and an (f, κ)-robust averaging rule F. Let H be an arbitrary set of $n - f$ honest nodes and Δ_H be a real value such that $\mathcal{L}_H(\boldsymbol{\theta_1}) - \mathcal{L}_H^* \leq \Delta_H$. If*

$$\gamma_t = \gamma = \min\left\{\frac{1}{4L}, \sqrt{\frac{(n-f)b\Delta_H}{\sigma^2 LT}}\right\}, \forall t \in [T], \tag{4.28}$$

then, provided that $T \geq \frac{16(n-f)bL\Delta_H}{\sigma^2}$*, the following holds true:*

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}_H(\boldsymbol{\theta_t})\|^2\right] \leq 8\sqrt{\frac{L\sigma^2\Delta_H}{(n-f)bT}} + 3\kappa\left(\frac{4\sigma^2}{b}+\zeta\right).$$

Proof Note that all the conditions stated in Theorem 4.1 hold true. Therefore, from Theorem 4.1, we obtain that

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}_H(\boldsymbol{\theta_t})\|^2\right] \leq \frac{4\left(\mathcal{L}_H(\boldsymbol{\theta_1})-\mathcal{L}_H^*\right)}{\gamma T} + \frac{4L\sigma^2}{(n-f)b}\gamma + 3\kappa\left(\frac{4\sigma^2}{b}+\zeta\right).$$

As $\mathcal{L}_H(\boldsymbol{\theta_1}) - \mathcal{L}_H^* \leq \Delta_H$ in the above, we have

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}_H(\boldsymbol{\theta_t})\|^2\right] \leq \frac{4\Delta_H}{\gamma T} + \frac{4L\sigma^2}{(n-f)b}\gamma + 3\kappa\left(\frac{4\sigma^2}{b}+\zeta\right). \tag{4.29}$$

As it is assumed that $T \geq \frac{16(n-f)bL\Delta_H}{\sigma^2}$, by the definition of γ in (4.28), we have

$$\gamma = \sqrt{\frac{(n-f)b\Delta_H}{\sigma^2 LT}}.$$

Substituting from the above in (4.29) concludes the proof. □

The corollary implies that if the server samples a model randomly from the set of all the models generated during the execution of Robust DMGD, then (in expectation) the model approximates a stationary point of $\mathcal{L}_H$ with a bounded error, despite the presence of up to f adversarial nodes. Specifically, under the conditions stated in Corollary 4.1 and upon executing T iterations of the Robust DMGD algorithm, if the server outputs $\widehat{\boldsymbol{\theta}}$ chosen uniformly from the set of intermediate models $\{\boldsymbol{\theta_1}, \ldots, \boldsymbol{\theta_T}\}$, then we achieve (f, ε)-stationary resilience (as

per Definition 4.2) where

$$\varepsilon \leq 8\sqrt{\frac{\Delta L\sigma^2}{(n-f)bT}} + 3\kappa\left(\frac{4\sigma^2}{b} + \zeta\right) \in O_T\left(\sqrt{\frac{\sigma^2}{(n-f)bT}} + \kappa\frac{\sigma^2}{b} + \kappa\zeta\right). \tag{4.30}$$

The order of the training error shown in (4.30) is composed of three main terms. The first term, $O_T\left(\sqrt{\frac{\sigma^2}{(n-f)bT}}\right)$, is the vanishing error rate that corresponds to that of the original DMGD algorithm running on at least $n - f$ honest nodes (cf. Corollary 3.1). This particular term approaches zero, as we increase the total number of iterations T of the algorithm; hence, it is not a dominant factor in the final training error of the algorithm. The second and third terms, $O_T\left(\kappa\frac{\sigma^2}{b}\right)$ and $O_T(\kappa\zeta)$, are non-vanishing errors introduced (in the worst case) by the presence of adversarial nodes. Note that, contrary to the original DMGD algorithm, in the presence of adversarial nodes, the local gradient noise (measured by σ) can lead to an incompressible error, apparently implying that the learning procedure might yield bad model accuracy (or incur a high training error). As shown later in Chap. 6, this augmentation of the error due to local gradient estimation noise can be diminished by embedding *gradient momentum* in the model updates at nodes' level. However, the error due to gradient dissimilarity (measured by ζ) is something that cannot be eliminated in general. We defer the formal discussion on the limitations of robustness due to gradient dissimilarity to Sect. 5.3 in Chap. 5.

We summarize, in Table 4.2, the stationary resilience property of the Robust DMGD algorithm when the aggregation rule F is one of those described in Sect. 4.1.3. Recall that each of these aggregation rules is shown to be an (f, κ)-robust averaging in Sect. 4.3.2 when we set parameter $q = f$, where $f < \frac{n}{2}$. Thus, the breakdown point of the Robust DMGD method equipped with any of

Table 4.2 The (f, ε)-stationary resilience property of Robust DMGD with the listed aggregation rules, obtained upon completion of T iterations in the general non-convex case under the conditions stated in Corollary 4.1. For all the aggregation rules, except CWTM, we assume that $\frac{f}{n} < \frac{1}{2}$. For CWTM$_f$, as per the discussion in Sect. 4.3.2, we assume that $\frac{f}{n} \leq \frac{c}{2(1+c)}$, where c is a positive real-valued constant

Aggregation rule	Breakdown point	Training error ε
CWMed	$\frac{1}{2}$	$O_{n,T}\left(\sqrt{\frac{\sigma^2}{(n-f)bT}} + \frac{\sigma^2}{b} + \zeta\right)$
GeoMed	$\frac{1}{2}$	$O_{n,T}\left(\sqrt{\frac{\sigma^2}{(n-f)bT}} + \frac{\sigma^2}{b} + \zeta\right)$
CWTM$_f$	$\frac{f+1}{n}$	$O_{n,T}\left(\sqrt{\frac{\sigma^2}{(n-f)bT}} + \frac{f}{n}\frac{\sigma^2}{b} + \frac{f}{n}\zeta\right)$
Multi-Krum$_{f,1}$	$\frac{f+1}{n}$	$O_{n,T}\left(\sqrt{\frac{\sigma^2}{(n-f)bT}} + \frac{\sigma^2}{b} + \zeta\right)$

these aggregation rules can be tuned to be as high as $\frac{1}{2}$. The order of the training error ε, shown in Table 4.2, is obtained by recalling that $\kappa \in O_n(1)$ for CWMed, GeoMed and Multi-Krum$_{f,1}$, and $\kappa \in O_n\left(\frac{f}{n}\right)$ for CWTM$_f$. For CWTM$_f$, we set $\frac{f}{n} < \frac{1}{2}\left(\frac{c}{1+c}\right)$, where c is a positive real-valued constant (see Sect. 4.3.2). We remark that though the non-vanishing error terms grow linearly in $\frac{f}{n}$ for CWTM$_f$, the same need not be true for the other three aggregation rules. This phenomenon can be explained by the fact that CWMed, GeoMed, and Multi-Krum$_{q_1,q_2}$ are *median-based* aggregation rules and hence do not benefit from the expected strength of the honest nodes in the system. Another property of median-based rules is that their breakdown point is always $\frac{1}{2}$, regardless of the robustness parameter f (i.e., the maximum number of adversarial nodes the system is designed to tolerate), whereas CWTM involves an averaging operation that helps it improve the accuracy when the system is expected to have a larger number of honest nodes. Specifically, we can trade the breakdown point of the Robust DMGD algorithm, when using CWTM as the aggregation rule, for better training error. More on this discussed shortly below. A similar property can also be imparted to the median-based aggregation rules through the use of a pre-aggregation scheme, introduced and analyzed later in Chap. 5.

Applying the Analysis to Robust DSGD and Robust DGD

The robustness analysis presented above applies immediately to robust DSGD (i.e., Algorithm 5 with batch-size $b = 1$) by simply substituting $b = 1$ in Theorem 4.1 and Corollary 4.1. In the case of robust DGD, i.e., DGD with robust aggregation in the global phase, we do not have gradient noise. Upon substituting $\sigma = 0$, and γ to be any constant less than $\frac{1}{4L}$, in Theorem 4.1, we deduce that robust DGD is (f, ε)-stationary resilient with $\varepsilon \in O_T\left(\frac{1}{T} + \kappa\zeta\right)$, where $\kappa \in O_n(1)$ when using CWMed, GeoMed, or Krum as the aggregation rule and $\kappa \in O_n\left(\frac{f}{n}\right)$ when using CWTM$_f$ as the aggregation rule. Expectedly, in contrast to Robust DMGD, when implementing robust DGD, the gradient computation noise does not contribute to the incompressible training error. However, the cost of computing an actual gradient on the whole training sample is quite prohibitive in practice. An efficient alternative to robust DGD, for diminishing the impact of noise on training error, is to incorporate gradient momentum in Robust DMGD. This scheme is presented later in Chap. 6.

Interplay Between Breakdown Point, Training Error, and Gradient Complexity

Consider the Robust DMGD method with aggregation rule CWTM$_f$ where the robustness parameter $f < \frac{n}{2}$. Upon executing T iterations, as shown in Table 4.2, we obtain a training error of

$$\varepsilon \in O_{n,T}\left(\sqrt{\frac{\sigma^2}{(n-f)bT}} + \frac{f}{n}\frac{\sigma^2}{b} + \frac{f}{n}\zeta\right).$$

First, no matter how many iterations T we choose to implement, the training error cannot be reduced to zero, unlike the original (non-robust) DMGD method. Specifically, even when $T \to \infty$, we have a training error of $\varepsilon_\infty \in O_n\left(\frac{f}{n}\frac{\sigma^2}{b} + \frac{f}{n}\zeta\right)$. The gradient complexity, i.e., computation spent by an honest node for computing gradients during the learning procedure, for obtaining an error of $\varepsilon + \varepsilon_\infty$ is equivalent to the gradient complexity of the DMGD method with $n - f$ nodes and training error ε, i.e., $O_{\frac{1}{\varepsilon}}\left(\frac{\sigma^2}{(n-f)\varepsilon^2}\right)$. Second, by choosing to have a smaller breakdown point, i.e., tolerating smaller fraction of adversarial nodes $\frac{f}{n}$, we can improve the asymptotic training error ε_∞. Specifically, the order of the breakdown point is linear in training error. In other words, we cannot simultaneously have a higher breakdown point and a smaller training error.

Although we can obtain a much smaller training error by increasing the batch-size b, e.g., when $b = m$ (i.e., the case of robust DGD), we have an asymptotic training error $\varepsilon_\infty \in O_n\left(\frac{f}{n}\zeta\right)$, increasing batch-size beyond a certain limit might be prohibitive in practice. Interestingly, we can obtain a similar effect without increasing the gradient complexity through the use of gradient momentum in the local phase (see Chap. 6).

4.4.3 Case of Strongly Convex Loss Function

In this section, we analyze the resilience property of Robust DMGD in the case when the average loss function for honest nodes is known to be strongly convex. Recall, from Sect. 3.2.3, that every strongly convex function satisfies the PL condition, but not every function that satisfies the PL condition is strongly convex. As done in Sect. 3.2, we assume that the average loss function for the honest nodes satisfies the PL inequality. Specifically, we assume that for every set H of $n - f$ honest nodes, their average loss function $\mathcal{L}_H$ satisfies the following:

Assumption 4.2 There exists a real value μ such that for every set H comprising $n - f$ honest nodes and for all $\boldsymbol{\theta} \in \mathbb{R}^d$,

$$\|\nabla \mathcal{L}_H(\boldsymbol{\theta})\|^2 \geq 2\mu \left(\mathcal{L}_H(\boldsymbol{\theta}) - \mathcal{L}_H^*\right),$$

where $\mathcal{L}_H^* \coloneqq \min_{\boldsymbol{\theta} \in \mathbb{R}^d} \mathcal{L}_H(\boldsymbol{\theta})$.

Under the above strong convexity assumption, we are able to obtain guarantees of resilience for Robust DMGD, owing to the following convergence result. Recall that, for a non-empty set $S \subseteq [n]$, we denote by $\mathcal{L}_S$ the average loss function of the nodes in S, $\mathcal{L}_S^* \coloneqq \min_{\boldsymbol{\theta} \in \mathbb{R}^d} \mathcal{L}_S(\boldsymbol{\theta})$, and $\mathbb{E}[\cdot]$ denotes the total expectation over the randomness in Algorithm 5 over T iterations, i.e., $\mathbb{E}[\cdot] = \mathbb{E}_1[\cdots \mathbb{E}_T[\cdot]]$.

Theorem 4.2 *Suppose Assumptions 3.1, 3.2, 4.1, and 4.2. Consider an iteration t of the Robust DMGD algorithm with batch-size $b \in [1, m)$ and learning rate $\gamma_t \leq \frac{1}{4L}$.*

If the aggregation F is an (f, κ)-robust averaging, then, for $H \subseteq [n]$ comprising $n - f$ honest nodes, the following holds true:

$$\mathbb{E}\left[\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t+1}}) - \mathcal{L}_H^*\right] \leq \left(1 - \frac{\mu}{2}\gamma_t\right) \mathbb{E}\left[\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t}}) - \mathcal{L}_H^*\right] + \gamma_t^2 L \left(\frac{\sigma^2}{(n-f)b} + 4\kappa\frac{\sigma^2}{b} + \kappa\zeta\right) + \gamma_t \frac{\kappa}{2}\left(\frac{4\sigma^2}{b} + \zeta\right). \tag{4.31}$$

Proof Similarly to (4.27) in the proof of Theorem 4.1, under Assumptions 3.1, 3.2, and 4.1, we obtain that

$$\mathbb{E}_t\left[\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t+1}})\right] \leq \mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t}}) - \gamma_t\left(\frac{1}{2} - L\gamma_t\right)\|\nabla\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t}})\|^2 + \gamma_t^2\frac{L\sigma^2}{(n-f)b} + \gamma_t\left(\frac{1}{2} + L\gamma_t\right)\left(4\kappa\frac{\sigma^2}{b} + \kappa\zeta\right).$$

Taking total expectation on both sides and denoting $U_t \coloneqq \mathbb{E}\left[\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t}}) - \mathcal{L}_H^*\right]$, we obtain that

$$U_{t+1} \leq U_t - \gamma_t\left(\frac{1}{2} - L\gamma_t\right)\mathbb{E}\left[\|\nabla\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t}})\|^2\right] + \gamma_t^2\frac{L\sigma^2}{(n-f)b} + \gamma_t\left(\frac{1}{2} + L\gamma_t\right)\left(4\kappa\frac{\sigma^2}{b} + \kappa\zeta\right).$$

As $0 < \gamma_t \leq \frac{1}{4L}$, from the above we obtain that

$$U_{t+1} \leq U_t - \frac{\gamma_t}{4}\mathbb{E}\left[\|\nabla\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t}})\|^2\right] + \gamma_t^2\frac{L\sigma^2}{(n-f)b} + \gamma_t\left(\frac{1}{2} + L\gamma_t\right)\kappa\left(\frac{4\sigma^2}{b} + \zeta\right).$$

Due to Assumption 4.2, we have

$$\mathbb{E}\left[\|\nabla\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t}})\|^2\right] \geq \mathbb{E}\left[2\mu\left(\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t}}) - \mathcal{L}_H^*\right)\right] = 2\mu U_t.$$

(continued)

Hence, from above, we get

$$U_{t+1} \leq \left(1 - \frac{\mu}{2}\gamma_t\right) U_t + \gamma_t^2 \frac{L\sigma^2}{(n-f)b} + \gamma_t \left(\frac{1}{2} + L\gamma_t\right) \left(4\kappa \frac{\sigma^2}{b} + \kappa\, \zeta\right).$$

This concludes the proof. □

Theorem 4.2 implies the following: If we run T iterations of the Robust DMGD algorithm under the conditions and specifications stated in Theorem 4.2, then the growth of the sequence $\left(\mathbb{E}\left[\mathcal{L}_H(\boldsymbol{\theta}_\mathbf{t}) - \mathcal{L}_H^*\right]\right)_{t\in[T+1]}$, given by (4.31), corresponds to that of $(U_t)_{t\in[T+1]}$ in Lemma 3.1 with parameters α_1, α_2, and α_3 set as follows:

$$\alpha_1 \coloneqq \frac{\mu}{2}\,, \alpha_2 \coloneqq L\left(\frac{\sigma^2}{(n-f)b} + 4\kappa\frac{\sigma^2}{b} + \kappa\zeta\right) \text{ and } \alpha_3 \coloneqq \frac{\kappa}{2}\left(\frac{4\sigma^2}{b} + \zeta\right). \tag{4.32}$$

Hence, upon conjugating the result of Theorem 4.2 with that of Lemma 3.1, we have the following corollary. For simplicity, we state the corollary in terms of parameters α_2 and α_3 defined above.

Corollary 4.2 *Suppose Assumptions 3.1, 3.2, 4.1, and 4.2. Let $T > 2$ be an even number. Consider T iterations of the robust DSGD algorithm with batch-size $b \in [1, m)$ and the following sequence of learning rates:*

$$\gamma_t = \begin{cases} \gamma_o & , \ t < T/2 \\ \frac{4}{\mu(s_o + t - T/2)} & , \ t \geq T/2 \end{cases},$$

where γ_o and s_o are positive real values such that $\gamma_o = \frac{4}{\mu s_o} \leq \frac{1}{4L}$. If the aggregation F is an (f, κ)-robust averaging, then, for any $H \subseteq [n]$ comprising $n - f$ honest nodes, the following holds true:

$$\mathbb{E}\left[\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{T+1}}) - \mathcal{L}_H^*\right] \leq \Delta_o \frac{e^{-\frac{\mu\gamma_o}{4}(T-2)}}{T^2} + \frac{8(\alpha_2\gamma_o + \alpha_3)(s_o-1)^2}{\mu\, T^2} + \frac{32\alpha_2}{\mu^2\, T} + \frac{4\alpha_3}{\mu},$$

where Δ_o is a real value such that $(s_o - 1)^2\left(\mathcal{L}_H(\boldsymbol{\theta}_\mathbf{1}) - \mathcal{L}_H^*\right) \leq \Delta_o$, parameters α_2 and α_3 are defined in (4.32).

Proof As $\gamma_o < \frac{1}{4L}$ and $\gamma_t \leq \gamma_o$ for all t, we have $\gamma_t \leq \gamma_0 < \frac{1}{L}$ for all $t \in [T]$. Therefore, owing to Theorem 4.2, for an arbitrary $t \in [T]$, we have (recalling the definitions of α_2 and α_3 in (4.32))

$$\mathbb{E}\left[\mathcal{L}_H(\boldsymbol{\theta_{t+1}}) - \mathcal{L}_H^*\right] \leq \left(1 - \frac{\mu}{2}\gamma_t\right) \mathbb{E}\left[\mathcal{L}_H(\boldsymbol{\theta_t}) - \mathcal{L}_H^*\right] + \gamma_t^2\alpha_2 + \gamma_t\alpha_3 \,.$$

Denoting $U_t \coloneqq \mathbb{E}\left[\mathcal{L}_H(\boldsymbol{\theta_t}) - \mathcal{L}_H^*\right]$, the above can be equivalently written as

$$U_{t+1} \leq \left(1 - \frac{\mu}{2}\gamma_t\right) U_t + \gamma_t^2\alpha_2 + \gamma_t\alpha_3 \,. \tag{4.33}$$

At this point, we would like to invoke Lemma 3.1 to obtain a convergence result for the sequence $(U_t)_{t\in[T+1]}$. For doing so, we need to verify that all the conditions of the lemma hold true. We choose $\alpha_1 = \frac{\mu}{2}$ and α_2, α_3 as defined in (4.32). First, note that the condition for the sequence $(U_t)_{t\in[T+1]}$ in (4.33) reduces to that in Lemma 3.1. Second, since $\frac{\mu}{2} = \alpha_1$ and $\mu \leq L$ when Assumptions 3.1 and 3.3 hold true simultaneously (shown in Appendix A.2), we have

$$\gamma_o = \frac{4}{\mu s_o} = \frac{2}{\alpha_1 s_o} < \frac{1}{4L} \leq \frac{1}{4\mu} \leq \frac{1}{8\alpha_1} < \frac{1}{\alpha_1} \,.$$

Therefore, the sequence $(\gamma_t)_{t\in[T]}$ also satisfies the respective condition of Lemma 3.1. Hence, upon substituting the $\alpha_1 = \frac{\mu}{2}$ in Lemma 3.1, we obtain that

$$U_{T+1} \leq (s_o - 1)^2 U_1 \frac{4e^{-\mu\gamma_o(T-2)/4}}{T^2} + \frac{8(\alpha_2\gamma_o + \alpha_3)(s_o - 1)^2}{\mu\, T^2} + \frac{32\alpha_2}{\mu^2\, T} + \frac{4\alpha_3}{\mu} \,.$$

Recall that $4(s_o-1)^2U_1 = 4(s_o-1)^2\left(\mathcal{L}_H(\boldsymbol{\theta_1}) - \mathcal{L}_H^*\right) \leq \Delta_o$. This concludes the proof. □

We can use the above corollary to deduce the resilience of Robust DMGD, in the strongly convex case, when the aggregation rule is one of those described in Sect. 4.1.3.

Resilience Property

In the case of strong convexity, recall that we output the latest model generated by algorithm. Specifically, the server outputs $\widehat{\boldsymbol{\theta}} \coloneqq \boldsymbol{\theta_{T+1}}$ after T iterations of the Robust DMGD method. Hence, under the conditions specified in Corollary 4.2, we achieve

(f, ε)-resilience (as per Definition 4.1) with

$$\varepsilon \leq \Delta_o \frac{e^{-\frac{\mu\gamma_o}{4}(T-2)}}{T^2} + \frac{8(\alpha_2\gamma_o + \alpha_3)(s_o-1)^2}{\mu\, T^2} + \frac{32\alpha_2}{\mu^2\, T} + \frac{4\alpha_3}{\mu}\,, \tag{4.34}$$

where recall from (4.32) that

$$\alpha_2 \coloneqq L\left(\frac{\sigma^2}{(n-f)b} + 4\kappa\frac{\sigma^2}{b} + \kappa\zeta\right) \text{ and } \alpha_3 = \frac{\kappa}{2}\left(\frac{4\sigma^2}{b} + \zeta\right).$$

Choosing $s_o = 16\frac{L}{\mu} \geq 16$ in Corollary 4.2,[4] we have $\gamma_o = \frac{1}{4L}$ (while satisfying the condition stated in the corollary). Using this value of γ_o in (4.34), and ignoring the constants (including Δ_o and s_o), we get

$$\varepsilon \in O_T\left(\frac{e^{-\frac{1}{K_{\mathcal{L}_H}}T}}{T^2} + \left(\left(\frac{1}{n-f} + \kappa\right)\frac{\sigma^2}{b} + \kappa\zeta\right)\frac{1}{\mu T}\left(\frac{1}{T} + K_{\mathcal{L}_H}\right) + \kappa\left(\frac{\sigma^2}{b} + \zeta\right)\right). \tag{4.35}$$

Note that $K_{\mathcal{L}_H} \coloneqq \frac{L}{\mu}$ denotes the condition number of the average loss function $\mathcal{L}_H$. In the general setting when there is non-zero gradient noise, i.e., $\sigma > 0$, by ignoring the higher-order (exponential and quadratic) terms in T in (4.35), we get[5]

$$\varepsilon \in O_T\left(K_{\mathcal{L}_H}\,\frac{\sigma^2}{(n-f)b}\,\frac{1}{T} + \kappa\left(\frac{\sigma^2}{b} + \zeta\right)\right). \tag{4.36}$$

We summarize, in Table 4.3, the training error shown in (4.36) corresponding to the aggregation rules described in Sect. 4.1.3, upon substituting their respective robustness coefficients from Sect. 4.3.2. Similar to the non-convex case, the first term, i.e., $O_T\left(K_{\mathcal{L}_H}\,\frac{\sigma^2}{(n-f)b}\,\frac{1}{T}\right)$, corresponds to that of the original DMGD algorithm running exclusively by $n-f$ honest nodes (see Sect. 3.2.3). The dominant error due to the adversarial nodes is represented by $O_T\left(\kappa\frac{\sigma^2}{b} + \kappa\zeta\right)$. As we already mentioned, we can eliminate the error due to the gradient noise, i.e., σ^2, through the use of local gradient momentum. But, the same might not be true in general for gradient dissimilarity, i.e., ζ. We present a formal discussion on the limitations of robustness due to gradient dissimilarity later in Sect. 5.3.2 of Chap. 5.

[4] Recall that $L \geq \mu$ when Assumptions 3.1 and 4.2 hold true at the same time (see Appendix A.2).

[5] We ignore μ, as it can be a priori fixed by simply normalizing (or scaling) the point-wise loss function.

Table 4.3 The (f, ε)-resilience property of Robust DMGD with the listed aggregation rules, obtained upon completion of T iterations in the strongly convex case under the conditions stated in Corollary 4.2. For all the aggregation rules, except CWTM, we assume that $\frac{f}{n} < \frac{1}{2}$. For CWTM$_f$, as per the discussion in Sect. 4.3.2, we assume that $\frac{f}{n} \leq \frac{c}{2(1+c)}$, where c is a positive real-valued constant

Aggregation rule	Breakdown point	Training error ε
CWMed	$\frac{1}{2}$	$O_{n,T}\left(\frac{K_{\mathcal{L}_H}\sigma^2}{(n-f)b}\,\frac{1}{T} + \frac{\sigma^2}{b} + \zeta\right)$
GeoMed	$\frac{1}{2}$	$O_{n,T}\left(\frac{K_{\mathcal{L}_H}\sigma^2}{(n-f)b}\,\frac{1}{T} + \frac{\sigma^2}{b} + \zeta\right)$
CWTM$_f$	$\frac{f+1}{n}$	$O_{n,T}\left(\frac{K_{\mathcal{L}_H}\sigma^2}{(n-f)b}\,\frac{1}{T} + \frac{f}{n}\frac{\sigma^2}{b} + \frac{f}{n}\zeta\right)$
Multi-Krum$_{f,1}$	$\frac{f+1}{n}$	$O_{n,T}\left(\frac{K_{\mathcal{L}_H}\sigma^2}{(n-f)b}\,\frac{1}{T} + \frac{\sigma^2}{b} + \zeta\right)$

Resilience of Robust DSGD and Robust DGD

We immediately obtain the resilience property for the robust DSGD method from (4.36) upon substituting the batch-size $b = 1$. In the special case of robust DGD, i.e., when each honest node computes local gradients for the whole training sample ($b = m$), the resilience property is given by (4.35) once we substitute $\sigma = 0$. Specifically, in the case of DGD, we have $\varepsilon \in O_T\left(e^{-\frac{1}{K_{\mathcal{L}_H}}T} + \kappa\zeta\right)$. We remark that, when $f = 0$, the convergence guarantee reduces to that of the traditional (non-robust) DGD method, provided the aggregation rule is CWTM$_f$ (simply because $\kappa \in O_n\left(\frac{f}{n}\right)$ for these rules). However, the same cannot be said for the other (median-based) aggregation rules, i.e., CWMed, GeoMed, and Krum (i.e., Multi-Krum$_{f,1}$). This shortcoming of median-based aggregates can be overcome through the use of pre-aggregation schemes, introduced shortly in Chap. 5.

4.5 Chapter Notes

We have presented in this chapter a primer on the concept of robustness in distributed machine learning (including federated learning). We have first highlighted the fragility of standard distributed gradient-descent methods and have provided an intuition to why averaging is not a robust approach to aggregating the incoming gradients at the server. Robustness calls for surrogate (robust) aggregation methods at the server, for which we have provided a set of examples. To formalize and measure the quality of robustness enjoyed by the variant of DMGD resulting from the usage of a surrogate aggregation method (called Robust DMGD), we have introduced the concepts of (f, κ)-robust averaging, and of (f, ε)-resilience. Lastly, we have presented an in-depth analysis for the (f, ε)-resilience property of

Robust DMGD in both cases: strongly convex and non-convex. Specifically, we have derived training error (residual error ε after T training steps), and its dependence on gradient complexity and breakdown point (i.e., tolerable fraction of adversarial nodes). The proofs that we have presented in this chapter are based upon our recent work on robust machine learning that appears in [3–5, 16]. Furthermore, the convergence analysis we have presented under strong convexity, using a scheduled learning rate, is inspired by the results presented in [24, 31].

Brief History of Robust Aggregation

The main building block of Robust DMGD is the *robust aggregation* of nodes' gradients at the server. Although the idea of robust aggregation has been widely adopted in *robust statistics* at least since the 1980s, e.g., see [21, 29], as far as we know, robust aggregation (specifically, trimmed mean) was introduced to robust distributed gradient descent as recently as 2016 in [32]. However, [32] only presented the scalar case, i.e., it was assumed that $d = 1$. Later, in 2017, geometric median was analyzed as an alternative to trimmed mean for robustness in *high-dimensional* distributed gradient descent [10]. In the same year, Krum was proposed to impart robustness to distributed *stochastic* gradient descent in [7], but the algorithm was only analyzed in a homogeneous setting. In the following years, various robust aggregations were proposed and studied, e.g., see [20, 22, 25, 37, 39], for more general distributed machine learning settings. However, the benefits of these robust aggregation schemes were analyzed under varying sets of assumptions. For this reason, it was impossible to rigorously compare their effectiveness. To remedy this, a unifying framework, based upon the definition of (f, λ)-*resilient averaging*, was presented in 2022 [16]. This enabled us to quantify the robustness provided by different aggregation schemes in a standard machine learning setting. While a similar notion to (f, λ)-resilient averaging, specifically, (c, δ)-*agnostic robustness*, was proposed earlier in 2021 [22], the latter fails to encapsulate many aggregation schemes. In 2023 [3], the definition of (f, λ)-resilient averaging was later refined to (f, κ)-*robust averaging*, which is what we have presented in this chapter. The main advantage of the latter, over the former definition, is that it facilitates a *tighter* resilience analysis (under the same set of assumptions). Although we have only presented a few aggregation rules that can be characterized by the definition of (f, κ)-robust averaging, the definition is generic enough to incorporate a wide range of robust aggregation schemes.

Lastly, it is important to notice that (f, κ)-robust averaging need not yield a tight robustness analysis in a setting that is different than the one we have considered. Specifically, in the case when the covariance of local gradients' noise is assumed to have *bounded spectral norm* instead of *bounded trace* (i.e., Assumption 3.2), (f, κ)-robust averaging yields suboptimal robustness. In that particular case, we should use a *spectral* robust averaging primitive, such as the ones defined in [4, 40].

Few Words on Robust Estimation

The problem of robust estimation is a special case of robust machine learning where the identity of adversarial nodes may change from one training round to another, and the training samples held by the honest nodes are assumed to be

drawn independently from an identical distribution. This justifies the use of standard robust statistical methods (such as CWTM, CWMed, or GeoMed) at the server level. However, proving the convergence of the resulting algorithm in the context of machine learning is not a straightforward adaptation of the results known in robust statistics and involves additional intricacies. To know more on the efficacy of other robust-statistics techniques, such as *distributional* and *spectral filtering*, to robust machine learning, we refer the interested reader to papers [1, 2, 9, 13, 27, 28]. Also, note that while we have studied the optimality of CMTM from the viewpoint of (f, κ)-robust averaging, it has been shown optimal even in the context of robust statistics [39].

Fault-Tolerant Distributed Optimization

The notion of (f, ε)-resilience is pivotal to the analysis we have provided in this chapter. This notion, largely inspired by the work of [26] in the literature of *fault-tolerant distributed optimization*, enables us to get a clear view of both the training error and the breakdown point of Robust DMGD. It is actually a formalization of the problem of |em Byzantine resilience introduced in the context of federated machine learning in [7, 10]. Since these pioneering works, the field of robustness in machine learning has evolved in many directions, e.g., see the survey paper [19]. We note that problem of robust machine learning, which is the key topic of this chapter, is also closely related to the problem of fault-tolerant distributed optimization. However, the latter mostly focuses on convex loss functions and mostly assumes local gradients to be non-stochastic. Nevertheless, many existing results (especially the lower bounds) in the fault-tolerant distributed optimization literature can also be applied to robust machine learning. We refer the interested reader for more details on the former to papers [20, 26, 32–34].

Simulations of Adversarial Behavior

We have presented in this chapter a theoretical framework for measuring robustness. The metrics introduced can be evaluated in practice, for a given learning scheme, through means of various simulations defining plausible actions for the adversarial nodes. These adversarial simulations are also commonly referred as *attacks*. The objective of these attacks is basically to test the robustness of a learning algorithm at hand. Some of the prominent attacks in the literature include *a little is enough* [6], *fall of empire* [36], *sign flipping* [2], *label flipping* [2], *MinSum* (and *MinMax*) [30], and *mimic* [23]. While label flipping and mimic are *data poisoning* attacks, i.e., an adversarial node simply corrupts its local training samples before responding to the server, the rest are categorized as *model poisoning* attacks wherein the adversarial nodes collude to fabricate malicious gradients during the learning procedure. It is often believed that the latter is stronger than the former; however, this belief remains to be proven. We refer the interested reader to papers [17, 30] to learn more about the differences and comparisons between these two types of attack. In this chapter, we do not present details on these specific malicious behaviors, rather we focus on robustness schemes that aim to protect a machine learning system from adversarial nodes that could behave arbitrarily bad.

Further Extensions of Robust DMGD

Like in Chap. 3, we have focused on server-based federated algorithms wherein the nodes have access to a *trusted* server. In practice, however, the server (even if it exists) might not always be trustworthy, in which case the nodes must implement (some elements of) decentralized machine learning to ensure robustness. Nevertheless, studying robustness in the server-based framework constitutes a pivotal step toward addressing the problem in the more complex peer-to-peer framework wherein the nodes do not have at disposal a trusted server. A few works, including [14, 15, 18, 35, 37, 38, 41], consider decentralized variants of Robust DMGD in peer-to-peer architecture. Although the underlying principle for robustness in decentralized DMGD is still robust aggregation, the analysis is more intricate due to the presence of *drift* between the nodes' local models, which is difficult to control when the underlying communication network is non-complete.

Lastly, we refer the interested reader to paper [12] for an analysis on robustness in the *FederatedAveraging* framework where in each iteration the server samples only some of the nodes to participate in the update procedure, and each participating node performs multiple local update steps before sending their individual updates to the server. The server then generates a global model update by taking a robust aggregation of the local updates received from the nodes.

References

1. Alistarh D, Allen-Zhu Z, Li J (2018) Byzantine stochastic gradient descent. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems, pp 4618–4628
2. Allen-Zhu Z, Ebrahimianghazani F, Li J, Alistarh D (2021) Byzantine-Resilient non-convex stochastic gradient descent. In: International Conference on Learning Representations
3. Allouah Y, Farhadkhani S, Guerraoui R, Gupta N, Pinot R, Stephan J (2023) Fixing by mixing: a recipe for optimal Byzantine ML under heterogeneity. In: Proceedings of The 26th International Conference on Artificial Intelligence and Statistics, vol 206. Proceedings of Machine Learning Research. PMLR, pp 1232–1300
4. Allouah Y, Guerraoui R, Gupta N, Pinot R, Stephan J (2023) On the privacy-robustness-utility trilemma in distributed learning. In: International Conference on Machine Learning, ICML 2023, 23–29 July 2023, Honolulu, Hawaii, USA. Krause A, Brunskill E, Cho K, Engelhardt B, Sabato S, Scarlett J (eds), vol 202. Proceedings of Machine Learning Research. PMLR, pp 569–626
5. Allouah Y, Guerraoui R, Gupta N, Rafaël P, Rizk G (2023) Robust distributed learning: tight error bounds and breakdown point under data heterogeneity. In: The 37th Conference on Neural Information Processing Systems
6. Baruch M, Baruch G, Goldberg Y (2019) A little is enough: circumventing defenses for distributed learning. In: Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, 8–14 December 2019, Long Beach, CA, USA
7. Blanchard P, Mhamdi EME, Guerraoui R, Stainer J (2017) Machine learning with adversaries: Byzantine tolerant gradient descent. In: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4–9, 2017, Long Beach, CA, USA. Guyon I, von Luxburg U, Bengio S, Wallach HM, Fergus R, Vishwanathan SVN, Garnett R (eds), pp 119–129

8. Blum M, Floyd RW, Pratt V, Rivest RL, Tarjan RE (1973) Time bounds for selection. J Comput Syst Sci 7(4):448–461. ISSN: 0022-0000
9. Charikar M, Steinhardt J, Valiant G (2017) Learning from untrusted data. In: Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, pp. 47–60
10. Chen Y, Su L, Xu J (2017) Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. Proc. ACM Meas. Analy. Comput. Syst. 1(2):1–25
11. Cohen MB, Lee YT, Miller GL, Pachocki J, Sidford A (2016) Geometric median in nearly linear time. In: Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18–21, 2016. ACM, pp 9–21
12. Data D, Diggavi S (2021) Byzantine-resilient high-dimensional sgd with local iterations on heterogeneous data. In: International Conference on Machine Learning. PMLR, pp 2478–2488
13. Diakonikolas I, Kamath G, Kane D, Li J, Steinhardt J, Stewart A (2019) Sever: a robust meta-algorithm for stochastic optimization. In: International Conference on Machine Learning. PMLR, pp 1596–1606
14. El Mhamdi EM, Farhadkhani S, Guerraoui R, Guirguis A, Hoang LN, Rouault S (2021) Collaborative learning in the jungle (Decentralized, byzantine, heterogeneous, asynchronous and nonconvex learning). In: The 35th Conference on Neural Information Processing Systems
15. Fang C, Yang Z, Bajwa WU (2022) BRIDGE: Byzantine-resilient decentralized gradient descent. IEEE Trans Signal Inf Process over Netw 8:610–626
16. Farhadkhani S, Guerraoui R, Gupta N, Pinot R, Stephan J (2022) Byzantine machine learning made easy by resilient averaging of momentums. In: International Conference on Machine Learning, ICML 2022, 17–23 July 2022, Baltimore, Maryland, USA, vol. 162. Proceedings of Machine Learning Research. PMLR, pp 6246–6283
17. Farhadkhani S, Guerraoui R, Villemaud O, et al (2022) An equivalence between data poisoning and Byzantine gradient attacks. In: International Conference on Machine Learning. PMLR, pp 6284–6323
18. Farhadkhani S, Guerraoui R, Gupta N, Hoang L, Pinot R, Stephan J (2023) Robust collaborative learning with linear gradient overhead. In: International Conference on Machine Learning, ICML 2023, 23–29 July 2023, Honolulu, Hawaii, USA, vol 202. Proceedings of Machine Learning Research. PMLR, pp 9761–9813
19. Guerraoui R, Gupta N, Pinot R (2023) Byzantine machine learning: a primer. In: ACM Comput Surv. Just Accepted. ISSN: 0360–0300
20. Gupta N, Vaidya NH (2020) Fault-tolerance in distributed optimization: the case of redundancy. In: Proceedings of the 39th Symposium on Principles of Distributed Computing, pp 365–374
21. Huber PJ (1981) Robust statistics. Wiley series in probability and mathematical statistics. Wiley, Hoboken
22. Karimireddy SP, He L, Jaggi M (2021) Learning from history for Byzantine Robust optimization. In: International Conference On Machine Learning, vol 139. Proceedings of Machine Learning Research
23. Karimireddy SP, He L, Jaggi M (2022) Byzantine-Robust learning on heterogeneous datasets via bucketing. In: International Conference on Learning Representations
24. Khaled A, Richtárik P (2023) Better theory for SGD in the nonconvex world. Trans Mach Learn Res. Survey Certification. ISSN: 2835-8856. https://openreview.net/forum?id=AU4qHN2VkS
25. Li L, Xu W, Chen T, Giannakis GB, Ling Q (2019) RSA: Byzantine-robust stochastic aggregation methods for distributed learning from heterogeneous datasets. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp 1544–1551
26. Liu S, Gupta N, Vaidya NH (2021) Approximate Byzantine Fault-tolerance in distributed optimization. In: Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing. PODC'21. Virtual Event, Italy: Association for Computing Machinery, pp 379–389. ISBN: 9781450385480
27. Merad I, Gaïffas S (2023) Robust methods for high-dimensional linear learning. J Mach Learn Res 24:165:1–165:44
28. PrasadA, Suggala AS, Balakrishnan S, Ravikumar P (2020) Robust estimation via robust gradient estimation. J Roy Statist Soc Ser B (Statist Methodol) 82(3):601–627

29. Rousseeuw PJ (1985) Multivariate estimation with high breakdown point. Math Statist Appl 8(37):283–297
30. Shejwalkar V, Houmansadr A (2021) Manipulating the Byzantine: optimizing model poisoning attacks and defenses for federated learning. In: NDSS
31. Stich SU (2019) Unified optimal analysis of the (stochastic) gradient method. In: arXiv preprint arXiv:1907.04232
32. Su L, Vaidya NH (2016) Fault-tolerant multi-agent optimization: optimal iterative distributed algorithms. In: Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, pp 425–434
33. Su L, Vaidya NH (2021) Byzantine-resilient multiagent optimization. IEEE Trans Autom Control 66(5):2227–2233
34. Sundaram S, Gharesifard B (2018) Distributed optimization under adversarial nodes. IEEE Trans Autom Control 64:1063–1076
35. Wu Z, Chen T, Ling Q (2023) Byzantine-resilient decentralized stochastic optimization with Robust aggregation rules. IEEE Trans Signal Process 71:3179–3195
36. Xie C, Koyejo O, Gupta I (2019) Fall of empires: breaking byzantinetolerant SGD by inner product manipulation. In: Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22–25, 2019, p 83
37. Yang Z, Bajwa WU (2019) ByRDiE: Byzantine-resilient distributed coordinate descent for decentralized learning. IEEE Trans Signal Inf Process over Netw 5(4):611–627
38. Yang Z, Gang A, Bajwa WU (2020) Adversary-resilient distributed and decentralized statistical inference and machine learning: an overview of recent advances under the Byzantine threat model. IEEE Signal Process Mag 37(3):146–159
39. Yin D, Chen Y, Ramchandran K, Bartlett PL (2018) Byzantine-Robust distributed learning: towards optimal statistical rates. In: Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10–15, 2018, vol 80. Proceedings of Machine Learning Research. PMLR, pp 5636–5645
40. Zhu B, Wang L, Pang Q, Wang S, Jiao J, Song D, Jordan MI (2023) Byzantine-Robust federated learning with optimal statistical rates. In: International Conference on Artificial Intelligence and Statistics, 25–27 April 2023, Palau de Congressos, Valencia, Spain. Ruiz FJR, Dy JG, van de Meent J (eds), vol 206. Proceedings of Machine Learning Research. PMLR, pp 3151–3178
41. Zhu J, Lin Y, Velasquez A, Liu J (2023) Resilient distributed optimization. In: American Control Conference, ACC 2023, San Diego, CA, USA, May 31–June 2, 2023. IEEE, pp 1307–1312

Chapter 5
Optimal Robustness

Abstract We presented, in the previous chapter, a way to render robustness to the classic distributed mini-batch gradient-descent (DMGD) method against (a minority of) adversarial nodes. Basically, we saw that upon replacing the simple averaging operation at the server by an aggregation scheme that satisfies the property of (f, κ)-robust averaging, the resulting *robust DMGD* method can tolerate up to f adversarial nodes. We formalized the notion of robustness against adversarial nodes through the definition of (f, ε)-resilience (and stationary resilience). Specifically, in the strongly convex case, "robustifying" DMGD with an (f, κ)-robust averaging rule is (f, ε)-resilient, i.e., outputs an ε-suboptimal model, where $\varepsilon \in O_\star\left(\kappa\sigma^2 + \kappa\zeta\right)$ upon executing a sufficiently large number of iterations T (We write $\phi(a) \in O_\star(\psi(a))$ if there exists a constant $c \in \mathbb{R}^+$ such that $\phi(a) \leq c\psi(a)$ for all $a \in \mathbb{R}^+$. Similarly, $\phi(a) \in \Omega_\star(\psi(a))$ if there exists a constant $c \in \mathbb{R}^+$ such that $\phi(a) \geq c\psi(a)$ for all $a \in \mathbb{R}^+$. These conventions are extensively used in this chapter.). A similar result was also shown in the non-convex case, but in terms of approximate stationarity. While we presented some simple (f, κ)-robust averaging rules, the resulting training error is not optimal for many of those schemes, e.g., Krum, geometric median, and coordinate-wise median. Specifically, their robustness coefficient κ is suboptimal with respect to the tolerable fraction of adversarial nodes $\frac{f}{n}$. This looseness significantly weakens the overall resilience of the learning algorithm in practical settings. In this chapter, we show how by using *pre-aggregations* schemes, specifically *nearest-neighbor mixing* (NNM) and *bucketing*, we can achieve order-optimal robustness. These schemes intervene prior to the aggregate, thereby yielding a two-step aggregation rule.

R. Guerraoui et al., *Robust Machine Learning*, Machine Learning: Foundations, Methodologies, and Applications, https://doi.org/10.1007/978-981-97-0688-4_5

5.1 Pre-aggregation by Nearest-Neighbor Mixing

In this section, we first introduce, in Sect. 5.1.1, a deterministic *pre-aggregation scheme* called *nearest-neighbor mixing* (NNM). We then show, in Sect. 5.1.2, why and how NNM enhances the robustness of an (f, κ)-robust averaging, which includes the aggregation rules presented in Chap. 4.

5.1.1 Description of Nearest-Neighbor Mixing

The main idea behind NNM is to reduce the gap between the input vectors and the true average of honest nodes. To do this, the algorithm *merges* each input vector with its nearest neighbors. Specifically, given a set of n vectors $\mathbf{v_1}, \ldots, \mathbf{v_n} \in \mathbb{R}^d$, NNM replaces every vector with the average of its $n - f$ nearest neighbors (including the vector itself). For each $i \in [n]$, NNM first finds a subset $N_i \subseteq [n]$ such that $|N_i| = n - f$, and if $j \in [n] \setminus N_i$, then $\left\|\mathbf{v_i} - \mathbf{v_j}\right\| \geq \|\mathbf{v_i} - \mathbf{v_k}\|$ for all $k \in N_i$. In other words, N_i represents the indices of $n - f$ vectors that are closest to $\mathbf{v_i}$. Ties can be broken arbitrarily. Then, the algorithm outputs $\mathrm{NNM}\,(\mathbf{v_1}, \ldots, \mathbf{v_n}) \coloneqq (\mathbf{z_1}, \ldots, \mathbf{z_n}) \in \mathbb{R}^{d \times n}$, where for each $i \in [n]$,

$$\mathbf{z_i} = \frac{1}{n-f} \sum_{j \in N_i} \mathbf{v_j}\,.$$

The mixing procedure of NNM is visualized in Fig. 5.1.

An intuition on why the set of vectors (corresponding to honest nodes) output by NNM can have a smaller heterogeneity than the set of original input vectors is as follows. Recall that H denotes the set of honest nodes. Consider two honest nodes $i, j \in H$ such that $\left\|\mathbf{v_i} - \mathbf{v_j}\right\| \geq \left\|\mathbf{v_{i'}} - \mathbf{v_{j'}}\right\|$ for all $i', j' \in H$, i.e., $\mathbf{v_i}, \mathbf{v_j}$ are

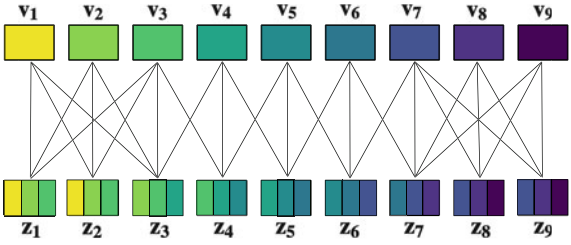

Fig. 5.1 Illustration of nearest-neighbor mixing (NNM)

maximally separated input vectors represented by H. Then,

$$\left\|\mathbf{z_i} - \mathbf{z_j}\right\| = \frac{1}{n-f} \left\| \sum_{k \in N_i \setminus N_j} \mathbf{v_k} - \sum_{k \in N_j \setminus N_i} \mathbf{v_k} \right\| .$$

Note that $\left|N_i \setminus N_j\right| = \left|N_j \setminus N_i\right| \leq f$. Therefore, we can write

$$\left\|\mathbf{z_i} - \mathbf{z_j}\right\| = \frac{1}{n-f} \times \left\| \sum_{k \in N_i \setminus N_j} (\mathbf{v_k} - \mathbf{v_i}) - \sum_{k \in N_j \setminus N_i} \left(\mathbf{v_k} - \mathbf{v_j}\right) + \left|N_i \setminus N_j\right| \left(\mathbf{v_i} - \mathbf{v_j}\right) \right\| .$$

Due to the triangle inequality, we obtain that

$$\left\|\mathbf{z_i} - \mathbf{z_j}\right\| \leq \frac{1}{n-f} \left(\sum_{k \in N_i \setminus N_j} \left\|\mathbf{v_k} - \mathbf{v_i}\right\| - \sum_{k \in N_j \setminus N_i} \left\|\mathbf{v_k} - \mathbf{v_j}\right\| + f \left\|\mathbf{v_i} - \mathbf{v_j}\right\| \right) .$$

As N_i represents $n-f$ input vectors closest to $\mathbf{v_i}$ and $|H| = n-f$, for all $k \in N_i$, we have $\left\|\mathbf{v_k} - \mathbf{v_i}\right\| \leq \max_{k' \in H} \left\|\mathbf{v_{k'}} - \mathbf{v_i}\right\| = \left\|\mathbf{v_j} - \mathbf{v_i}\right\|$. Similarly, for all $k \in N_j$, we have $\left\|\mathbf{v_k} - \mathbf{v_i}\right\| \leq \left\|\mathbf{v_j} - \mathbf{v_i}\right\|$. Using this above, we get

$$\left\|\mathbf{z_i} - \mathbf{z_j}\right\| \leq \frac{3f}{n-f} \left\|\mathbf{v_i} - \mathbf{v_j}\right\| .$$

This suggests that when f is small enough, specifically when $\frac{3f}{n-f} < 1$, the separation between the set of vectors $\{\mathbf{z_i}, i \in H\}$ might be smaller than that of the original vectors $\{\mathbf{v_i}, i \in H\}$. The above intuition is further refined below, in Sect. 5.1.2, to prove that NNM reduces the robustness coefficient, hence amplifies the robustness, of a robust averaging.

5.1.2 *Robustness Amplification*

In this section, we analyze the robustness amplification property of NNM. We show that the composition of NNM with an (f, κ)-robust rule F results in a new aggregation rule $F' := F \circ \text{NNM}$ that has an improved robustness coefficient compared to F. The analysis proceeds in two steps. First, we obtain a generic result, in Lemma 5.1, that relates the empirical variance of the vectors NNM outputs to the variance of the input vectors. Then, we build upon this result, in Theorem 5.1, to precisely characterize the robustness amplification property of NNM.

Variance Reduction

Lemma 5.1, presented below, allows us to demonstrate the variance-reduction property of NNM. Specifically, we consider arbitrarily a vector $\mathbf{u} \in \mathbb{R}^d$, a set of n vectors $\{\mathbf{v_1}, \ldots, \mathbf{v_n}\}$ from $\mathbb{R}^d$, and a subset $S \subseteq [n]$ comprising $n - f$ elements. We then obtain an upper bound on the (Euclidean) distance between the average $\bar{\mathbf{v}}_\mathbf{S}$ of vectors $\mathbf{v_i}$, $i \in S$ and the average of $n - f$ vectors in $\{\mathbf{v_1}, \ldots, \mathbf{v_n}\}$ that are nearest neighbors of $\mathbf{u}$. The bound suggests that NNM leads to a reduction in the (empirical) variance, as we elaborate immediately after presenting Lemma 5.1 (and its proof).

Lemma 5.1 *Consider an arbitrary vector $\mathbf{u} \in \mathbb{R}^d$ and an arbitrary collection of n vectors $\mathbf{v_1}, \ldots, \mathbf{v_n} \in \mathbb{R}^d$. Let $N_\mathbf{u} \subseteq [n]$ be a set of indices representing $n - f$ nearest neighbors of $\mathbf{u}$ in the set $\{\mathbf{v_1}, \ldots, \mathbf{v_n}\}$. Then, for any $S \subseteq [n]$ such that $|S| = n - f$, the following holds true:*

$$\|\mathbf{z_u} - \bar{\mathbf{v}}_\mathbf{S}\|^2 \leq \frac{4f}{(n-f)^2} \sum_{i \in S} \|\mathbf{v_i} - \mathbf{u}\|^2 \ ,$$

where $\mathbf{z_u} \coloneqq \frac{1}{n-f} \sum_{i \in N_\mathbf{u}} \mathbf{v_i}$ *and* $\bar{\mathbf{v}}_\mathbf{S} \coloneqq \frac{1}{n-f} \sum_{i \in S} \mathbf{v_i}$.

Proof Consider an arbitrary $S \subseteq [n]$ such that $|S| = n - f$. Recall that $N_\mathbf{u} \subseteq [n]$ is a set of indices of $n - f$ nearest neighbors of $\mathbf{u}$ in $\{\mathbf{v_1}, \ldots, \mathbf{v_n}\}$. Then, by definition of $\mathbf{z_u}$ and $\bar{\mathbf{v}}_\mathbf{S}$, we obtain that

$$\begin{aligned} \|\mathbf{z_u} - \bar{\mathbf{v}}_\mathbf{S}\|^2 &= \left\| \frac{1}{n-f} \sum_{i \in N_\mathbf{u}} \mathbf{v_i} - \frac{1}{n-f} \sum_{i \in S} \mathbf{v_i} \right\|^2 \\ &= \frac{1}{(n-f)^2} \left\| \sum_{i \in N_\mathbf{u}} \mathbf{v_i} - \sum_{i \in S} \mathbf{v_i} \right\|^2 . \end{aligned}$$

Further simplifying the difference in the right-hand side, we get

$$\|\mathbf{z_u} - \bar{\mathbf{v}}_\mathbf{S}\|^2 = \frac{1}{(n-f)^2} \left\| \sum_{i \in N_\mathbf{u} \setminus S} \mathbf{v_i} - \sum_{i \in S \setminus N_\mathbf{u}} \mathbf{v_i} \right\|^2 .$$

(continued)

Since both S and $|N_{\mathbf{u}}|$ are subsets of $[n]$ comprising an equal number of elements, $|N_{\mathbf{u}} \setminus S| = |S \setminus N_{\mathbf{u}}|$, we can write the above as

$$\|\mathbf{z_u} - \bar{\mathbf{v}}_S\|^2 = \frac{1}{(n-f)^2} \left\| \sum_{i \in N_{\mathbf{u}} \setminus S} (\mathbf{v_i} - \mathbf{u}) - \sum_{i \in S \setminus N_{\mathbf{u}}} (\mathbf{v_i} - \mathbf{u}) \right\|^2 .$$

Applying the triangle and the Jensen's inequalities, we obtain that

$$\|\mathbf{z_u} - \bar{\mathbf{v}}_S\|^2 \leq \frac{|S \setminus N_{\mathbf{u}}| + |N_{\mathbf{u}} \setminus S|}{(n-f)^2} \times \left[\sum_{i \in N_{\mathbf{u}} \setminus S} \|\mathbf{v_i} - \mathbf{u}\|^2 + \sum_{i \in S \setminus N_{\mathbf{u}}} \|\mathbf{v_i} - \mathbf{u}\|^2 \right] .$$

As S and $N_{\mathbf{u}} \subseteq [n]$ with $|S| = |N_{\mathbf{u}}| = n - f$, we have $|S \setminus N_{\mathbf{u}}| = |N_{\mathbf{u}} \setminus S| \leq f$. Using this above, we get

$$\|\mathbf{z_u} - \bar{\mathbf{v}}_S\|^2 \leq \frac{2f}{(n-f)^2} \left[\sum_{i \in N_{\mathbf{u}} \setminus S} \|\mathbf{v_i} - \mathbf{u}\|^2 + \sum_{i \in S \setminus N_{\mathbf{u}}} \|\mathbf{v_i} - \mathbf{u}\|^2 \right] . \tag{5.1}$$

As $N_{\mathbf{u}}$ is a set of $n - f$ nearest neighbors to $\mathbf{u}$, we obtain that

$$\sum_{i \in N_{\mathbf{u}} \setminus S} \|\mathbf{v_i} - \mathbf{u}\|^2 \leq \sum_{i \in N_{\mathbf{u}}} \|\mathbf{v_i} - \mathbf{u}\|^2 \leq \sum_{i \in S} \|\mathbf{v_i} - \mathbf{u}\|^2 . \tag{5.2}$$

As $S \setminus N_{\mathbf{u}} \subseteq S$, we also have

$$\sum_{i \in S \setminus N_{\mathbf{u}}} \|\mathbf{v_i} - \mathbf{u}\|^2 \leq \sum_{i \in S} \|\mathbf{v_i} - \mathbf{u}\|^2 . \tag{5.3}$$

Substituting from (5.2) and (5.3) in (5.1), we get

$$\|\mathbf{z_u} - \bar{\mathbf{v}}_S\|^2 \leq \frac{4f}{(n-f)^2} \sum_{i \in S} \|\mathbf{v_i} - \mathbf{u}\|^2 .$$

This concludes the proof. □

The bound presented in Lemma 5.1 implies variance reduction by NNM. Specifically, consider an arbitrary $S \subseteq [n]$ of size $n - f$ and an arbitrary $i \in S$. Then, upon substituting $\mathbf{u} = \mathbf{v_i}$ in Lemma 5.1, we get

$$\|\mathbf{z}_i - \bar{\mathbf{v}}_\mathbf{S}\|^2 \leq \frac{4f}{(n-f)^2} \sum_{j \in S} \left\|\mathbf{v_j} - \mathbf{v_i}\right\|^2 ,$$

where recall that $\mathbf{z_i}$ is as defined in Sect. 5.1.1. As the above holds true for any $i \in S$, we obtain that

$$\sum_{i \in S} \|\mathbf{z}_i - \bar{\mathbf{v}}_\mathbf{S}\|^2 \leq \frac{4f}{(n-f)^2} \sum_{i \in S} \sum_{i \in S} \left\|\mathbf{v_j} - \mathbf{v_i}\right\|^2 ,$$

where recall that $(\mathbf{z_1}, \ldots \mathbf{z_n}) \coloneqq \text{NNM}\,(\mathbf{v_1}, \ldots, \mathbf{v_n})$. Note, as explained earlier in Remark 4.5, that $\sum_{i \in S} \sum_{i \in S} \left\|\mathbf{v_j} - \mathbf{v_i}\right\|^2 = 2\,|S| \sum_{i \in S} \|\mathbf{v_i} - \bar{\mathbf{v}}_\mathbf{S}\|^2$. Therefore, from above we obtain that

$$\frac{1}{|S|} \sum_{i \in S} \|\mathbf{z}_i - \bar{\mathbf{v}}_\mathbf{S}\|^2 \leq \frac{8f}{n-f} \cdot \frac{1}{|S|} \sum_{i \in S} \|\mathbf{v_i} - \bar{\mathbf{v}}_\mathbf{S}\|^2 . \tag{5.4}$$

Let $\bar{\mathbf{z}}_\mathbf{S} \coloneqq \frac{1}{|S|} \sum_{i \in S} \mathbf{z_i}$. Then, we obtain that

$$\sum_{i \in S} \|\mathbf{z}_i - \bar{\mathbf{v}}_\mathbf{S}\|^2 = \sum_{i \in S} \|\mathbf{z}_i - \bar{\mathbf{z}}_\mathbf{S} - (\bar{\mathbf{v}}_\mathbf{S} - \bar{\mathbf{z}}_\mathbf{S})\|^2 = \sum_{i \in S} \|\mathbf{z}_i - \bar{\mathbf{z}}_\mathbf{S}\|^2 + |S|\,\|\bar{\mathbf{v}}_\mathbf{S} - \bar{\mathbf{z}}_\mathbf{S}\|^2 .$$

Substituting from above in (5.4), we obtain that

$$\frac{1}{|S|} \sum_{i \in S} \|\mathbf{z}_i - \bar{\mathbf{z}}_\mathbf{S}\|^2 + \|\bar{\mathbf{v}}_\mathbf{S} - \bar{\mathbf{z}}_\mathbf{S}\|^2 \leq \frac{8f}{n-f} \cdot \frac{1}{|S|} \sum_{i \in S} \|\mathbf{v_i} - \bar{\mathbf{v}}_\mathbf{S}\|^2 . \tag{5.5}$$

This implies that the variance of the vectors that NNM outputs corresponding to the honest nodes, i.e., when $S = H$, is smaller than the original variance, provided that the fraction of adversarial nodes $\frac{f}{n}$ is sufficiently small (such that $\frac{8f}{n-f} < 1$). Specifically, due to (5.5), we have

$$\frac{1}{|H|} \sum_{i \in H} \|\mathbf{z}_i - \bar{\mathbf{z}}_\mathbf{S}\|^2 \leq \frac{8f}{n-f} \cdot \frac{1}{|H|} \sum_{i \in H} \|\mathbf{v_i} - \bar{\mathbf{v}}_\mathbf{S}\|^2 .$$

In the following, we use the above analysis, specifically (5.5), to derive the amplification in the robustness of a robust aggregation rule imparted by NNM.

Provably Tighter Robustness Coefficient

The aforementioned variance-reduction property of NNM enables us to show how this scheme enhances the robustness coefficient of an (f, κ)-robust averaging. Specifically, the following result holds true.

Theorem 5.1 *Consider an aggregation rule $F : \mathbb{R}^{d \times n} \to \mathbb{R}^d$. If F is an (f, κ)-robust averaging, then $F \circ \mathrm{NNM}$ is an (f, κ')-robust averaging where*

$$\kappa' \leq \frac{8f}{n-f}(\kappa + 1)\,.$$

Proof Consider n arbitrary vectors $\mathbf{v_1}, \ldots, \mathbf{v_n} \in \mathbb{R}^d$, and an arbitrary $S \subseteq [n]$ such that $|S| = n - f$. Denote $(\mathbf{z_1}, \ldots, \mathbf{z_n}) \coloneqq \mathrm{NNM}(\mathbf{v_1}, \ldots, \mathbf{v_n})$ and $\overline{\mathbf{z}}_\mathbf{S} \coloneqq \frac{1}{|S|}\sum_{i \in S} \mathbf{z_i}$. Since F is assumed to be an (f, κ)-robust averaging, the following holds true:

$$\begin{aligned} \|F \circ \mathrm{NNM}(\mathbf{v_1}, \ldots, \mathbf{v_n}) - \overline{\mathbf{z}}_\mathbf{S}\|^2 &= \|F(\mathbf{z_1}, \ldots, \mathbf{z_n}) - \overline{\mathbf{z}}_\mathbf{S}\|^2 \\ &\leq \frac{\kappa}{|S|}\sum_{i \in S} \|\mathbf{z_i} - \overline{\mathbf{z}}_\mathbf{S}\|^2\,. \end{aligned} \tag{5.6}$$

Note that for any real values a_1, a_2 and $a_3 > 0$, due to Young's inequality, we have $(a_1 + a_2)^2 \leq (1 + a_3)(a_1)^2 + (1 + \frac{1}{a_3})(a_2)^2$. Therefore, using the triangle inequality, we obtain that

$$\begin{aligned} \|F(\mathbf{z_1}, \ldots, \mathbf{z_n}) - \overline{\mathbf{v}}_\mathbf{S}\|^2 &= \|F(\mathbf{z_1}, \ldots, \mathbf{z_n}) - \overline{\mathbf{z}}_\mathbf{S} + \overline{\mathbf{z}}_\mathbf{S} - \overline{\mathbf{v}}_\mathbf{S}\|^2 \\ &\leq \left(1 + \frac{1}{\kappa}\right)\|F(\mathbf{z_1}, \ldots, \mathbf{z_n}) - \overline{\mathbf{z}}_\mathbf{S}\|^2 \\ &\quad + (1 + \kappa)\,\|\overline{\mathbf{z}}_\mathbf{S} - \overline{\mathbf{v}}_\mathbf{S}\|^2\,. \end{aligned}$$

Substituting from (5.6) in the above, we get

$$\begin{aligned} \|F(\mathbf{z_1}, \ldots, \mathbf{z_n}) - \overline{\mathbf{v}}_\mathbf{S}\|^2 &\leq \left(1 + \frac{1}{\kappa}\right)\frac{\kappa}{|S|}\sum_{i \in S}\|\mathbf{z_i} - \overline{\mathbf{z}}_\mathbf{S}\|^2 \\ &\quad + (1 + \kappa)\,\|\overline{\mathbf{z}}_\mathbf{S} - \overline{\mathbf{v}}_\mathbf{S}\|^2 \\ &= (1 + \kappa)\left(\frac{1}{|S|}\sum_{i \in S}\|\mathbf{z_i} - \overline{\mathbf{z}}_\mathbf{S}\|^2 + \|\overline{\mathbf{z}}_\mathbf{S} - \overline{\mathbf{v}}_\mathbf{S}\|^2\right)\,. \end{aligned} \tag{5.7}$$

(continued)

Substituting from (5.5) in (5.7), we obtain that

$$\|F(\mathbf{z_1}, \ldots, \mathbf{z_n}) - \overline{\mathbf{v}}_S\|^2 \leq \frac{8f(\kappa+1)}{n-f} \cdot \frac{1}{|S|} \sum_{i \in S} \|\mathbf{v_i} - \overline{\mathbf{v}}_S\|^2 \ .$$

This completes the proof. □

Recall that the lower bound for the robustness coefficient κ is in $\frac{f}{n-2f} \in \Omega_n\left(\frac{f}{n}\right)$ (see Appendix C.1). This observation, in conjunction with Theorem 5.1, implies that if F has a robustness coefficient $\kappa \in O_n(1)$, the two-step aggregation rule $F' = F \circ \text{NNM}$ has an order-optimal robustness coefficient $\kappa' \in O_n\left(\frac{f}{n}\right)$. Note that the condition $\kappa \in O_n(1)$ is fairly general, thereby incorporating many standard robust aggregation rules, including those presented in Chap. 4. Specifically, if there exists a real value $\nu > 0$ such that $n \geq (2+\nu)f$, then all the aforementioned aggregation rules (with parameter $q = f$) are (f, κ)-robust with $\kappa \in O_n(1)$. Hence, composing any of the aggregation rules from Chap. 4 with NNM yields an (order-)optimal robust averaging. Table 5.1 presents the values for κ' obtained upon composing different aggregation rules with NNM. Even though CWTM achieves an order-optimal robustness coefficient, i.e., $\kappa \in O_n\left(\frac{f}{n}\right)$, without the use of NNM, the use of NNM significantly improves the empirical performance of CWTM.[1]

Table 5.1 List of robustness coefficients κ' for robust aggregation rules from Chap. 4 when composed with NNM. The values are computed by applying Theorem 5.1. The last column provides the order of κ' assuming that there exists a real value $\nu > 0$ such that $n \geq (2+\nu)f$

Aggregation rule	Robustness coefficient κ'	κ' when $n \geq (2+\nu)f$
CWMed	$\frac{8f}{n-f}\left(4\left(1+\frac{f}{n-f}\right)^2+1\right)$	$O_\star\left(\frac{f}{n}\right)$
GeoMed	$\frac{8f}{n-f}\left(4\left(1+\frac{f}{n-2f}\right)^2+1\right)$	
CWTM$_f$	$\frac{8f}{n-f}\left(\frac{6f}{n-2f}\left(1+\frac{f}{n-2f}\right)+1\right)$	
Multi-Krum$_{f,1}$	$\frac{8f}{n-f}\left(6\left(1+\frac{f}{n-2f}\right)+1\right)$	

[1] See paper [1] for further details on this phenomenon.

Limitations of NNM

The main computational bottleneck of NNM is the search of nearest neighbors, which has time complexity $O_n\left(dn^2\right)$. Nevertheless, this remains comparable to (or even smaller than) several aggregation rules, including Multi-Krum and MVA. Furthermore, unlike *spectral schemes*, which have been omitted from this book, NNM preserves linear dependency on the model size, i.e., d, which could be extremely large in modern-day machine learning (usually, $d \gg n$).[2] Furthermore, note that the robustness amplification NNM offers is in order of magnitude. In particular, as shown in Theorem 5.1, NNM essentially scales the robustness coefficient of the original aggregation rule by a factor of $\frac{8f}{n-f}$ that belongs to $O_n\left(\frac{f}{n}\right)$. Although this yields an amplification in the robustness when $\frac{f}{n}$ is small, the same need not be true in the case when $\frac{f}{n} \leq \frac{\kappa}{8+9\kappa}$ where κ is the robustness coefficient of the original aggregation rule. Hence, for aggregation rules that have large robustness coefficients, NNM can only guarantee improvement in the robustness in the low tolerance regime, i.e., the robustness parameter f cannot be arbitrarily close to $\frac{n}{2}$. Nevertheless, empirical observations show overwhelmingly that NNM indeed improves the robustness of most existing (f, κ)-robust averaging rules.

5.2 Pre-aggregation by Bucketing

In this section, we introduce another pre-aggregation scheme for fixing the suboptimality of median-based aggregation rules. As mentioned earlier, the main bottleneck of NNM is the computational cost of the method. To circumvent this, we can replace the nearest-neighbor operation in NNM by *random bucketing* of input vectors. The resulting scheme, which is simply called *Bucketing*, also reduces the dispersion of the input vectors, but only in *expectation*. We describe the bucketing procedure in Sect. 5.2.1 and demonstrate, in Sect. 5.2.2, the main property of this scheme, specifically the stochastic variance reduction. In Sect. 5.2.2, we also discuss how this method, though more computationally efficient than NNM, can suffer some limitations in terms of the robustness guarantee.

5.2.1 Description of Bucketing

Bucketing is a pre-aggregation method, parameterized by an integer $s \in [n]$, that averages random subsets of s inputs, prior to feeding them to an aggregation rule F. Given a collection of n vectors $\mathbf{v_1}, \ldots, \mathbf{v_n} \in \mathbb{R}^d$, bucketing first randomly shuffles the inputs. Specifically, consider a random permutation $\tau : \{1, \ldots, n\} \rightarrow$

[2] Examples of spectral aggregation can be found in papers [3, 4, 15].

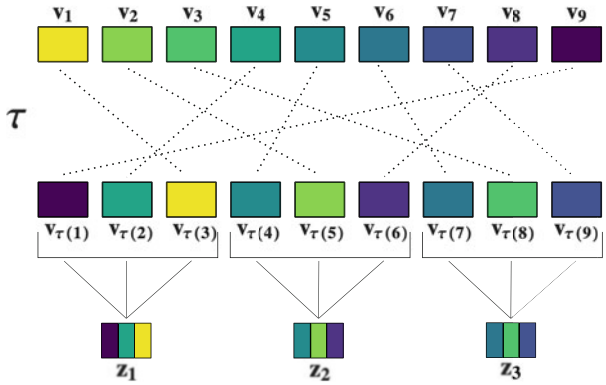

Fig. 5.2 Illustration of the Bucketing pre-aggregation scheme

$\{1, \ldots, n\}$ with $\mathbb{P}[\tau(i) = j] = \frac{1}{n}$ for all $i, j \in [n]$. Bucketing uses τ to create a natural order on $\mathbf{v_1}, \ldots, \mathbf{v_n}$. Denoting $\mathbf{v_{\tau(1)}}, \ldots, \mathbf{v_{\tau(n)}}$ the ordered vectors, the scheme partitions the ordered set $\left(\mathbf{v_{\tau(1)}}, \ldots, \mathbf{v_{\tau(n)}}\right)$ into $\lceil n/s \rceil$ *buckets*, denoted by $\mathcal{B}_1, \ldots, \mathcal{B}_{\lceil n/s \rceil}$. For every $i \in [\lceil n/s \rceil]$, we have

$$\mathcal{B}_i = \{\tau\left((i-1)s + 1\right), \ldots, \tau\left(\min\{is, n\}\right)\}.$$

Essentially, the algorithm creates $\lceil n/s \rceil$ buckets and fills them one by one with s vectors chosen sequentially from the set $\left(\mathbf{v_{\tau(1)}}, \ldots, \mathbf{v_{\tau(n)}}\right)$. If n is a multiple of s, then each bucket has s vectors. Otherwise, we would have one bucket of size r where $r = n$ modulo s. The construction of the buckets is illustrated in Fig. 5.2.

Ultimately, the scheme feeds the average of the vectors in each bucket to the aggregation rule at hand. Specifically, Bucketing outputs $\textsc{Buck}_s(\mathbf{v_1}, \ldots, \mathbf{v_n}) \coloneqq \left(\mathbf{z_1}, \ldots, \mathbf{z_{\lceil n/s \rceil}}\right)$, where for each $i \in [\lceil n/s \rceil]$, we have

$$\mathbf{z_i} = \frac{1}{|\mathcal{B}_i|} \sum_{j \in \mathcal{B}_i} \mathbf{v_j}.$$

The key rationale for Bucketing is that the set of averaged vectors $\mathbf{z_1}, \ldots, \mathbf{z_{\lceil n/s \rceil}}$ has a variance that is lower than that of the original input vectors $\mathbf{v_1}, \ldots, \mathbf{v_n}$. Thus, similarly to NNM, when we compose a robust averaging scheme with Bucketing, the robustness coefficient of the rule improves. The main advantage of Bucketing over NNM is the computational cost, which for Bucketing is essentially the same as that of the classic arithmetic averaging. Note, however, that Bucketing induces additional randomness in the aggregation procedure due to the shuffling operation it uses for

creating buckets. This source of randomness cannot be overlooked in practice, even when the honest nodes compute the exact true gradients of their local loss functions. This amplifies the uncertainty in each iteration of the learning procedure, hence resulting in a weaker *probabilistic resilience* guarantee, unlike NNM. We elaborate this point further in the next section.

5.2.2 Robustness Amplification for Bucketing

Here, we analyze the robustness amplification property of Bucketing. We show that Bucketing, similarly to NNM, can reduce the dispersion of the vectors effectively fed to the aggregation rule. In fact, it reduces the average distance to the average of the honest inputs. We then highlight some limitations of this method.

Variance Reduction with Respect to the Honest Average
Consider a set of vectors $\mathbf{v_1}, \ldots, \mathbf{v_n}$ and let $(\mathbf{z_1}, \ldots, \mathbf{z}_{\lceil \mathbf{n}/s \rceil}) \coloneqq \textsc{Buck}_s(\mathbf{v_1}, \ldots, \mathbf{v_n})$. We show in Theorem 5.2 that the average distance of $\bar{\mathbf{v}}_\mathbf{S}$, i.e., the average of the vectors represented by $S \subseteq [n]$ comprising $n - f$ elements, from the vectors $\mathbf{z_1}, \ldots, \mathbf{z}_{\lceil \mathbf{n}/s \rceil}$ (output by Bucketing) is smaller than the average distance of $\bar{\mathbf{v}}_\mathbf{S}$ from the initial vectors $\mathbf{v_1}, \ldots, \mathbf{v_n}$. Theorem 5.2 demonstrates that Bucketing reduces the dispersion of the honest nodes' vectors around their average.

Theorem 5.2 *Consider n arbitrary vectors $\mathbf{v_1}, \ldots, \mathbf{v_n} \in \mathbb{R}^d$ and an arbitrary $s \in [n]$ such that n is divisible by s. Let $(\mathbf{z_1}, \ldots, \mathbf{z_{n/s}}) \coloneqq \textsc{Buck}_s(\mathbf{v_1}, \ldots, \mathbf{v_n})$. Then, for any $S \subseteq [n]$ such that $|S| = n - f > 1$, denoting $\tilde{S} \coloneqq \{i \in [n/s] \mid \mathcal{B}_i \subseteq S\}$, the following holds true:*

$$\mathbb{E}_\tau \left[\frac{1}{\left|\tilde{S}\right|} \sum_{i \in \tilde{S}} \left\| \mathbf{z_i} - \bar{\mathbf{v}}_\mathbf{S} \right\|^2 \right] \leq \frac{1}{s} \cdot \frac{1}{|S|} \sum_{i \in S} \left\| \mathbf{v_i} - \bar{\mathbf{v}}_\mathbf{S} \right\|^2 ,$$

where the expectation $\mathbb{E}_\tau[\cdot]$ is over the distribution of the random permutation τ used in Bucketing, and $\bar{\mathbf{v}}_\mathbf{S} \coloneqq \frac{1}{S} \sum_{i \in S} \mathbf{v_i}$.

Proof Consider an arbitrary $S \subseteq [n]$ such that $|S| = n - f$. First, note that $\tilde{S}$ represents the indices of output vectors $\mathbf{z_1}, \ldots, \mathbf{z_{n/s}}$ that correspond to the buckets containing vectors only from the set S. Let $P_{\tilde{S}}$ denote the set of all $\tilde{S}$'s

(continued)

obtained over all possible permutation mappings $[n] \to [n]$. Then, we have

$$\mathbb{E}_\tau\left[\frac{1}{\left|\tilde{S}\right|}\sum_{i\in\tilde{S}}\|\mathbf{z_i}-\overline{\mathbf{v}}_\mathbf{S}\|^2\right]=\frac{1}{\left|P_{\tilde{S}}\right|}\sum_{\tilde{S}\in P_{\tilde{S}}}\mathbb{E}_\tau\left[\frac{1}{\left|\tilde{S}\right|}\sum_{i\in\tilde{S}}\|\mathbf{z_i}-\overline{\mathbf{v}}_\mathbf{S}\|^2\,\middle|\,\tilde{S}\right]$$

$$=\frac{1}{\left|P_{\tilde{S}}\right|}\sum_{\tilde{S}\in P_{\tilde{S}}}\left(\frac{1}{\left|\tilde{S}\right|}\sum_{i\in\tilde{S}}\mathbb{E}_\tau\left[\|\mathbf{z_i}-\overline{\mathbf{v}}_\mathbf{S}\|^2\mid\tilde{S}\right]\right).$$

Hence, it suffices to show that, for any $i \in [n/s]$,

$$\mathbb{E}_\tau\left[\|\mathbf{z_i}-\overline{\mathbf{v}}_\mathbf{S}\|^2\cdot\mathbb{1}\left\{\mathcal{B}_i\subseteq S\right\}\right]\le\frac{1}{s}\cdot\frac{1}{|S|}\sum_{i\in S}\|\mathbf{v_i}-\overline{\mathbf{v}}_\mathbf{S}\|^2\,. \tag{5.8}$$

Given the random set $\mathcal{B}_i$, by the triangle and the Jensen's inequalities, we have

$$\|\mathbf{z_i}-\overline{\mathbf{v}}_\mathbf{S}\|^2=\left\|\frac{1}{|\mathcal{B}_i|}\sum_{k\in\mathcal{B}_i}(\mathbf{v_k}-\overline{\mathbf{v}}_\mathbf{S})\right\|^2\le\frac{1}{|\mathcal{B}_i|}\sum_{k\in\mathcal{B}_i}\|\mathbf{v_k}-\overline{\mathbf{v}}_\mathbf{S}\|^2\,.$$

Recall that n is assumed to be divisible by s; hence, all buckets are of size s, i.e., $|\mathcal{B}_i| = s$. We can write the above as

$$\|\mathbf{z_i}-\overline{\mathbf{v}}_\mathbf{S}\|^2\le\frac{1}{s}\sum_{k\in\mathcal{B}_i}\|\mathbf{v_k}-\overline{\mathbf{v}}_\mathbf{S}\|^2=\frac{1}{s}\mathbb{E}_{k\sim\mathcal{B}_i}\left[\|\mathbf{v_k}-\overline{\mathbf{v}}_\mathbf{S}\|^2\right],$$

where $k \sim \mathcal{B}_i$ means that k is uniformly distributed over the set $\mathcal{B}_i$. Taking the conditional expectation $\mathbb{E}_\tau[\cdot\,|\,\mathcal{B}_i]$ on both sides, we get

$$\mathbb{E}_\tau\left[\|\mathbf{z_i}-\overline{\mathbf{v}}_\mathbf{S}\|^2\,\middle|\,\mathcal{B}_i\right]\le\frac{1}{s}\,\mathbb{E}_{k\sim\mathcal{B}_i}\left[\|\mathbf{v_k}-\overline{\mathbf{v}}_\mathbf{S}\|^2\,\middle|\,\mathcal{B}_i\right]. \tag{5.9}$$

Note that $\mathbb{E}_\tau\left[\|\mathbf{z_i}-\overline{\mathbf{v}}_\mathbf{S}\|^2\cdot\mathbb{1}\left\{\mathcal{B}_i\subseteq S\right\}\right]=\mathbb{E}_\tau\left[\|\mathbf{z_i}-\overline{\mathbf{v}}_\mathbf{S}\|^2\,\middle|\,\mathcal{B}_i\right]\mathbb{P}_\tau\left[\mathcal{B}_i\subseteq S\right]$. Substituting from (5.9), we obtain that

$$\mathbb{E}_\tau\left[\|\mathbf{z_i}-\overline{\mathbf{v}}_\mathbf{S}\|^2\cdot\mathbb{1}\left\{\mathcal{B}_i\subseteq S\right\}\right]\le\frac{1}{s}\,\mathbb{E}_{k\sim\mathcal{B}_i}\left[\|\mathbf{v_k}-\overline{\mathbf{v}}_\mathbf{S}\|^2|\mathcal{B}_i\right]\mathbb{P}_\tau\left[\mathcal{B}_i\subseteq S\right]. \tag{5.10}$$

(continued)

We can write

$$\mathbb{E}_{k\sim\mathcal{B}_i}\left[\|\mathbf{v_k}-\bar{\mathbf{v}}_\mathbf{S}\|^2\right]\mathbb{P}_\tau\left[\mathcal{B}_i\subseteq S\right]$$
$$=\mathbb{E}_\tau\left[\mathbb{E}_{k\sim\mathcal{B}_i}\left[\|\mathbf{v_k}-\bar{\mathbf{v}}_\mathbf{S}\|^2|\mathcal{B}_i\right]\cdot\mathbb{1}\left\{\mathcal{B}_i\subseteq S\right\}\right].$$

Since τ has uniform distribution over the set of all possible permutation mappings $[n]\to[n]$, the right-hand side in the above is equivalent to the expectation of $\|\mathbf{v_k}-\bar{\mathbf{v}}_\mathbf{S}\|^2$, where k is a random variable with uniform distribution over S. Specifically, we have

$$\mathbb{E}_\tau\left[\mathbb{E}_{k\sim\mathcal{B}_i}\left[\|\mathbf{v_k}-\bar{\mathbf{v}}_\mathbf{S}\|^2|\mathcal{B}_i\right]\cdot\mathbb{1}\left\{\mathcal{B}_i\subseteq S\right\}\right]=\mathbb{E}_{k\sim S}\left[\|\mathbf{v_k}-\bar{\mathbf{v}}_\mathbf{S}\|^2\right],$$

where $k\sim S$ simply means that k is uniformly distributed over S. Therefore,

$$\mathbb{E}_{k\sim\mathcal{B}_i}\left[\|\mathbf{v_k}-\bar{\mathbf{v}}_\mathbf{S}\|^2\right]\mathbb{P}_\tau\left[\mathcal{B}_i\subseteq S\right]=\mathbb{E}_{k\sim S}\left[\|\mathbf{v_k}-\bar{\mathbf{v}}_\mathbf{S}\|^2\right].$$

Substituting from above in (5.10) yields

$$\mathbb{E}_\tau\left[\|\mathbf{z_i}-\bar{\mathbf{v}}_\mathbf{S}\|^2\cdot\mathbb{1}\left\{\mathcal{B}_i\subseteq S\right\}\right]\leq\frac{1}{s}\mathbb{E}_{k\sim S}\left[\|\mathbf{v_k}-\bar{\mathbf{v}}_\mathbf{S}\|^2\right].$$

As $\mathbb{E}_{k\sim S}\left[\|\mathbf{v_k}-\bar{\mathbf{v}}_\mathbf{S}\|^2\right]=\frac{1}{|S|}\sum_{k\in S}\|\mathbf{v_k}-\bar{\mathbf{v}}_\mathbf{S}\|^2$, the above proves (5.8), which concludes the proof.

Limitations of Bucketing

Recall that the property that enables NNM to boost the robustness of aggregations is that it deterministically reduces the average distance to the honest average, as shown in (5.4). But, unlike NNM, Bucketing only reduces this distance *in expectation* (over the random permutations), as shown in Theorem 5.2. This means that Bucketing can potentially improve the robustness coefficients of an aggregation rule, but only stochastically. In particular, there could exist iterations in the learning procedure where Bucketing does not reduce the distance to the honest average. These iterations provide an opportunity to the adversarial nodes to disrupt the learning procedure. Furthermore, as the creation of buckets reduces the number of vectors fed to the aggregation, it reduces the overall breakdown point. Indeed, in some executions, Bucketing could assign all the f adversarial inputs to different buckets. In which case, the adversarial nodes can arbitrarily control the average of f buckets, i.e., render the average of f buckets adversarial, which effectively augments the fraction of adversarial vectors fed to the aggregation. Specifically, because the number of buckets are $\lceil n/s\rceil$, the fraction of adversarial inputs after Bucketing is $\frac{f}{\lceil n/s\rceil}$ in the

worst-case scenario. This implies that the bucket size s can be at most $\left\lfloor \frac{n}{2f} \right\rfloor$ in order to ensure that $\frac{f}{\lceil n/s \rceil} < \frac{1}{2}$ (i.e., only a minority of the vectors output by Bucketing are adversarial). This means that when $f \approx n/2$, s cannot be more than 1, which is equivalent to doing no pre-aggregation step at all.

5.3 Order-Optimal Training Error

We presented above two meta schemes, specifically, *nearest-neighbor mixing* and *bucketing*, to address suboptimal robustness of some of the aggregation rules presented in Chap. 4. In this section, we first highlight that (for large enough batch-size b and the number of iterations T) incorporating these methods in robust DMGD yields a training error in $\varepsilon \in O_\star\left(\frac{f}{n}\zeta\right)$. Then, we show that this error is unavoidable under the set of assumptions we consider, especially the gradient dissimilarity assumption (i.e., Assumption 4.1), hence making the algorithm order optimal in terms of robustness against adversarial nodes.

5.3.1 Upper Bound on the Training Error

We begin by highlighting that, upon replacing the classic averaging operation in DMGD by a composition of a pre-aggregation such as NNM and an (f, κ)-robust averaging, the server can obtain an ε-approximate optimal (or stationary) model where $\varepsilon \in O_{T,n}\left(\frac{f}{n}\left(\frac{\sigma^2}{b} + \zeta\right)\right)$. To see this, let us recall, from Chap. 4, that using an (f, κ)-robust averaging in DMGD yields a training error $\varepsilon \in O_T\left(\sqrt{\frac{\sigma^2}{b(n-f)T}} + \kappa\frac{\sigma^2}{b} + \kappa\zeta\right)$ in the non-convex case, and $\varepsilon \in O_T\left(\frac{\sigma^2}{(n-f)T} + \kappa\sigma^2 + \kappa\zeta\right)$ in the strongly convex case. Recall that the former error is with respect to the stationarity of the loss function, while the latter is with respect to the optimal value of the loss function. Now, assuming that $\kappa \in O_n(1)$ and that we use NNM as a pre-aggregation, as per Table 5.1, we obtain training errors

$$\varepsilon \in O_{T,n}\left(\sqrt{\frac{\sigma^2}{b(n-f)T}} + \frac{f}{n}\frac{\sigma^2}{b} + \frac{f}{n}\zeta\right) \text{ and}$$

$$\varepsilon \in O_{T,n}\left(\frac{\sigma^2}{b(n-f)T} + \frac{f}{n}\frac{\sigma^2}{b} + \frac{f}{n}\zeta\right),$$

respectively, for the non-convex and strongly convex cases. Note that, if we choose the batch-size $b \in \Omega_\star\left(\frac{\sigma^2}{\zeta}\right)$, we get[3]

$$\varepsilon \in O_{T,n}\left(\sqrt{\frac{\sigma^2}{b(n-f)T}} + \frac{f}{n}\zeta\right) \text{ and } \varepsilon \in O_{T,n}\left(\frac{\sigma^2}{b(n-f)T} + \frac{f}{n}\zeta\right),$$

respectively, for the non-convex and strongly convex cases. Finally, setting the number of iterations $T \in \Omega_\star\left(\frac{n^2}{f^2(n-f)}\frac{1}{\zeta^2}\right)$ and $T \in \Omega_\star\left(\frac{n}{f(n-f)}\frac{1}{\zeta}\right)$, respectively, for the non-convex and strongly convex cases, we get training error $\varepsilon \in O_\star\left(\frac{f}{n}\zeta\right)$ in both cases. We show in the following that this training error is order optimal, in the sense that no distributed algorithm (either in a sever-based or a peer-to-peer communication topology) can yield a training error smaller than $\frac{f}{n}\zeta$ times a multiplicative factor. This claim is formally shown in Sect. 5.3.2.

5.3.2 *Lower Bound on the Training Error*

In this section, we analyze in Theorem 5.3 the contribution of the gradient dissimilarity ζ toward the training error of any distributed machine learning algorithm (including robust DMGD). Essentially, we show that under Assumption 4.1, the error is unavoidably $\Omega_\star\left(\frac{f}{n}\zeta\right)$ in the presence of f adversarial nodes. However, note that the theorem does not imply a lower bound for general distributed machine learning tasks under adversarial nodes, as the assumption of uniformly bounded gradient dissimilarity (i.e., Assumption 4.1) need not hold true for many machine learning problems (see Sect. 5.4). Yet, the result is a good indicator for the suboptimality of robust DMGD (or its variants), in terms of the asymptotic training error.

Theorem 5.3 *Suppose Assumptions 3.1, 3.2, and 4.1. Let $f > 0$ and $n > 2f$. Then, a distributed algorithm is (f, ε)-stationary resilient only if $\varepsilon \geq \frac{f}{4n}\zeta$.*

Proof To prove the theorem, we consider a mean estimation problem, a special case of the learning problem formalized in Chap. 2, where $\mathcal{X} = \mathcal{Y} = \mathbb{R}^d$, $\Theta = \mathbb{R}^d$, and $h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}$ for all $\mathbf{x} \in \mathbb{R}^d$. Given a parameter $\boldsymbol{\theta}$ and data

(continued)

[3] Here we implicitly assume that m is also in $\Omega_\star\left(\frac{\sigma^2}{\zeta}\right)$. Otherwise, it suffices to take $b = m$, i.e., compute the full gradient to remove any term related to the gradient stochasticity.

point $(\mathbf{x}, y)$, we consider a quadratic loss function

$$\ell\left(h_{\boldsymbol{\theta}}\left(\mathbf{x}\right), y\right) \coloneqq \frac{L}{2}\left\|h_{\boldsymbol{\theta}}\left(\mathbf{x}\right) - y\right\|^2 = \frac{L}{2}\left\|\boldsymbol{\theta} - y\right\|^2.$$

It is easy to check that Assumption 3.1, i.e., Lipschitz smoothness, holds true for the above loss function. We consider a distributed setting with n nodes, where each node i samples a single data point (i.e., $m = 1$) from its local data distribution $\mathcal{D}_i$. Suppose that the local distribution $\mathcal{D}_i$ for node $i \in \{1, \ldots, n-f\}$ characterizes a Dirac delta distribution at $(0, 0) \in \mathbb{R}^d \times \mathbb{R}^d$, i.e., the probability of sampling $(0, 0)$ according to $\mathcal{D}_i$ is 1. Furthermore, suppose that $\mathcal{D}_i$ for node $i \in \{n-f+1, \ldots, n-f\}$ is a Dirac delta distribution at $(\mathbf{z}, \mathbf{z})$, where $\mathbf{z} \in \mathbb{R}^d$ is such that

$$\|\mathbf{z}\|^2 = \frac{(n-f)^2}{f(n-2f)L^2}\,\zeta\,.$$

By definition of the above distributions, it is also easy to check that Assumption 3.2 also holds true, with the covariance trace $\sigma^2 = 0$. Next, we will consider two possible scenarios. In the first scenario, the set of honest nodes is $H_1 = \{1, \ldots, n-f\}$. In the second scenario, the set of honest nodes is $H_2 = \{f+1, \ldots, n\}$. Observe that in the first scenario, as all the local loss functions are identical to $\mathcal{L}_{H_1}(\boldsymbol{\theta}) \coloneqq \frac{L}{2}\|\boldsymbol{\theta}\|^2$, the gradient dissimilarity is zero, i.e., Assumption 4.1 is trivially satisfied. In the second scenario, observe that the local loss function $\mathcal{L}_i(\boldsymbol{\theta}) = \frac{L}{2}\|\boldsymbol{\theta}\|^2$ for all $i \in \{f+1, \ldots, n-f\}$, and $\mathcal{L}_i(\boldsymbol{\theta}) = \frac{L}{2}\|\boldsymbol{\theta} - \mathbf{z}\|^2$ for all $i \in \{n-f, \ldots, n\}$. Thus,

$$\mathcal{L}_{H_2}(\boldsymbol{\theta}) \coloneqq \frac{L}{2(n-f)}\left((n-2f)\|\boldsymbol{\theta}\|^2 + f\|\boldsymbol{\theta} - \mathbf{z}\|^2\right).$$

Therefore, for all $\boldsymbol{\theta}$, we have

$$\nabla\mathcal{L}_{H_2}(\boldsymbol{\theta}) = L\left(\boldsymbol{\theta} - \frac{f}{n-f}\,\mathbf{z}\right).$$

(continued)

From above, we obtain that, for any $\boldsymbol{\theta}$, we have

$$\begin{aligned}
&\sum_{i \in H_2} \left\| \nabla \mathcal{L}_i(\boldsymbol{\theta}) - \nabla \mathcal{L}_{H_2}(\boldsymbol{\theta}) \right\|^2 \\
&= (n-2f) \left\| L\boldsymbol{\theta} - L\left(\boldsymbol{\theta} - \frac{f}{n-f}\mathbf{z}\right) \right\|^2 \\
&\quad + f \left\| L(\boldsymbol{\theta} - \mathbf{z}) - L\left(\boldsymbol{\theta} - \frac{f}{n-f}\mathbf{z}\right) \right\|^2 \\
&= \frac{L^2 \|\mathbf{z}\|^2}{(n-f)^2} \left((n-2f) f^2 + f(n-2f)^2 \right) = \frac{f(n-2f)L^2}{(n-f)} \|\mathbf{z}\|^2 .
\end{aligned}$$

Hence, we get

$$\frac{1}{n-f} \sum_{i \in H_2} \left\| \nabla \mathcal{L}_i(\boldsymbol{\theta}) - \nabla \mathcal{L}_{H_2}(\boldsymbol{\theta}) \right\|^2 = \frac{f(n-2f)L^2}{(n-f)^2} \|\mathbf{z}\|^2 = \zeta .$$

The above implies that Assumption 4.1 is also satisfied in the second scenario. Hence, we conclude that if there exists a distributed algorithm Π that is guaranteed to be (f, ε)-stationary resilience under Assumptions 3.1 and 3.2 and 4.1, then by Definition 4.2 the output of Π, denoted by $\hat{\boldsymbol{\theta}}$, should satisfy the condition (4.10) in both the scenarios. Specifically, for an (f, ε)-stationary resilient distributed algorithm Π with output $\hat{\boldsymbol{\theta}}$, we have

$$\max \left\{ \left\| \nabla \mathcal{L}_{H_1}\left(\hat{\boldsymbol{\theta}}\right) \right\|^2, \ \left\| \nabla \mathcal{L}_{H_2}\left(\hat{\boldsymbol{\theta}}\right) \right\|^2 \right\} \leq \varepsilon .$$

Therefore,

$$\varepsilon \geq \frac{1}{2} \left(\left\| \nabla \mathcal{L}_{H_1}\left(\hat{\boldsymbol{\theta}}\right) \right\|^2 + \left\| \nabla \mathcal{L}_{H_2}\left(\hat{\boldsymbol{\theta}}\right) \right\|^2 \right) . \tag{5.11}$$

Substituting the respective values of $\nabla \mathcal{L}_{H_1}(\boldsymbol{\theta})$ and $\nabla \mathcal{L}_{H_2}(\boldsymbol{\theta})$ in the above, we obtain that

$$\varepsilon \geq \frac{L^2}{2} \left(\left\| \hat{\boldsymbol{\theta}} \right\|^2 + \left\| \hat{\boldsymbol{\theta}} - \frac{f}{n-f}\mathbf{z} \right\|^2 \right) \geq \frac{L^2}{4} \left\| \frac{f}{n-f}\mathbf{z} \right\|^2 = \frac{f}{4(n-2f)} \zeta .$$

The second inequality in the above follows from the triangle and the Jensen's inequalities, applied in this order. Since $n - 2f \leq n$, from above we obtain that $\varepsilon \geq \frac{f}{4n} \zeta$, which concludes the proof. □

According to Theorem 5.3, there exists a federated machine learning problem (in the setting we consider) and a set of $n - f$ honest nodes H for which a given distributed algorithm Π incurs an error of at least $\frac{f}{4n}\zeta$. More precisely, if $\hat{\boldsymbol{\theta}}$ denotes the model that Π outputs, then $\left\|\nabla \mathcal{L}_H\left(\hat{\boldsymbol{\theta}}\right)\right\|^2 \geq \varepsilon$, where $\varepsilon \in \Omega_\star\left(\frac{f}{n}\zeta\right)$. As expected, the error is proportional to both the fraction of adversarial nodes and the gradient dissimilarity. The (order of the) lower bound applies directly to the strongly convex case. We can verify this by recalling the fact that when the loss function $\mathcal{L}_H$ is μ-strongly convex (i.e., satisfies Assumption 4.2) and L-smooth (due to Assumption 3.1), for all $\boldsymbol{\theta}$,

$$\|\nabla \mathcal{L}_H(\boldsymbol{\theta})\|^2 \leq \frac{L^2}{\mu}\left(\mathcal{L}_H(\boldsymbol{\theta}) - \mathcal{L}_H^*\right).$$

Hence, if $\|\nabla \mathcal{L}_H(\boldsymbol{\theta})\|^2 \geq \varepsilon$, then $\mathcal{L}_H(\boldsymbol{\theta}) - \mathcal{L}_H^* \geq \frac{\mu}{L^2}\varepsilon$. We can also derive a more precise lower bound (with respect to the constants μ and L) in the strongly convex case by simply adapting the proof of Theorem 5.3.

Impact of Local Gradient Noise
Note that in the counterexample used to derive the lower bound in Theorem 5.3, we did not consider any stochasticity, i.e., each node could compute the true gradient of its local loss function. It is natural to ponder whether the lower bound is generally higher due to the gradient noise usually present in SGD (or mini-batch GD)-based machine learning algorithms. This lower bound matches the upper bound we derived in Sect. 5.3.1. Note nevertheless that matching the upper bound with robust DMGD demands a large batch-size, specifically $b \in \Omega_\star\left(\frac{\sigma^2}{\zeta}\right)$. This inflates the gradient complexity of the algorithm, to the point of making it impractical. In Chap. 6, we analyze a variant of the robust DMGD algorithm that incorporates *gradient momentum* at the (honest) nodes' end, and whose asymptotic (robust) training error is actually independent of the local gradient noise. This demonstrates that we can indeed obtain an order-optimal training error under adversarial nodes, using an algorithm with a reasonable gradient complexity.

5.4 Chapter Notes

We have presented in this chapter a couple of techniques for pre-aggregation processing of input vectors to tighten the robustness of a large class of aggregation rules. The pre-aggregation scheme of *nearest-neighbor mixing (NNM)* was originally proposed in [1]. In NNM, each input vector is mapped to the average of a cluster comprising $n - f$ vectors: the input vector itself and $n - f - 1$ other vectors closest (or nearest) to it. When the vectors obtained after applying NNM are fed to an (f, κ)-robust averaging rule where $\kappa \in O_n(1)$, the output satisfies (f, κ')-robustness

where $\kappa' \in O_n\left(\frac{f}{n}\right)$. It is important to note that this linear dependence of κ' on the fraction of adversarial workers $\frac{f}{n}$ is order optimal and cannot be improved upon, in general. However, NNM is computationally taxing; specifically, it imposes an overhead that is quadratic in n, the number of input vectors. A simpler alternative to NNM is *Bucketing*, proposed in [8] and then further analyzed in [6].[4] This method only imposes an overhead that is linear in n. The shortcoming of bucketing is its stochastic nature, as it yields order-optimal robustness only in a probabilistic sense (in expectation over the randomness of the shuffling done for creating *buckets*). The proofs for NNM are borrowed from our work on robust machine learning [1].

We have then demonstrated why optimizing κ, specifically achieving $\kappa \in O_n\left(\frac{f}{n}\right)$, is essential to obtain a tight training error in the presence of adversarial workers, when measuring data heterogeneity through the lens of ζ-gradient dissimilarity (introduced in the previous chapter as Assumption 4.1). Although the general limitation on training error, in the case of robustness, under heterogeneity was first demonstrated in the context of distributed optimization [7, 12], the notion of ζ-gradient dissimilarity was later introduced and studied concurrently in [5] and [8]. In fact, the main idea for the lower bound that we presented in Theorem 5.3 comes from a lower bound proof initially presented in [8].

Beyond ζ-Gradient Dissimilarity

While insightful, the heterogeneity model of ζ-gradient dissimilarity is restrictive in practice. Indeed, many machine learning tasks fail to satisfy this notion of heterogeneity. The more general alternative is the model of (G, B)-*gradient dissimilarity*, which has been widely studied in non-robust regime [9–11, 13, 14]. In the context of robust machine learning, recent results in papers [2, 6, 8] have used the definition of (G, B)-*gradient dissimilarity* to demonstrate that heterogeneity can further deteriorate robustness in general. Specifically, it has been proven, in [2], that data heterogeneity brings down the breakdown point, i.e., federated machine learning systems are generally more vulnerable to an adversary that was previously believed due to heterogeneous local data-generating distributions. Nevertheless, for pedagogical reasons, we only provide robustness analysis under ζ-gradient dissimilarity. Moreover, analyzing robustness in this rather simple heterogeneity condition provides key insights that help us improve the upper bounds (on training error) for robust machine learning algorithms in general. Indeed, the original motivation for the pre-aggregation schemes that we have presented in this chapter was through ζ-gradient dissimilarity.

[4] Note that results similar to Theorem 5.2 can be found in [6].

References

1. Allouah Y, Farhadkhani S, Guerraoui R, Gupta N, Pinot R, Stephan J (2023) Fixing by mixing: a recipe for optimal byzantine ML under heterogeneity. In: Proceedings of the 26th International Conference on Artificial Intelligence and Statistics, vol 206. Proceedings of Machine Learning Research. PMLR, pp 1232–1300
2. Allouah Y, Guerraoui R, Gupta N, Rafaël P, Rizk G (2023) Robust distributed learning: tight error bounds and breakdown point under data heterogeneity. In: The 37th Conference on Neural Information Processing Systems
3. Charikar M, Steinhardt J, Valiant G (2017) Learning from untrusted data. In: Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, pp 47–60
4. Diakonikolas I, Kamath G, Kane D, Li J, Steinhardt J, Stewart A (2019) Sever: A robust meta-algorithm for stochastic optimization. In: International Conference on Machine Learning. PMLR, pp 1596–1606
5. El Mhamdi EM, Farhadkhani S, Guerraoui R, Guirguis A, Hoang LN, Rouault S (2021) Collaborative learning in the jungle (decentralized, byzantine, heterogeneous, asynchronous and nonconvex learning) In: The 35th Conference on Neural Information Processing Systems
6. Gorbunov E, Horváth S, Richtárik P, Gidel G (2023) Variance reduction is an antidote to byzantines: better rates, weaker assumptions and communication compression as a cherry on the top. In: The 11th International Conference on Learning Representations
7. Gupta N, Vaidya NH (2019) Byzantine fault-tolerant parallelized stochastic gradient descent for linear regression. In: 2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton). IEEE, pp 415–420
8. Karimireddy SP, He L, Jaggi M (2022) Byzantine-robust learning on heterogeneous datasets via bucketing. In: International Conference on Learning Representations
9. Karimireddy SP, Kale S, Mohri M, Reddi S, Stich S, Suresh AT (2020) Scaffold: Stochastic controlled averaging for federated learning. In: International Conference on Machine Learning. PMLR, pp 5132–5143
10. Khaled A, Richtárik P (2023) Better theory for SGD in the nonconvex world. Trans Mach Learn Res. Survey certification. ISSN: 2835–8856
11. Koloskova A, Loizou N, Boreiri S, Jaggi M, Stich SU (2020) A unified theory of decentralized SGD with changing topology and local updates. REFERENCES 109. In: Proceedings of the 37th International Conference on Machine Learning, ICML, 13–18 July 2020, Virtual Event, vol 119. Proceedings of Machine Learning Research. PMLR, pp 5381–5393
12. Liu S, Gupta N, Vaidya NH (2021) Approximate byzantine fault-tolerance in distributed optimization. In: Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing, PODC'21. Virtual event. Association for Computing Machinery, Italy, pp 379–389. ISBN: 9781450385480
13. Mitra A, Jaafar R, Pappas GJ, Hassani H (2021) Linear convergence in federated learning: Tackling client heterogeneity and sparse gradients. Adv Neural Inf Process Syst 34:14606–14619
14. Noble M, Bellet A, Dieuleveut A (2022) Differentially private federated learning on heterogeneous data. In: International Conference on Artificial Intelligence and Statistics. PMLR, pp 10110–10145
15. Prasad A, Suggala AS, Balakrishnan S, Ravikumar P (2020) Robust estimation via robust gradient estimation. J R Stat Soc B (Stat Methodol) 82(3):601–627

Chapter 6
Practical Robustness

Abstract In Chaps. 4 and 5, we introduced a basic robustification of the distributed mini-batch gradient-descent (DMGD) method by essentially replacing the averaging operation at the server with a robust aggregation rule, and then with pre-aggregation schemes. These modifications to DMGD yield an order-optimal asymptotic training error, when using sufficiently large batch-sizes, despite the presence of some adversarial nodes in the system. The resulting Robust DMGD method, however, might be impractical in some cases, as large batch-sizes inflate significantly the gradient complexity. In this chapter, we introduce an advanced technique for reducing the *gradient complexity* of Robust DMGD, while preserving the order-optimal nature of the asymptotic training error. Thereby, yielding a practical solution to robustness. The technique consists in using *Polyak's momentum* on the local mini-batch gradients in order to diminish the effect of local gradient noise in the training error of the learning algorithm. This provides an order-optimal robustness against adversarial nodes with a reasonable gradient complexity.

6.1 Robust DMGD with Local Momentum

In this section, first we describe the Robust DMGD method with *local gradient momentum*, and we present preliminary analyses on the influence of momentum on the aggregation error (under adversarial nodes) and the bias introduced by the use of momentum. Embedding momentum at local level consists in having nodes send Polyak's momentum of their mini-batch gradients to the server. In turn, the server updates the models using a robust aggregation of the local momentums. Analogous to Robust DMGD, adversarial nodes can send arbitrary vectors for their momentums. The resulting Robust DMGD algorithm with local Polyak's momentum is described in Algorithm 6. As Polyak's gradient momentum is also referred to as the *heavy-ball* method, hereafter we also refer to Algorithm 6 as the robust *distributed mini-batch heavy-ball* (or *Robust DMHB*) method.

Similarly to the analysis of Robust DMGD presented in Sect. 4.4, to analyze the convergence of Robust DMHB, we decompose the update rule in (6.3) into two parts: (i) the conventional DMGD algorithm with momentum, i.e., DMHB method,

R. Guerraoui et al., *Robust Machine Learning*, Machine Learning: Foundations, Methodologies, and Applications, https://doi.org/10.1007/978-981-97-0688-4_6

Algorithm 6 Robust DMHB

Initialization: The **server** chooses an arbitrary initial model $\boldsymbol{\theta_1} \in \mathbb{R}^d$ (similar to the original DMGD method presented in Algorithm 4). Each **honest** node i chooses an initial momentum vector $\mathbf{m_0^{(i)}} \coloneqq \mathbf{0} \in \mathbb{R}^d$. In each iteration $t \in \{1, 2, \ldots\}$, the server maintains a model parameter $\boldsymbol{\theta_t}$ that is broadcast to all the nodes. The rest of the iteration is as follows:

1. **Local phase.** Each **honest** node $i \in [n]$ independently and randomly samples a set of b data points $S_t^{(i)}$ from the local training sample $\mathcal{S}_i$ (without replacement) and computes the gradient

$$\mathbf{g_t^{(i)}} \coloneqq \frac{1}{b} \sum_{(\mathbf{x},y)\in S_t^{(i)}} \nabla_{\boldsymbol{\theta}} \ell\big(h_{\boldsymbol{\theta_t}}(\mathbf{x}), y\big)\,.$$

   Then, the node updates its previous momentum vector $\mathbf{m_{t-1}^{(i)}}$ to

$$\mathbf{m_t^{(i)}} \coloneqq \beta_t\, \mathbf{m_{t-1}^{(i)}} + (1-\beta_t)\, \mathbf{g_t^{(i)}}\,, \tag{6.1}$$

   where β_t is the *momentum coefficient* for iteration t.
2. **Global phase.** Each node $i \in [n]$ returns to the server the updated momentum vector $\mathbf{m_t^{(i)}}$. If node i is **honest**, then $\mathbf{m_t^{(i)}}$ is defined as per (6.1). Otherwise, $\mathbf{m_t^{(i)}}$ could be an arbitrary vector in $\mathbb{R}^d$. Upon receiving the momentum vectors from all the nodes, the server computes their aggregate

$$\widehat{\mathbf{m}}_\mathbf{t} \coloneqq F\left(\mathbf{m_t^{(1)}}, \ldots, \mathbf{m_t^{(n)}}\right)\,. \tag{6.2}$$

   Using the aggregate momentum vector $\widehat{\mathbf{m}}_\mathbf{t}$, the server updates the current model $\boldsymbol{\theta_t}$ to

$$\boldsymbol{\theta}_{\mathbf{t+1}} \coloneqq \boldsymbol{\theta_t} - \gamma_t\, \widehat{\mathbf{m}}_\mathbf{t}\,, \tag{6.3}$$

   where γ_t is the learning rate at iteration t.

and (ii) the perturbation due to the (robust) aggregation error. Specifically denoting by H a set of $n - f$ honest nodes for each iteration t, we can rewrite the update rule (6.3) as follows:

$$\boldsymbol{\theta}_{\mathbf{t+1}} = \boldsymbol{\theta_t} - \gamma_t\, \mathbf{m_t^H} - \gamma_t\, \boldsymbol{\delta_t}\,, \tag{6.4}$$

where $\mathbf{m_t^H} \coloneqq \frac{1}{|H|} \sum_{i\in H} \mathbf{m_t^{(i)}}$, and

$$\boldsymbol{\delta_t} \coloneqq \widehat{\mathbf{m}}_\mathbf{t} - \mathbf{m_t^H} = F\left(\mathbf{m_t^{(1)}}, \ldots, \mathbf{m_t^{(n)}}\right) - \mathbf{m_t^H}\,. \tag{6.5}$$

According to (6.4), we can treat Algorithm 6 as a perturbation of the DMHB method being executed by the set of honest nodes H. Below, in Sects. 6.1.1 and 6.1.2, we analyze, respectively, the effects of the momentum coefficient β on the aggregation error over the momentum vectors, i.e., $\boldsymbol{\delta_t}$, and the *deviation* of the average momentum from the true gradient of the (honest) average loss function,

i.e., $\mathbf{m}_\mathbf{t}^\mathbf{H} - \nabla \mathcal{L}_H(\boldsymbol{\theta}_\mathbf{t})$. The former enables us to exclusively study the effect of adversarial nodes on the convergence of Algorithm 6, and the latter plays a crucial role in demonstrating the convergence of the unperturbed DMHB method, i.e., the convergence of (6.4) when $\boldsymbol{\delta}_\mathbf{t} \equiv \mathbf{0}$. In Sect. 6.2, upon combining these two analyses, we obtain the final convergence result for Algorithm 6.

Remark 6.1 Recall that, in Sect. 4.2, the failure model we consider assumes that the identity of adversarial nodes remains fixed from one iteration to another. Therefore, for robustness parameter f (i.e., the maximum number of adversarial nodes we indent to tolerate), we have (at least) $n - f$ nodes behaving honestly for the entire duration of the algorithm. It is precisely the persistent correctness of $n - f$ nodes that is exploited in the method of distributed momentum for limiting the adversarial nodes from taking advantage of the uncertainty in gradient computations, whereas, because Robust DMGD employs *instantaneous* robust averaging of the nodes' gradient (blatantly disregarding their "history"), the static identity of adversarial nodes during the entire learning procedure does not confer any benefits to Robust DMGD.

6.1.1 Robust Aggregation Error

We see in this section that the effect of local gradient noise on the aggregation error $\boldsymbol{\delta}_\mathbf{t}$ can be diminished by choosing a large enough momentum coefficient β. We first analyze the separation (a.k.a. *drift*) between the local momentums of the honest nodes. For this particular analysis, we do not require the aggregation rule to be a robust averaging. Recall that we let H denote a set of $n - f$ honest nodes. Also recall, from Sect. 3.2 in Chap. 3, that $\mathbb{E}_t[\cdot]$ denotes the conditional expectation $\mathbb{E}[\cdot \,|\, \mathcal{P}_t]$. However, in the context of Robust DMHB, we redefine the history $\mathcal{P}_t$ to be

$$\mathcal{P}_t \coloneqq \left\{ (\boldsymbol{\theta}_\mathbf{1}, \ldots, \boldsymbol{\theta}_\mathbf{t}) \ , \ \left(\mathbf{m}_\mathbf{1}^\mathbf{(i)}, \ldots, \mathbf{m}_\mathbf{t-1}^\mathbf{(i)} \right)_{i \in [n]} \right\} ,$$

and, by convention, $\mathcal{P}_1 = \{\boldsymbol{\theta}_\mathbf{1}\}$. The reason we include the local momentum vectors, and not the local gradients, in $\mathcal{P}_t$ is because in any iteration t the output of the aggregation $\widehat{\mathbf{m}}_\mathbf{t} \coloneqq F\left(\mathbf{m}_\mathbf{t}^\mathbf{(1)}, \ldots, \mathbf{m}_\mathbf{t}^\mathbf{(n)} \right)$ is a function of the momentum vectors received from all the nodes, including the adversarial ones. While it is true that for an honest node i, the previous momentum vector $\mathbf{m}_\mathbf{t-1}^\mathbf{(i)}$ is a deterministic function of the past gradients $\mathbf{g}_\mathbf{1}^\mathbf{(i)}, \ldots, \mathbf{g}_\mathbf{t-1}^\mathbf{(i)}$, the same need not be true for $\mathbf{m}_\mathbf{t}^\mathbf{(j)}$ when node j is adversarial. Notation $\mathbb{E}[\cdot]$ still represents the total expectation over the randomness in Algorithm 6. Specifically, if we execute T iterations of Algorithm 6, then $\mathbb{E}[\cdot] \coloneqq \mathbb{E}_1[\cdots \mathbb{E}_T[\cdot]]$.

Lemma 6.1 *Suppose Assumptions 3.2 and 4.1. Consider Algorithm 6 with a constant momentum coefficient, i.e., $\beta_t = \beta$ for each iteration t, and batch-size $b \in [1, m)$. Then, for all t, the following holds true:*

$$\mathbb{E}\left[\frac{1}{|H|}\sum_{i\in H}\left\|\mathbf{m_t^{(i)}} - \mathbf{m_t^H}\right\|^2\right] \leq 3\left(\frac{1-\beta}{1+\beta}\right)\left(1+\frac{1}{n-f}\right)\frac{\sigma^2}{b} + 3\zeta\,,$$

where recall that $\mathbf{m_t^H} \coloneqq \frac{1}{|H|}\sum_{i\in H}\mathbf{m_t^{(i)}}$.

Proof Consider an arbitrary node $i \in H$ and an arbitrary iteration t of Algorithm 6. From (6.1), when $\beta_t = \beta$ for all t, we obtain that

$$\mathbf{m_t^{(i)}} = (1-\beta)\sum_{k=1}^{t}\beta^{t-k}\mathbf{g_k^{(i)}}\,. \tag{6.6}$$

Let $\mathbf{g_t^H} \coloneqq \frac{1}{|H|}\sum_{i\in H}\mathbf{g_t^{(i)}}$. Recall that $\mathbf{m_t^H} \coloneqq \frac{1}{|H|}\sum_{i\in H}\mathbf{m_t^{(i)}}$. Therefore, from (6.6), we obtain that

$$\frac{1}{|H|}\sum_{i\in H}\left\|\mathbf{m_t^{(i)}} - \mathbf{m_t^H}\right\|^2 = \frac{(1-\beta)^2}{|H|}\sum_{i\in H}\left\|\sum_{k=1}^{t}\beta^{t-k}\left(\mathbf{g_k^{(i)}} - \mathbf{g_t^H}\right)\right\|^2\,. \tag{6.7}$$

Recall that $\nabla\mathcal{L}_H(\boldsymbol{\theta}) \coloneqq \frac{1}{|H|}\sum_{i\in H}\nabla\mathcal{L}_i(\boldsymbol{\theta})$. Using the triangle inequality, we obtain that

$$\left\|\sum_{k=1}^{t}\beta^{t-k}\left(\mathbf{g_k^{(i)}} - \mathbf{g_t^H}\right)\right\| \leq \left\|\sum_{k=1}^{t}\beta^{t-k}\left(\mathbf{g_k^{(i)}} - \nabla\mathcal{L}_i(\boldsymbol{\theta_t})\right)\right\|$$
$$+\left\|\sum_{k=1}^{t}\beta^{t-k}\left(\mathbf{g_t^H} - \nabla\mathcal{L}_H(\boldsymbol{\theta_t})\right)\right\| + \left\|\sum_{k=1}^{t}\beta^{t-k}\left(\nabla\mathcal{L}_i(\boldsymbol{\theta_t}) - \nabla\mathcal{L}_H(\boldsymbol{\theta_t})\right)\right\|\,.$$

(continued)

By Jensen's inequality, as $(\cdot)^2$ is convex, for real values a_1, a_2, a_3, we have $(a_1 + a_2 + a_3)^2 \leq 3(a_1)^2 + 3(a_2)^2 + 3(a_3)^2$. Therefore,

$$\begin{aligned}\left\|\sum_{k=1}^{t}\beta^{t-k}\left(\mathbf{g_k^{(i)}} - \mathbf{g_t^H}\right)\right\|^2 \leq\ & 3\left\|\sum_{k=1}^{t}\beta^{t-k}\left(\mathbf{g_k^{(i)}} - \nabla\mathcal{L}_i\left(\boldsymbol{\theta}_\mathbf{t}\right)\right)\right\|^2 \\ & + 3\left\|\sum_{k=1}^{t}\beta^{t-k}\left(\mathbf{g_t^H} - \nabla\mathcal{L}_H\left(\boldsymbol{\theta}_\mathbf{t}\right)\right)\right\|^2 \\ & + 3\left\|\sum_{k=1}^{t}\beta^{t-k}\left(\nabla\mathcal{L}_i\left(\boldsymbol{\theta}_\mathbf{t}\right) - \nabla\mathcal{L}_H\left(\boldsymbol{\theta}_\mathbf{t}\right)\right)\right\|^2.\end{aligned}$$

Let us now introduce the following notations.[a]

$$\begin{aligned}A_t^{(i)} &:= \left\|\sum_{k=1}^{t}\beta^{t-k}\left(\mathbf{g_k^{(i)}} - \nabla\mathcal{L}_i\left(\boldsymbol{\theta}_\mathbf{t}\right)\right)\right\|^2, \\ B_t^{(i)} &:= \left\|\sum_{k=1}^{t}\beta^{t-k}\left(\mathbf{g_t^H} - \nabla\mathcal{L}_H\left(\boldsymbol{\theta}_\mathbf{t}\right)\right)\right\|^2 \\ \text{and } C_t^{(i)} &:= \left\|\sum_{k=1}^{t}\beta^{t-k}\left(\nabla\mathcal{L}_i\left(\boldsymbol{\theta}_\mathbf{t}\right) - \nabla\mathcal{L}_H\left(\boldsymbol{\theta}_\mathbf{t}\right)\right)\right\|^2. \end{aligned} \tag{6.8}$$

Thus, we can rewrite the above inequality as

$$\left\|\sum_{k=1}^{t}\beta^{t-k}\left(\mathbf{g_k^{(i)}} - \mathbf{g_t^H}\right)\right\|^2 \leq 3A_t^{(i)} + 3B_t^{(i)} + 3C_t^{(i)}.$$

As i in the above is assumed arbitrary from H, by simply substituting from the above in (6.7), we have

$$\frac{1}{|H|}\sum_{i\in H}\left\|\mathbf{m_t^{(i)}} - \mathbf{m_t^H}\right\|^2 \leq 3(1-\beta)^2\left(A_t^H + 3B_t^H + 3C_t^H\right), \tag{6.9}$$

where we use the notation $\{\cdot\}_t^H := \frac{1}{|H|}\sum_{i\in H}\{\cdot\}_t^{(i)}$. Next, we obtain upper bounds for A_t^H, B_t^H, and C_t^H under Assumptions 3.2 and 4.1.

(continued)

Bound on A_t^H. Let i be an arbitrary node in H and t be an arbitrary integer such that $t > 1$. We can write

$$A_t^{(i)} = \left\| \sum_{k=1}^{t-1} \beta^{t-k} \left(\mathbf{g_k^{(i)}} - \nabla \mathcal{L}_i(\boldsymbol{\theta_k}) \right) + \left(\mathbf{g_t^{(i)}} - \nabla \mathcal{L}_i(\boldsymbol{\theta_t}) \right) \right\|^2 .$$

Recall that for any vector $\mathbf{v} \in \mathbb{R}^d$, $\|\mathbf{v}\|^2 = \langle \mathbf{v}, \mathbf{v} \rangle$. Hence, by definition of the norm, we obtain that

$$\begin{aligned} A_t^{(i)} = &\left\| \sum_{k=1}^{t-1} \beta^{t-k} \left(\mathbf{g_k^{(i)}} - \nabla \mathcal{L}_i(\boldsymbol{\theta_k}) \right) \right\|^2 + \left\| \mathbf{g_t^{(i)}} - \nabla \mathcal{L}_i(\boldsymbol{\theta_t}) \right\|^2 \\ &+ 2 \left\langle \sum_{k=1}^{t-1} \beta^{t-k} \left(\mathbf{g_k^{(i)}} - \nabla \mathcal{L}_i(\boldsymbol{\theta_k}) \right), \mathbf{g_t^{(i)}} - \nabla \mathcal{L}_i(\boldsymbol{\theta_t}) \right\rangle . \end{aligned}$$

By definition of $A_t^{(i)}$ in (6.8), $\left\| \sum_{k=1}^{t-1} \beta^{t-k} \left(\mathbf{g_k^{(i)}} - \nabla \mathcal{L}_i(\boldsymbol{\theta_k}) \right) \right\|^2 = \beta^2 A_{t-1}^{(i)}$. Using this in the above, we obtain that

$$\begin{aligned} A_t^{(i)} = &\beta^2 A_{t-1}^{(i)} + \left\| \mathbf{g_t^{(i)}} - \nabla \mathcal{L}_i(\boldsymbol{\theta_t}) \right\|^2 \\ &+ 2 \left\langle \sum_{k=1}^{t-1} \beta^{t-k} \left(\mathbf{g_k^{(i)}} - \nabla \mathcal{L}_i(\boldsymbol{\theta_k}) \right), \mathbf{g_t^{(i)}} - \nabla \mathcal{L}_i(\boldsymbol{\theta_t}) \right\rangle . \end{aligned}$$

Applying the conditional expectation $\mathbb{E}_t[\cdot]$ on both sides, we obtain that

$$\begin{aligned} \mathbb{E}_t \left[A_t^{(i)} \right] = &\beta^2 A_{t-1}^{(i)} + \mathbb{E}_t \left[\left\| \mathbf{g_t^{(i)}} - \nabla \mathcal{L}_i(\boldsymbol{\theta_t}) \right\|^2 \right] \\ &+ 2 \left\langle \sum_{k=1}^{t-1} \beta^{t-k} \left(\mathbf{g_k^{(i)}} - \nabla \mathcal{L}_i(\boldsymbol{\theta_k}) \right), \mathbb{E}_t \left[\mathbf{g_t^{(i)}} - \nabla \mathcal{L}_i(\boldsymbol{\theta_t}) \right] \right\rangle . \end{aligned} \tag{6.10}$$

Note that as $\mathbb{E}_t \left[\mathbf{g_t^{(i)}} \right] = \nabla \mathcal{L}_i(\boldsymbol{\theta_t})$,

$$\left\langle \sum_{k=1}^{t-1} \beta^{t-k} \left(\mathbf{g_k^{(i)}} - \nabla \mathcal{L}_i(\boldsymbol{\theta_k}) \right), \mathbb{E}_t \left[\mathbf{g_t^{(i)}} - \nabla \mathcal{L}_i(\boldsymbol{\theta_t}) \right] \right\rangle = 0 .$$

(continued)

Using the above in (6.10), we obtain that

$$\mathbb{E}_t\left[A_t^{(i)}\right] = \beta^2 A_{t-1}^{(i)} + \mathbb{E}_t\left[\left\|\mathbf{g_t^{(i)}} - \nabla\mathcal{L}_i(\boldsymbol{\theta_t})\right\|^2\right].$$

Now, similarly to the proof of Theorem 3.1, by definition of $\mathbf{g_t^{(i)}}$ and using Assumption 3.2, we get $\mathbb{E}_t\left[\left\|\mathbf{g_t^{(i)}} - \nabla\mathcal{L}_i(\boldsymbol{\theta_t})\right\|^2\right] \leq \frac{\sigma^2}{b}$. Thus, from the above, we obtain that

$$\mathbb{E}_t\left[A_t^{(i)}\right] \leq \beta^2 A_{t-1}^{(i)} + \frac{\sigma^2}{b}.$$

Applying the total expectation $\mathbb{E}[\cdot]$ on both sides, and recalling t in the above is an arbitrary integer greater than 1, we obtain the recursion:

$$\mathbb{E}\left[A_t^{(i)}\right] \leq \beta^2\,\mathbb{E}\left[A_{t-1}^{(i)}\right] + \frac{\sigma^2}{b}, \quad \forall t \geq 2.$$

Therefore, by telescopic expansion of the above, we obtain that

$$\mathbb{E}\left[A_t^{(i)}\right] \leq \beta^{2(t-1)}\,\mathbb{E}\left[A_1^{(i)}\right] + \frac{\sigma^2}{b}\sum_{k=0}^{t-2}\beta^{2k}, \quad \forall t \geq 2. \tag{6.11}$$

By definition of $A_t^{(i)}$ in (6.8), and Assumption 3.2, we have

$$\mathbb{E}\left[A_1^{(i)}\right] = \mathbb{E}\left[\left\|\mathbf{g_1^{(i)}} - \nabla\mathcal{L}_i(\boldsymbol{\theta_1})\right\|^2\right] \leq \frac{\sigma^2}{b}.$$

Substituting from the above in (6.11), and recalling that $\beta < 1$, we obtain that

$$\mathbb{E}\left[A_t^{(i)}\right] \leq \frac{\sigma^2}{b}\sum_{k=0}^{t-1}\beta^{2k} \leq \frac{\sigma^2}{b}\frac{1}{1-\beta^2}, \quad \forall t.$$

As the above holds true for any $i \in H$, we finally obtain that

$$\mathbb{E}\left[A_t^{H}\right] = \frac{1}{|H|}\sum_{i\in H}\mathbb{E}\left[A_t^{(i)}\right] \leq \frac{\sigma^2}{b}\frac{1}{1-\beta^2}, \quad \forall t. \tag{6.12}$$

(continued)

Bound on B_t^H. As the mini-batched gradients $\{\mathbf{g_t^{(i)}},\ i \in H\}$ are mutually independent. Hence similarly to 3.13 in the proof of Theorem 3.1, we have

$$\mathbb{E}\left[\left\|\mathbf{g_t^H} - \nabla\mathcal{L}_H(\boldsymbol{\theta_t})\right\|^2\right] = \frac{1}{(n-f)^2}\sum_{i\in H}\mathbb{E}\left[\left\|\mathbf{g_t^{(i)}} - \nabla\mathcal{L}_i(\boldsymbol{\theta_t})\right\|^2\right] \leq \frac{\sigma^2}{b(n-f)}\,.$$

We can then follow the same steps as for analyzing A_t^H, to obtain

$$\mathbb{E}\left[B_t^H\right] = \frac{1}{|H|}\sum_{i\in H}\mathbb{E}\left[B_t^{(i)}\right] \leq \frac{\sigma^2}{b(n-f)}\frac{1}{(1-\beta^2)}, \text{ for all iteration } t \geq 2. \tag{6.13}$$

Bound on C_t^H. Recall from (6.8) that, for all $i \in H$ and each iteration t,

$$C_t^{(i)} = \left\|\sum_{k=1}^{t}\beta^{t-k}\left(\nabla\mathcal{L}_i(\boldsymbol{\theta_t}) - \nabla\mathcal{L}_H(\boldsymbol{\theta_t})\right)\right\|^2.$$

Applying triangle and Jensen's inequalities (in that order), we obtain that

$$C_t^{(i)} \leq \left(\sum_{k=1}^{t}\beta^{t-k}\right)\sum_{k=1}^{t}\beta^{t-k}\left\|\nabla\mathcal{L}_i(\boldsymbol{\theta_t}) - \nabla\mathcal{L}_H(\boldsymbol{\theta_t})\right\|^2.$$

Therefore, taking the average on both sides over all $i \in H$, we obtain that

$$C_t^H := \frac{1}{|H|}\sum_{i\in H}C_t^{(i)} \leq \left(\sum_{k=1}^{t}\beta^{t-k}\right)\frac{1}{|H|}\sum_{i\in H}\sum_{k=1}^{t}\beta^{t-k} \times \left\|\nabla\mathcal{L}_i(\boldsymbol{\theta_t}) - \nabla\mathcal{L}_H(\boldsymbol{\theta_t})\right\|^2.$$

(continued)

Thus, as $|H| = n - f$, due to Assumption 4.1, we have, for all t,

$$\begin{aligned} C_t^H &\leq \left(\sum_{k=1}^{t} \beta^{t-k}\right) \sum_{k=1}^{t} \beta^{t-k} \frac{1}{|H|} \sum_{i \in H} \|\nabla \mathcal{L}_i(\boldsymbol{\theta}_\mathbf{t}) - \nabla \mathcal{L}_H(\boldsymbol{\theta}_\mathbf{t}\|^2 \\ &\leq \left(\sum_{k=1}^{t} \beta^{t-k}\right)^2 \zeta \\ &\leq \frac{\zeta}{(1-\beta)^2}. \end{aligned} \tag{6.14}$$

The last inequality in the above follows from the fact that $\beta < 1$.

Concluding Step. Applying total expectation $\mathbb{E}[\cdot]$ on both sides in (6.9), we have

$$\mathbb{E}\left[\frac{1}{|H|} \sum_{i \in H} \left\|\mathbf{m}_\mathbf{t}^{(\mathbf{i})} - \mathbf{m}_\mathbf{t}^{\mathbf{H}}\right\|^2\right] \leq 3(1-\beta)^2 \times \left(\mathbb{E}\left[A_t^H\right] + \mathbb{E}\left[B_t^H\right] + \mathbb{E}\left[C_t^H\right]\right).$$

Substituting from (6.12), (6.13), and (6.14) in the above, we obtain that

$$\mathbb{E}\left[\frac{1}{|H|} \sum_{i \in H} \left\|\mathbf{m}_\mathbf{t}^{(\mathbf{i})} - \mathbf{m}_\mathbf{t}^{\mathbf{H}}\right\|^2\right] \leq 3\left(\frac{1+\beta}{1-\beta}\right)\left(1 + \frac{1}{n-f}\right)\frac{\sigma^2}{b} + 3\zeta.$$

This concludes the proof. □

[a]Note that $B_t^{(i)}$ is independent on i, but we use this notation for consistency.

In the following, we combine the result of Lemma 6.1 with the property of robust averaging (stated in Definition 4.3) to obtain an upper bound on the aggregation error $\boldsymbol{\delta}_\mathbf{t}$ (defined with respect to the momentum vectors in (6.5)). For the analysis, we assume that the aggregation rule F in Algorithm 6 is an (f, κ)-robust averaging.

Lemma 6.2 *Suppose Assumptions 3.2 and 4.1. Consider Algorithm 6 with an (f, κ)-robust averaging rule F, a batch-size $b \in [1, m)$, and a constant momentum*

coefficient, i.e., $\beta_t = \beta$ for each iteration t. Then, for all t, we have

$$\mathbb{E}\left[\|\boldsymbol{\delta_t}\|^2\right] \leq 6\kappa\left(\frac{1-\beta}{1+\beta}\right)\frac{\sigma^2}{b} + 3\kappa\zeta\,,$$

where $\mathbf{m_t^H} \coloneqq \frac{1}{|H|}\sum_{i\in H}\mathbf{m_t^{(i)}}$.

Proof Let aggregation F in Algorithm 6 be an arbitrary (f,κ)-robust averaging rule. Consider an arbitrary iteration t. Recall, from (6.5), that

$$\boldsymbol{\delta_t} = \widehat{\mathbf{m}}_\mathbf{t} - \mathbf{m_t^H}\,,$$

where recall from (6.2) that $\widehat{\mathbf{m}}_\mathbf{t} \coloneqq F\left(\boldsymbol{m_t^{(1)}}, \ldots, \mathbf{m_t^{(n)}}\right)$. Since F is (f,κ)-robust averaging, by Definition 4.3, we obtain that

$$\|\boldsymbol{\delta_t}\|^2 = \left\|\widehat{\mathbf{m}}_\mathbf{t} - \mathbf{m_t^H}\right\|^2 \leq \frac{\kappa}{n-f}\sum_{i\in H}\left\|\mathbf{m_t^{(i)}} - \mathbf{m_t^H}\right\|^2. \tag{6.15}$$

Applying total expectation $\mathbb{E}[\cdot]$ on both sides, we obtain that

$$\mathbb{E}\left[\|\boldsymbol{\delta_t}\|^2\right] \leq \frac{\kappa}{n-f}\sum_{i\in H}\mathbb{E}\left[\left\|\mathbf{m_t^{(i)}} - \mathbf{m_t^H}\right\|^2\right]. \tag{6.16}$$

From Lemma 6.1, under the stated conditions, we have (as $|H| = n-f \geq 1$)

$$\begin{aligned}\frac{1}{n-f}\sum_{i\in H}\mathbb{E}\left[\left\|\mathbf{m_t^{(i)}} - \mathbf{m_t^H}\right\|^2\right] &\leq 3\left(\frac{1-\beta}{1+\beta}\right)\left(1+\frac{1}{n-f}\right)\frac{\sigma^2}{b} + 3\zeta \\ &\leq 6\left(\frac{1-\beta}{1+\beta}\right)\frac{\sigma^2}{b} + 3\zeta\,.\end{aligned}$$

Substituting from the above in (6.16) concludes the proof. □

Momentum Reduces the Effect of Noise

Recall from Lemma 4.1 that the bound on the aggregation error for local gradients (in Robust DMGD) is given by $4\kappa\,\frac{\sigma^2}{b} + \kappa\,\zeta$. In other words, the error grows linearly in local gradient noise σ^2. On the contrary, as shown in Lemma 6.2, the effect of σ^2

in the aggregation error for local momentums (in Robust DMHB) reduces by a factor of $\frac{1-\beta}{1+\beta}$, which decreases monotonically with the momentum coefficient β. Indeed, as the value of β approaches 1, the influence of σ^2 diminishes to 0. However, as we see in Sect. 6.1.2, a larger momentum coefficient yields a greater deviation between the honest nodes' average momentum and the average gradient. The deviation of the instantaneous gradients from their corresponding momentums (accumulating past gradients) not only affects the convergence rate of the training procedure but could also prevent the algorithm from converging to a stationary point if the momentum coefficient is not chosen carefully. In short, we cannot take β to be arbitrarily large. We design and analyze later, in Sect. 6.2, a suitable momentum coefficient that favorably controls the aforementioned trade-off between the aggregation error and the deviation. But first, let us see exactly how the momentum deviation grows with respect to the momentum coefficient.

6.1.2 Deviation Between Momentums and Gradients

In the previous section, we saw that the use of a large momentum coefficient diminishes the effect of gradient noise on the aggregation error resulting from the presence of adversarial nodes. However, in this section, upon analyzing the bias in the gradient estimation introduced by Polyak's momentum, we observe that an arbitrarily large momentum coefficient can negatively affect the convergence guarantee. In Lemma 6.3, we analyze the growth of the deviation given by

$$\boldsymbol{\xi_t} \coloneqq \mathbf{m_t^H} - \nabla \mathcal{L}_H(\boldsymbol{\theta_t}), \tag{6.17}$$

where recall that $\mathbf{m_t^H} \coloneqq \frac{1}{|H|}\sum_{i\in H} \mathbf{m_t^{(i)}}$. The result holds true for any generic aggregation rule F, which need not be robust averaging.

Lemma 6.3 *Suppose Assumptions 3.2 and 4.1. Consider Algorithm 6 with batch-size $b \in [1, m)$. Then, for any iteration $t > 1$, we have*

$$\begin{aligned}\mathbb{E}_t\left[\left\|\boldsymbol{\xi_t}\right\|^2\right] \leq\ & (1+\gamma_{t-1}L)\,(1+4\gamma_{t-1}L)\,\beta_t^2\left\|\boldsymbol{\xi_{t-1}}\right\|^2 \\ & + 2L\gamma_{t-1}(1+\gamma_{t-1}L)\beta_t^2\left\|\boldsymbol{\delta_{t-1}}\right\|^2 \\ & + 4L\gamma_{t-1}(1+\gamma_{t-1}L)\beta_t^2\left\|\nabla\mathcal{L}_H(\boldsymbol{\theta_{t-1}})\right\|^2 + (1-\beta_t)^2\frac{\sigma^2}{b(n-f)}\,.\end{aligned}$$

Proof Consider an arbitrary iteration $t > 1$. Substituting from (6.1), $\mathbf{m_t^H} = \beta_t \mathbf{m_{t-1}^H} + (1-\beta_t)\mathbf{g_t^H}$, in (6.17) we obtain that

$$\boldsymbol{\xi_t} = \beta_t \mathbf{m_{t-1}^H} + (1-\beta_t)\mathbf{g_t^H} - \nabla \mathcal{L}_H(\boldsymbol{\theta_t})\,.$$

By adding and subtracting $\beta_t \nabla \mathcal{L}_H(\boldsymbol{\theta_{t-1}})$ on the right-hand side, we obtain that

$$\begin{aligned}\boldsymbol{\xi_t} =& \beta_t \left(\mathbf{m_{t-1}^H} - \nabla \mathcal{L}_H(\boldsymbol{\theta_{t-1}})\right) + (1-\beta_t)\left(\mathbf{g_t^H} - \nabla \mathcal{L}_H(\boldsymbol{\theta_t})\right) \\ &+ \beta_t \left(\nabla \mathcal{L}_H(\boldsymbol{\theta_{t-1}}) - \nabla \mathcal{L}_H(\boldsymbol{\theta_t})\right)\,.\end{aligned}$$

Recalling that $\mathbf{m_{t-1}^H} - \nabla \mathcal{L}_H(\boldsymbol{\theta_{t-1}}) = \boldsymbol{\xi_{t-1}}$ in the above, we have

$$\boldsymbol{\xi_t} = \beta_t \boldsymbol{\xi_{t-1}} + (1-\beta_t)\left(\mathbf{g_t^H} - \nabla \mathcal{L}_H(\boldsymbol{\theta_t})\right) + \beta_t \left(\nabla \mathcal{L}_H(\boldsymbol{\theta_{t-1}}) - \nabla \mathcal{L}_H(\boldsymbol{\theta_t})\right)\,.$$

Thus, as $\|\mathbf{v}\|^2 = \langle \mathbf{v},\, \mathbf{v}\rangle$ for a real-valued vector $\mathbf{v}$, we obtain that

$$\begin{aligned}\left\|\boldsymbol{\xi_t}\right\|^2 =& \beta_t^2 \left\|\boldsymbol{\xi_{t-1}}\right\|^2 + (1-\beta_t)^2 \left\|\mathbf{g_t^H} - \nabla \mathcal{L}_H(\boldsymbol{\theta_t})\right\|^2 \\ &+ \beta_t^2 \left\|\nabla \mathcal{L}_H(\boldsymbol{\theta_{t-1}}) - \nabla \mathcal{L}_H(\boldsymbol{\theta_t})\right\|^2 \\ &+ 2\beta_t(1-\beta_t)\left\langle \boldsymbol{\xi_{t-1}},\, \mathbf{g_t^H} - \nabla \mathcal{L}_H(\boldsymbol{\theta_t})\right\rangle \\ &+ 2\beta_t^2 \left\langle \boldsymbol{\xi_{t-1}},\, \nabla \mathcal{L}_H(\boldsymbol{\theta_{t-1}}) - \nabla \mathcal{L}_H(\boldsymbol{\theta_t})\right\rangle \\ &+ 2\beta_t(1-\beta_t)\left\langle \mathbf{g_t^H} - \nabla \mathcal{L}_H(\boldsymbol{\theta_t}),\, \nabla \mathcal{L}_H(\boldsymbol{\theta_{t-1}}) - \nabla \mathcal{L}_H(\boldsymbol{\theta_t})\right\rangle.\end{aligned}$$

Recall that $\boldsymbol{\xi_{t-1}}$, $\boldsymbol{\theta_{t-1}}$, and $\boldsymbol{\theta_t}$ are deterministic functions of $\mathcal{P}_t$ (defined in (3.6)) and that $\mathbb{E}_t\left[\mathbf{g_t^H}\right] = \nabla \mathcal{L}_H(\boldsymbol{\theta_t})$. Thus, applying the conditional expectation $\mathbb{E}_t\left[\cdot\right]$ on both sides in the above, we obtain that

$$\begin{aligned}\mathbb{E}_t\left[\left\|\boldsymbol{\xi_t}\right\|^2\right] =& \beta_t^2 \left\|\boldsymbol{\xi_{t-1}}\right\|^2 + (1-\beta_t)^2 \mathbb{E}_t\left[\left\|\mathbf{g_t^H} - \nabla \mathcal{L}_H(\boldsymbol{\theta_t})\right\|^2\right] \\ &+ \beta_t^2 \left\|\nabla \mathcal{L}_H(\boldsymbol{\theta_{t-1}}) - \nabla \mathcal{L}_H(\boldsymbol{\theta_t})\right\|^2 + 2\beta_t^2 \left\langle \boldsymbol{\xi_{t-1}},\, \nabla \mathcal{L}_H(\boldsymbol{\theta_{t-1}}) - \nabla \mathcal{L}_H(\boldsymbol{\theta_t})\right\rangle.\end{aligned} \tag{6.18}$$

Under Assumption 3.2, as honest nodes compute local mini-batch gradients independently (given the current model $\boldsymbol{\theta_t}$), similarly to 3.13 in the proof of Theorem 3.1, we have $\mathbb{E}_t\left[\left\|\mathbf{g_t^H} - \nabla \mathcal{L}_H(\boldsymbol{\theta_t})\right\|^2\right] \leq \frac{\sigma^2}{b(n-f)}$. Moreover, due to Lipschitz smoothness, i.e., Assumption 3.1, we have

(continued)

$\|\nabla\mathcal{L}_H(\boldsymbol{\theta_{t-1}}) - \nabla\mathcal{L}_H(\boldsymbol{\theta_t})\| \leq L\,\|\boldsymbol{\theta_t} - \boldsymbol{\theta_{t-1}}\|$. Therefore, as

$$\left\langle \boldsymbol{\xi_{t-1}},\ \nabla\mathcal{L}_H(\boldsymbol{\theta_{t-1}}) - \nabla\mathcal{L}_H(\boldsymbol{\theta_t})\right\rangle \leq \left\|\boldsymbol{\xi_{t-1}}\right\| \|\nabla\mathcal{L}_H(\boldsymbol{\theta_{t-1}}) - \nabla\mathcal{L}_H(\boldsymbol{\theta_t})\|\ ,$$

owing to the Cauchy–Schwarz inequality, we obtain that

$$\left\langle \boldsymbol{\xi_{t-1}},\ \nabla\mathcal{L}_H(\boldsymbol{\theta_{t-1}}) - \nabla\mathcal{L}_H(\boldsymbol{\theta_t})\right\rangle \leq L\left\|\boldsymbol{\xi_{t-1}}\right\| \|\boldsymbol{\theta_t} - \boldsymbol{\theta_{t-1}}\|\ .$$

Using the above inequalities in (6.18), we obtain that

$$\begin{aligned}\mathbb{E}_t\left[\left\|\boldsymbol{\xi_t}\right\|^2\right] \leq & \beta_t^2\left\|\boldsymbol{\xi_{t-1}}\right\|^2 + (1-\beta_t)^2\frac{\sigma^2}{b(n-f)} + \beta_t^2 L^2\|\boldsymbol{\theta_t} - \boldsymbol{\theta_{t-1}}\|^2 \\ & + 2\beta_t^2 L\left\|\boldsymbol{\xi_{t-1}}\right\|\|\boldsymbol{\theta_t} - \boldsymbol{\theta_{t-1}}\|\ .\end{aligned}$$

Substituting, from (6.3) in Algorithm 6, $\boldsymbol{\theta_t} = \boldsymbol{\theta_{t-1}} - \gamma_{t-1}\,\widehat{\mathbf{m}}_{\boldsymbol{t-1}}$ in the above, we obtain that

$$\begin{aligned}\mathbb{E}_t\left[\left\|\boldsymbol{\xi_t}\right\|^2\right] \leq & \beta_t^2\left\|\boldsymbol{\xi_{t-1}}\right\|^2 + (1-\beta_t)^2\frac{\sigma^2}{b(n-f)} + \beta_t^2 L^2\gamma_{t-1}^2\|\widehat{\mathbf{m}}_{\boldsymbol{t-1}}\|^2 \\ & + 2\beta_t^2 L\gamma_{t-1}\left\|\boldsymbol{\xi_{t-1}}\right\|\|\widehat{\mathbf{m}}_{\boldsymbol{t-1}}\|\ .\end{aligned}$$

As $2\left\|\boldsymbol{\xi_{t-1}}\right\|\|\widehat{\mathbf{m}}_{\boldsymbol{t-1}}\| \leq \left\|\boldsymbol{\xi_{t-1}}\right\|^2 + \|\widehat{\mathbf{m}}_{\boldsymbol{t-1}}\|^2$, from the above, we obtain that

$$\begin{aligned}\mathbb{E}_t\left[\left\|\boldsymbol{\xi_t}\right\|^2\right] \leq & (1+\gamma_{t-1}L)\beta_t^2\left\|\boldsymbol{\xi_{t-1}}\right\|^2 + (1-\beta_t)^2\frac{\sigma^2}{b(n-f)} \\ & + \gamma_{t-1}L(1+\gamma_{t-1}L)\beta_t^2\|\widehat{\mathbf{m}}_{\boldsymbol{t-1}}\|^2\ . \end{aligned} \tag{6.19}$$

By the definition of $\boldsymbol{\delta_t}$ in (6.5), we have $\widehat{\mathbf{m}}_{\boldsymbol{t-1}} = \boldsymbol{\delta_{t-1}} + \mathbf{m}^{\mathbf{H}}_{\boldsymbol{t-1}}$. Thus, using the triangle and Jensen's inequalities, we have $\|\widehat{\mathbf{m}}_{\boldsymbol{t-1}}\|^2 \leq 2\,\|\boldsymbol{\delta_{t-1}}\|^2 + 2\left\|\mathbf{m}^{\mathbf{H}}_{\boldsymbol{t-1}}\right\|^2$. Similarly, by the definition of $\boldsymbol{\xi_t}$ in (6.17), we have $\left\|\mathbf{m}^{\mathbf{H}}_{\boldsymbol{t-1}}\right\|^2 \leq 2\left\|\boldsymbol{\xi_{t-1}}\right\|^2 + 2\,\|\nabla\mathcal{L}_H(\boldsymbol{\theta_{t-1}})\|^2$. Substituting from the latter in the former, we obtain that

$$\|\widehat{\mathbf{m}}_{\boldsymbol{t-1}}\|^2 \leq 2\,\|\boldsymbol{\delta_{t-1}}\|^2 + 4\left\|\boldsymbol{\xi_{t-1}}\right\|^2 + 4\,\|\nabla\mathcal{L}_H(\boldsymbol{\theta_{t-1}})\|^2\ .$$

(continued)

Substituting from the above in (6.19), we obtain that

$$\begin{aligned}\mathbb{E}_t\left[\left\|\boldsymbol{\xi}_{\mathbf{t}}\right\|^2\right] \leq \beta_t^2(1+\gamma_{t-1}L)\,(1+4\gamma_{t-1}L)\left\|\boldsymbol{\xi}_{\mathbf{t-1}}\right\|^2 \\ + (1-\beta_t)^2\frac{\sigma^2}{b(n-f)} \\ + 2\gamma_{t-1}L(1+\gamma_{t-1}L)\beta_t^2\left\|\boldsymbol{\delta}_{\mathbf{t-1}}\right\|^2 \\ + 4\gamma_{t-1}L(1+\gamma_{t-1}L)\beta_t^2\left\|\nabla\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t-1}})\right\|^2 .\end{aligned}$$

This concludes the proof. □

Large Momentum Coefficient Increases Deviation
For simplicity, suppose that we have a constant learning rate γ and a constant momentum coefficient β. Then, observe that the rate of growth of the deviation analyzed in Lemma 6.3 is effectively given by the factor $(1+\gamma L)\,(1+4\gamma L)\,\beta^2$. For the deviation to be stable, i.e., to be bounded eventually by a finite value, this factor needs to be less than (or equal to) 1. Specifically, if β^2 is larger than $(1+\gamma L)$, the honest nodes' average momentum can deviate arbitrarily from the average of their local gradients, which could prevent the learning algorithm from converging to a stationary point. This observation is also valid in the case when there are no adversarial nodes. This rate of growth of the deviation $\boldsymbol{\xi}_{\mathbf{t}}$, with respect to β, is contrary to that of the aggregation error $\boldsymbol{\delta}_{\mathbf{t}}$. Although controlling the former is important to the convergence of the algorithm, the latter is critical to the robustness against adversarial nodes. In the following section, we present a candidate for the momentum coefficient that favorably controls the opposing consequences of local momentum on deviation and aggregation error.

6.2 Resilience of Robust Distributed Mini-Batch Heavy Ball

In this section, we analyze the resilience (and stationary resilience) property of Robust DMHB, i.e., Algorithm 6. For the analysis, we make use of the above results pertaining to the aggregation error and the momentum deviation, i.e., Lemmas 6.2 and 6.3, respectively. We begin by presenting an important sub-result that concerns the growth of the loss function $\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t}})$ over the trajectory of the parameter vector $\boldsymbol{\theta}_{\mathbf{t}}$ in Algorithm 6. Although studying the loss function growth is essential to characterizing the final convergence of Robust DMHB, it is not sufficient on its own, unlike the case of Robust DMGD. We need to address an additional complication. Specifically, the presence of non-vanishing momentum deviation leads to a stochastic bias in the learning procedure that must be carefully controlled.

6.2.1 Loss Function Growth Under Distributed Momentum

We analyze, in Lemma 6.4, the difference between the current loss $\mathcal{L}_H(\boldsymbol{\theta_t})$ and the previous loss $\mathcal{L}_H(\boldsymbol{\theta_{t-1}})$ for an arbitrary iteration $t > 1$ of Robust DMHB, i.e., Algorithm 6. As expected, the loss function growth depends upon both the aggregation error $\boldsymbol{\delta_{t-1}}$ and the deviation $\boldsymbol{\xi_{t-1}}$.

Lemma 6.4 *Suppose Assumption 3.1. Consider Algorithm 6 with batch-size $b \in [1, m)$. Then, for any iteration $t > 1$, the following holds true:*

$$\begin{aligned}\mathcal{L}_H(\boldsymbol{\theta_t}) - \mathcal{L}_H(\boldsymbol{\theta_{t-1}}) \leq & -\gamma_{t-1}\left(\frac{1}{2} - 2\gamma_{t-1}L\right)\|\nabla\mathcal{L}_H(\boldsymbol{\theta_{t-1}})\|^2 \\ & + \gamma_{t-1}\left(1 + 2\gamma_{t-1}L\right)\left\|\boldsymbol{\xi_{t-1}}\right\|^2 \\ & + \gamma_{t-1}\left(1 + \gamma_{t-1}L\right)\|\boldsymbol{\delta_{t-1}}\|^2 \,.\end{aligned}$$

Proof Consider an arbitrary iteration $t > 1$. Since $\mathcal{L}_H$ is Lipschitz smooth function with coefficient L, owing to Assumption 3.1, we have

$$\mathcal{L}_H(\boldsymbol{\theta_t}) - \mathcal{L}_H(\boldsymbol{\theta_{t-1}}) \leq \langle \boldsymbol{\theta_t} - \boldsymbol{\theta_{t-1}},\, \nabla\mathcal{L}_H(\boldsymbol{\theta_{t-1}})\rangle + \frac{L}{2}\|\boldsymbol{\theta_t} - \boldsymbol{\theta_{t-1}}\|^2 \,.$$

For details on the derivation of the above, see Lemma A.1 in Appendix A.1. Substituting from (6.4), $\boldsymbol{\theta_t} = \boldsymbol{\theta_{t-1}} - \gamma_{t-1}\,\mathbf{m}^{\mathbf{H}}_{\mathbf{t-1}} - \gamma_{t-1}\,\boldsymbol{\delta_{t-1}}$, we obtain that

$$\begin{aligned}\mathcal{L}_H(\boldsymbol{\theta_t}) - \mathcal{L}_H(\boldsymbol{\theta_{t-1}}) \leq & -\gamma_{t-1}\left\langle \mathbf{m}^{\mathbf{H}}_{\mathbf{t-1}},\, \nabla\mathcal{L}_H(\boldsymbol{\theta_{t-1}})\right\rangle \\ & - \gamma_{t-1}\langle \boldsymbol{\delta_{t-1}},\, \nabla\mathcal{L}_H(\boldsymbol{\theta_{t-1}})\rangle \\ & + \gamma_{t-1}^2\frac{L}{2}\left\|\mathbf{m}^{\mathbf{H}}_{\mathbf{t-1}} + \boldsymbol{\delta_{t-1}}\right\|^2 \,.\end{aligned}$$

Substituting from (6.17), $\mathbf{m}^{\mathbf{H}}_{\mathbf{t-1}} = \nabla\mathcal{L}_H(\boldsymbol{\theta_{t-1}}) + \boldsymbol{\xi_{t-1}}$, we obtain that

$$\begin{aligned}&\mathcal{L}_H(\boldsymbol{\theta_t}) - \mathcal{L}_H(\boldsymbol{\theta_{t-1}}) \leq -\gamma_{t-1}\|\nabla\mathcal{L}_H(\boldsymbol{\theta_{t-1}})\|^2 - \gamma_{t-1}\left\langle \boldsymbol{\xi_{t-1}},\, \nabla\mathcal{L}_H(\boldsymbol{\theta_{t-1}})\right\rangle \\ &- \gamma_{t-1}\langle \boldsymbol{\delta_{t-1}},\, \nabla\mathcal{L}_H(\boldsymbol{\theta_{t-1}})\rangle + \gamma_{t-1}^2\frac{L}{2}\left\|\mathbf{m}^{\mathbf{H}}_{\mathbf{t-1}} + \boldsymbol{\delta_{t-1}}\right\|^2 \,. \qquad (6.20)\end{aligned}$$

Next, we separately analyze the last three terms on the right-hand side.

(continued)

First, by Cauchy–Schwarz inequality, as $2a_1a_2 \leq \frac{1}{a_3}(a_1)^2 + a_3(a_2)^2,\ \forall a_3 > 0$, we obtain that

$$2\left|\left\langle \boldsymbol{\xi}_{\mathbf{t-1}},\ \nabla\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t-1}})\right\rangle\right| \leq 2\left\|\boldsymbol{\xi}_{\mathbf{t-1}}\right\| \left\|\nabla\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t-1}})\right\| \leq 2\left\|\boldsymbol{\xi}_{\mathbf{t-1}}\right\|^2 + \frac{1}{2}\left\|\nabla\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t-1}})\right\|^2 . \tag{6.21}$$

Second, similarly to (6.21), we obtain that

$$2\left|\langle \boldsymbol{\delta}_{\mathbf{t-1}},\ \nabla\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t-1}})\rangle\right| \leq 2\left\|\boldsymbol{\delta}_{\mathbf{t-1}}\right\|^2 + \frac{1}{2}\left\|\nabla\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t-1}})\right\|^2 . \tag{6.22}$$

Last, using the triangle and Jensen's inequalities, we obtain that

$$\left\|\mathbf{m}^{\mathbf{H}}_{\mathbf{t-1}} + \boldsymbol{\delta}_{\mathbf{t-1}}\right\|^2 \leq 2\left\|\mathbf{m}^{\mathbf{H}}_{\mathbf{t-1}}\right\|^2 + 2\left\|\boldsymbol{\delta}_{\mathbf{t-1}}\right\|^2 .$$

Substituting $\mathbf{m}^{\mathbf{H}}_{\mathbf{t-1}} = \nabla\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t-1}}) + \boldsymbol{\xi}_{\mathbf{t-1}}$, and repeating the above reasoning due to the triangle and Jensen's inequalities, we obtain that

$$\left\|\mathbf{m}^{\mathbf{H}}_{\mathbf{t-1}} + \boldsymbol{\delta}_{\mathbf{t-1}}\right\|^2 \leq 4\left\|\boldsymbol{\xi}_{\mathbf{t-1}}\right\|^2 + 4\left\|\nabla\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t-1}})\right\|^2 + 2\left\|\boldsymbol{\delta}_{\mathbf{t-1}}\right\|^2 . \tag{6.23}$$

From (6.21), (6.22), and (6.23), we obtain that

$$-\left\langle \boldsymbol{\xi}_{\mathbf{t-1}},\ \nabla\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t-1}})\right\rangle - \langle \boldsymbol{\delta}_{\mathbf{t-1}},\ \nabla\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t-1}})\rangle + \gamma_{t-1}\frac{L}{2}\left\|\mathbf{m}^{\mathbf{H}}_{\mathbf{t-1}} + \boldsymbol{\delta}_{\mathbf{t-1}}\right\|^2 \leq$$
$$(1 + 2\gamma_{t-1}L)\left\|\boldsymbol{\xi}_{\mathbf{t-1}}\right\|^2 + (1 + \gamma_{t-1}L)\left\|\boldsymbol{\delta}_{\mathbf{t-1}}\right\|^2$$
$$+ \left(\frac{1}{2} + 2\gamma_{t-1}L\right)\left\|\nabla\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t-1}})\right\|^2 .$$

Multiplying both sides by γ_{t-1} and then plugging the resulting inequality in (6.20) conclude the proof. □

Complications Due to Momentum Deviation

We note that, compared to the case of Robust DMGD, the loss function growth in Robust DMHB depends on an additional quantity that is the momentum deviation $\boldsymbol{\xi}_{\mathbf{t}}$. Recall, from Lemma 6.3, that we do not have, unlike the aggregation error in Lemma 6.2, a uniform bound on momentum deviation at our disposal, which makes the convergence analysis of Algorithm 6 more intricate. To overcome this challenge, we resort to the general technique of *Lyapunov analysis*. Essentially, in Lyapunov

analysis, we study the growth of a candidate (auxiliary) function, referred to as a *Lyapunov* or a *potential function*, over the trajectory of an iterative algorithm (in our case, it is the Robust DMHB algorithm). A candidate Lyapunov function is a real-valued positive function of $\boldsymbol{\theta}$ whose value reduces as $\boldsymbol{\theta}$ approaches a desirable *state*, e.g., a local minimum of the average loss function $\mathcal{L}_H$. An example of a candidate Lyapunov function is the optimization error for the loss function, i.e., $\mathcal{L}_H(\boldsymbol{\theta}) - \mathcal{L}_H^*$. Recall that we studied this particular Lyapunov function in order to derive the convergence result of DMGD and Robust DMGD in Chaps. 3 and 4, respectively. However, for analyzing the convergence of Robust DMHB, we need to incorporate momentum deviation into the Lyapunov function, as presented shortly in Sect. 6.2.2. Upon analyzing the chosen Lyapunov function, we simultaneously capture the opposing effects of distributed momentum on both the aggregation error and the bias in gradients' estimates (i.e., the momentum deviation). This allows us to design a suitable constant momentum coefficient β that makes the Robust DMHB algorithm converge to an approximate stationary point with optimal asymptotic error, while offering a reasonable gradient complexity. Similar to the case of Robust DMGD, we begin with the non-convex case.

6.2.2 Case of Non-convex Loss Function

In Theorem 6.1, presented below, we derive the convergence of Robust DMHB, i.e., Algorithm 6, to (approximate) stationarity by assuming the aggregation rule F to be an (f, κ)-robust averaging. Using the convergence result, we then derive the (f, ε)-stationary resilience property that can be achieved using Robust DMHB, with the aggregation schemes described previously in Chaps. 4 and 5. We consider a constant learning rate γ, i.e., $\gamma_t = \gamma$ for all t, such that $\gamma \leq \frac{1}{c}$, where c is a positive constant. To control the growth of momentum deviation, while diminishing the impact of noise on the aggregation error at the same, we consider a constant momentum coefficient β such that

$$\beta := \sqrt{1 - c\gamma}\,. \tag{6.24}$$

We can quickly verify that, with the above value of the momentum coefficient β, the dominating growth factor for the momentum deviation, given by $(1 + \gamma L)(1 + 4\gamma L)\beta^2$ in Lemma 6.3, can be made less than 1 by choosing c sufficiently large (e.g., $c > 5L$). This means that we can control the deviation between the momentums and the instantaneous gradients. On the contrary, by choosing a learning rate that decreases with the total number of iterations T of the Robust DMHB algorithm, the above momentum coefficient also diminishes the effect of local gradient noise on the aggregation error analyzed in Lemma 6.2. These observations suggest that perhaps the momentum coefficient defined in (6.24) could enable convergence of the algorithm, despite the opposing effects of β on the aggregation error and the deviation of momentum vectors from true gradients.

To obtain a convergence result, we analyze the growth of a Lyapunov function $V_t : \boldsymbol{\theta_t} \mapsto \mathbb{R}_+$, defined below, over the sequence of parameter vectors $\boldsymbol{\theta_t}$ generated upon executing T iterations of Robust DMHB. Specifically,

$$V_t \coloneqq 2\left(\mathcal{L}_H(\boldsymbol{\theta_t}) - \mathcal{L}_H^*\right) + \frac{1}{8L}\left\|\boldsymbol{\xi_t}\right\|^2 . \tag{6.25}$$

Recall that $\mathcal{L}_H^* \coloneqq \min_{\boldsymbol{\theta}\in\mathbb{R}^d} \mathcal{L}_H(\boldsymbol{\theta})$, L denotes a Lipschitz smoothness coefficient of the point-wise loss function $\ell(\cdot, \boldsymbol{\theta})$ (with respect to $\boldsymbol{\theta}$) defined in Assumption 3.1, and $\boldsymbol{\xi_t} \coloneqq \mathbf{m_t^H} - \nabla\mathcal{L}_H(\boldsymbol{\theta_t})$ is the momentum deviation.We see, in Lemma 6.5, that the growth of this particular Lyapunov function can be uniformly bounded from above, unlike that of the loss function $\mathcal{L}_H(\boldsymbol{\theta_t})$. The bound can be controlled by tuning the learning rate γ, when the momentum coefficient β is defined by (6.24).

Lemma 6.5 *Suppose Assumptions 3.1, 3.2, and 4.1. Consider Algorithm 6 with an (f, κ)-robust averaging rule F, a batch-size $b \in [1, m)$, $\gamma_t = \gamma \leq \frac{1}{24L}$ for each iteration t, and a constant momentum coefficient β defined by (6.24) with $c = 24L$. Then, for V_t defined in (6.25), we obtain that, for each iteration $t > 1$,*

$$\mathbb{E}\left[V_t - V_{t-1}\right] \leq -\frac{\gamma}{4}\,\mathbb{E}\left[\left\|\nabla\mathcal{L}_H(\boldsymbol{\theta_{t-1}})\right\|^2\right] + 3\gamma\,\mathbb{E}\left[\left\|\boldsymbol{\delta_{t-1}}\right\|^2\right] + \frac{(1-\beta)^2\sigma^2}{8Lb(n-f)} .$$

Proof Consider an arbitrary iteration $t > 1$. Denoting $z \coloneqq \frac{1}{8L}$, we can write

$$V_t = 2\left(\mathcal{L}_H(\boldsymbol{\theta_t}) - \mathcal{L}_H^*\right) + z\left\|\boldsymbol{\xi_t}\right\|^2 . \tag{6.26}$$

From Lemma 6.3, upon substituting $\gamma_{t-1} = \gamma$, $\beta_t = \beta$, and applying the total expectation $\mathbb{E}[\cdot]$ on both sides of the inequality, we obtain that

$$\begin{aligned}\mathbb{E}\left[\left\|\boldsymbol{\xi_t}\right\|^2\right] \leq & (1+\gamma L)\,(1+4\gamma L)\,\beta^2\,\mathbb{E}\left[\left\|\boldsymbol{\xi_{t-1}}\right\|^2\right] \\ & + 2L\gamma(1+\gamma L)\beta^2\,\mathbb{E}\left[\left\|\boldsymbol{\delta_{t-1}}\right\|^2\right] \\ & + 4L\gamma(1+\gamma L)\beta^2\,\mathbb{E}\left[\left\|\nabla\mathcal{L}_H(\boldsymbol{\theta_{t-1}})\right\|^2\right] + \frac{(1-\beta)^2\sigma^2}{b(n-f)} .\end{aligned} \tag{6.27}$$

(continued)

Next, from Lemma 6.4, we obtain that

$$\mathbb{E}\left[\mathcal{L}_H(\boldsymbol{\theta_t}) - \mathcal{L}_H(\boldsymbol{\theta_{t-1}})\right] \leq -\gamma\left(\frac{1}{2} - 2\gamma L\right)\mathbb{E}\left[\|\nabla\mathcal{L}_H(\boldsymbol{\theta_{t-1}})\|^2\right] + \gamma\,(1+2\gamma L)\,\mathbb{E}\left[\left\|\boldsymbol{\xi_{t-1}}\right\|^2\right] + \gamma\,(1+\gamma L)\,\mathbb{E}\left[\|\boldsymbol{\delta_{t-1}}\|^2\right]. \tag{6.28}$$

Thus, substituting from (6.28) and (6.27) in (6.26), we obtain that

$$\begin{aligned}\mathbb{E}\left[V_t - V_{t-1}\right] &= 2\,\mathbb{E}\left[\mathcal{L}_H(\boldsymbol{\theta_t}) - \mathcal{L}_H(\boldsymbol{\theta_{t-1}})\right] + z\,\mathbb{E}\left[\left\|\boldsymbol{\xi_t}\right\|^2 - \left\|\boldsymbol{\xi_{t-1}}\right\|^2\right]\\ &\leq \left(-2\gamma\left(\frac{1}{2} - 2\gamma L\right) + 4zL\gamma(1+\gamma L)\beta^2\right)\mathbb{E}\left[\|\nabla\mathcal{L}_H(\boldsymbol{\theta_{t-1}})\|^2\right]\\ &\quad + \left(2\gamma\,(1+2\gamma L) + z(1+\gamma L)\,(1+4\gamma L)\,\beta^2 - z\right)\mathbb{E}\left[\left\|\boldsymbol{\xi_{t-1}}\right\|^2\right]\\ &\quad + \left(2\gamma\,(1+\gamma L) + z2L\gamma(1+\gamma L)\beta^2\right)\mathbb{E}\left[\|\boldsymbol{\delta_{t-1}}\|^2\right] + z\frac{(1-\beta)^2\sigma^2}{b(n-f)}.\end{aligned}$$

Let us denote

$$A \coloneqq 2\left(\frac{1}{2} - 2\gamma L\right) - 4zL(1+\gamma L)\beta^2, \tag{6.29}$$

$$B \coloneqq 2\gamma\,(1+2\gamma L) + z(1+\gamma L)\,(1+4\gamma L)\,\beta^2 - z, \text{ and} \tag{6.30}$$

$$C \coloneqq 2\gamma\,(1+\gamma L) + z2L\gamma(1+\gamma L)\beta^2. \tag{6.31}$$

Thus, we can rewrite the above inequality as

$$\begin{aligned}\mathbb{E}\left[V_t - V_{t-1}\right] \leq &-\gamma A\,\mathbb{E}\left[\|\nabla\mathcal{L}_H(\boldsymbol{\theta_{t-1}})\|^2\right]\\ &+ B\,\mathbb{E}\left[\left\|\boldsymbol{\xi_{t-1}}\right\|^2\right] + C\,\mathbb{E}\left[\|\boldsymbol{\delta_{t-1}}\|^2\right]\\ &+ z(1-\beta)^2\frac{\sigma^2}{b(n-f)}.\end{aligned} \tag{6.32}$$

Next, we separately analyze A, B, and C.

Term A. Upon substituting $z = \frac{1}{8L}$ in (6.29), and recalling that $\gamma \leq \frac{1}{24L}$ and $\beta^2 < 1$, we obtain that

$$A \geq 1 - 4\gamma L - \frac{4L}{8L}(1+\gamma L) \geq \frac{1}{2} - \frac{9\gamma L}{2} \geq \frac{1}{2} - \frac{3}{16} \geq \frac{1}{4}. \tag{6.33}$$

(continued)

Term B. From (6.30), we obtain that

$$\begin{aligned} B &= 2\gamma\,(1+2\gamma L) + z\beta^2\left(1+5\gamma L+4\gamma^2L^2\right) - z \\ &= -\left(1-\beta^2\right)z + \gamma\left(2+4\gamma L+5z\beta^2L+4z\beta^2L\gamma L\right). \end{aligned}$$

Substituting $z=\frac{1}{8L}$, and recalling that $\beta^2\leq 1$ and $\gamma\leq\frac{1}{24L}$, we obtain that

$$B \leq -(1-\beta^2)\frac{1}{8L} + \gamma\left(2+\frac{4}{24}+\frac{5}{8}+\frac{4}{24\times 8}\right) \leq -(1-\beta^2)\frac{1}{8L}+3\gamma\,.$$

Substituting from (6.24) with $c=24L$, $1-\beta^2=24\gamma L$ in the above, we obtain that

$$B \leq -24\gamma L\,\frac{1}{8L}+3\gamma = 0\,. \tag{6.34}$$

Term C. Substituting $z=\frac{1}{8L}$ in (6.31), and recalling that $\beta^2<1$ and $\gamma\leq\frac{1}{24L}$, we obtain that

$$C \leq 2\gamma\left(1+\gamma L+\frac{1}{8}(1+\gamma L)\right) \leq \frac{9\gamma}{4}\,(1+\gamma L) \leq \frac{9\gamma}{4}\left(1+\frac{1}{24}\right) \leq 3\gamma\,. \tag{6.35}$$

Substituting from (6.33), (6.34), and (6.35) into (6.32), we obtain that

$$\begin{aligned} \mathbb{E}\left[V_t - V_{t-1}\right] \leq &-\frac{\gamma}{4}\,\mathbb{E}\left[\|\nabla\mathcal{L}_H(\boldsymbol{\theta_{t-1}})\|^2\right] \\ &+3\gamma\,\mathbb{E}\left[\|\boldsymbol{\delta_{t-1}}\|^2\right] + z\frac{(1-\beta)^2\sigma^2}{b(n-f)}\,. \end{aligned}$$

Recall that $z=\frac{1}{8L}$ in the above. This concludes the proof. □

In the above analysis, we obtain an upper bound on the growth of the considered Lyapunov function V_t. Note that this upper bound is independent of the momentum deviation. Consequently, as the dependence on the aggregation error $\mathbb{E}\left[\|\boldsymbol{\delta_{t-1}}\|^2\right]$ can be uniformly bounded under constant momentum, thanks to Lemma 6.2, we can now obtain an upper bound on the average of the squared gradient norm for the loss function over the trajectory of Robust DMHB (i.e., Algorithm 6). Specifically, we have the following convergence result.

Theorem 6.1 *Suppose Assumptions 3.1, 3.2, and 4.1. Consider $T \geq 2$ iterations of Algorithm 6 with an (f, κ)-robust averaging F, a batch-size $b \in [1, m)$, a constant learning rate, i.e., $\gamma_t = \gamma \leq \frac{1}{24L}$ for all $t \in [T]$, and a constant momentum coefficient β defined by (6.24) with $c = 24L$. Then, the following holds true:*

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}_H(\boldsymbol{\theta_t})\|^2\right] \leq \frac{9\left(\mathcal{L}_H(\boldsymbol{\theta_1}) - \mathcal{L}_H^*\right)}{\gamma T} + 576L\frac{\sigma^2}{b}\left(3\kappa + \frac{1}{n-f}\right)\gamma + 36\kappa\zeta\,.$$

Proof From Lemma 6.5, we have, for all $t \in [T+1]$ such that $t > 1$,

$$\mathbb{E}\left[V_t - V_{t-1}\right] \leq -\frac{\gamma}{4}\,\mathbb{E}\left[\|\nabla\mathcal{L}_H(\boldsymbol{\theta_{t-1}})\|^2\right] + 3\gamma\,\mathbb{E}\left[\|\boldsymbol{\delta_{t-1}}\|^2\right] + \frac{(1-\beta)^2\sigma^2}{8Lb(n-f)}\,.$$

Taking summation on both sides from $t = 2$ to $t = T + 1$, we obtain that

$$\mathbb{E}\left[V_{T+1} - V_1\right] \leq -\frac{\gamma}{4}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}_H(\boldsymbol{\theta_t})\|^2\right] + 3\gamma\sum_{t=1}^{T}\mathbb{E}\left[\|\boldsymbol{\delta_t}\|^2\right] + \frac{(1-\beta)^2\sigma^2}{8Lb(n-f)}T\,.$$

Note, by definition of V_t in (6.25), that $V_{T+1} \geq 0$. Thus, from the above, we get

$$\frac{\gamma}{4}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}_H(\boldsymbol{\theta_t})\|^2\right] \leq \mathbb{E}\left[V_1\right] + 3\gamma\sum_{t=1}^{T}\mathbb{E}\left[\|\boldsymbol{\delta_t}\|^2\right] + \frac{(1-\beta)^2\sigma^2}{8Lb(n-f)}\,T\,. \tag{6.36}$$

As $\beta \geq 0$ and $1 - \beta^2 = 24L\,\gamma$ (obtain upon substituting $c = 24L$ in (6.24)), we obtain that

$$(1-\beta)^2 = \frac{\left(1-\beta^2\right)^2}{(1+\beta)^2} \leq \left(1-\beta^2\right)^2 = 576L^2\,\gamma^2\,. \tag{6.37}$$

(continued)

Substituting from above in (6.36), we obtain that

$$\frac{\gamma}{4}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}_H(\boldsymbol{\theta_t})\|^2\right] \leq \mathbb{E}\left[V_1\right] + 3\gamma\sum_{t=1}^{T}\mathbb{E}\left[\|\boldsymbol{\delta_t}\|^2\right] + \frac{72L\sigma^2}{b(n-f)}\gamma^2 T\,.$$

Multiplying both sides by $\frac{4}{\gamma T}$, we obtain that

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}_H(\boldsymbol{\theta_t})\|^2\right] \leq \frac{4}{\gamma T}\,\mathbb{E}\left[V_1\right] + \frac{12}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\boldsymbol{\delta_t}\|^2\right] + \frac{288L\sigma^2}{b(n-f)}\gamma\,. \tag{6.38}$$

We obtain next an upper bound on the accumulated expected aggregation error $\sum_{t=1}^{T}\mathbb{E}\left[\|\boldsymbol{\delta_t}\|^2\right]$, thanks to Lemma 6.2.

Invoking Lemma 6.2. Since F is assumed to be an (f,κ)-robust averaging, from Lemma 6.2 we have, for all $t \in [T+1]$,

$$\mathbb{E}\left[\|\boldsymbol{\delta_t}\|^2\right] \leq 6\kappa\left(\frac{1-\beta}{1+\beta}\right)\frac{\sigma^2}{b} + 3\kappa\zeta\,.$$

Similarly to (6.37), we have $\frac{1-\beta}{1+\beta} \leq 24L\gamma$. Using this above, we obtain that

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\boldsymbol{\delta_t}\|^2\right] \leq (6\times 24)\kappa L\gamma\frac{\sigma^2}{b} + 3\kappa\zeta = 144\kappa L\,\gamma\frac{\sigma^2}{b} + 3\kappa\zeta\,. \tag{6.39}$$

Substituting from (6.39) in (6.38), we obtain that

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}_H(\boldsymbol{\theta_t})\|^2\right] \leq \frac{4}{\gamma T}\,\mathbb{E}\left[V_1\right] + 1728\kappa L\,\gamma\frac{\sigma^2}{b} + \frac{288L\sigma^2}{b(n-f)}\gamma + 36\kappa\zeta\,. \tag{6.40}$$

For a penultimate step, we obtain an upper bound on V_1.

Bound on V_1. By the definition of V_t in (6.25), and the fact that $\boldsymbol{\theta_1}$ is given a priori, i.e., $\mathbb{E}\left[\mathcal{L}_H(\boldsymbol{\theta_1})\right] = \mathcal{L}_H(\boldsymbol{\theta_1})$, we have

$$\mathbb{E}\left[V_1\right] \coloneqq 2\left(\mathcal{L}_H(\boldsymbol{\theta_1}) - \mathcal{L}_H^*\right) + \frac{1}{8L}\,\mathbb{E}\left[\left\|\boldsymbol{\xi_1}\right\|^2\right]. \tag{6.41}$$

(continued)

By the definition of $\boldsymbol{\xi_t}$ in (6.17)

$$\boldsymbol{\xi_1} = \boldsymbol{m_1^H} - \nabla \mathcal{L}_H(\boldsymbol{\theta_1}) \,.$$

Recall that $\boldsymbol{m_0^{(i)}} = 0$ for all $i \in H$. Thus, $\boldsymbol{m_1^H} = (1-\beta)\mathbf{g_1^H}$ (see (6.1)). Therefore,

$$\begin{aligned}\mathbb{E}\left[\left\|\boldsymbol{\xi_1}\right\|^2\right] &= \mathbb{E}\left[\left\|\boldsymbol{m_1^H} - \nabla \mathcal{L}_H(\boldsymbol{\theta_1})\right\|^2\right] = \mathbb{E}\left[\left\|(1-\beta)\mathbf{g_1^H} - \nabla \mathcal{L}_H(\boldsymbol{\theta_1})\right\|^2\right] \\ &= \mathbb{E}\left[\left\|(1-\beta)\left(\mathbf{g_1^H} - \nabla \mathcal{L}_H(\boldsymbol{\theta_1})\right) - \beta \nabla \mathcal{L}_H(\boldsymbol{\theta_1})\right\|^2\right].\end{aligned}$$

Expanding the right-hand side, we obtain that

$$\begin{aligned}\mathbb{E}\left[\left\|\boldsymbol{\xi_1}\right\|^2\right] =& (1-\beta)^2\, \mathbb{E}\left[\left\|\mathbf{g_1^H} - \nabla \mathcal{L}_H(\boldsymbol{\theta_1})\right\|^2\right] + \beta^2 \left\|\nabla \mathcal{L}_H(\boldsymbol{\theta_1})\right\|^2 \\ &- 2\beta(1-\beta)\left\langle \mathbb{E}\left[\mathbf{g_1^H}\right] - \nabla \mathcal{L}_H(\boldsymbol{\theta_1}),\, \nabla \mathcal{L}_H(\boldsymbol{\theta_1})\right\rangle.\end{aligned}$$

Recall that $\mathbb{E}\left[\mathbf{g_1^H}\right] = \nabla \mathcal{L}_H(\boldsymbol{\theta_1})$. Recall also that, due to Assumption 3.2 and the independent computations of local mini-batch gradients by the honest nodes, $\mathbb{E}\left[\left\|\mathbf{g_1^H} - \nabla \mathcal{L}_H(\boldsymbol{\theta_1})\right\|^2\right] \leq \frac{\sigma^2}{b(n-f)}$. Using these in the above, we obtain that

$$\mathbb{E}\left[\left\|\boldsymbol{\xi_1}\right\|^2\right] \leq \frac{(1-\beta)^2\sigma^2}{b(n-f)} + \beta^2 \left\|\nabla \mathcal{L}_H(\boldsymbol{\theta_1})\right\|^2 .$$

As the loss function $\mathcal{L}_H(\boldsymbol{\theta})$ is Lipschitz smooth, due to Assumption 3.1, we have $\left\|\nabla \mathcal{L}_H(\boldsymbol{\theta})\right\|^2 \leq 2L(\mathcal{L}_H(\boldsymbol{\theta}) - \mathcal{L}_H^*)$ for all $\boldsymbol{\theta}$ (see Appendix A.2). Therefore,

$$\mathbb{E}\left[\left\|\boldsymbol{\xi_1}\right\|^2\right] \leq \frac{(1-\beta)^2\sigma^2}{b(n-f)} + 2\beta^2 L\left(\mathcal{L}_H(\boldsymbol{\theta_1}) - \mathcal{L}_H^*\right) .$$

Substituting from above in (6.41), we obtain that

$$\begin{aligned}\mathbb{E}\left[V_1\right] \leq& 2\left(\mathcal{L}_H(\boldsymbol{\theta_1}) - \mathcal{L}_H^*\right) \\ &+ \frac{1}{8L}\left(\frac{(1-\beta)^2\sigma^2}{b(n-f)} + 2\beta^2 L\left(\mathcal{L}_H(\boldsymbol{\theta_1}) - \mathcal{L}_H^*\right)\right) .\end{aligned}$$

(continued)

As $\beta < 1$ and $(1-\beta)^2 \leq \left(1-\beta^2\right)^2 = 576\gamma^2 L^2$ (see (6.37)), from above, we obtain that

$$\mathbb{E}\left[V_1\right] \leq \frac{9}{4}\left(\mathcal{L}_H(\boldsymbol{\theta_1}) - \mathcal{L}_H^*\right) + \frac{72L\sigma^2}{b(n-f)}\gamma^2 . \tag{6.42}$$

Concluding Step. Substituting from (6.42) in (6.40), we obtain that

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}_H(\boldsymbol{\theta_t})\|^2\right] \leq \frac{4}{\gamma T}\left(\frac{9}{4}\left(\mathcal{L}_H(\boldsymbol{\theta_1}) - \mathcal{L}_H^*\right) + \frac{72L\sigma^2}{b(n-f)}\gamma^2\right) + 1728\kappa L\gamma\frac{\sigma^2}{b} + \frac{288L\sigma^2}{b(n-f)}\gamma + 36\kappa\zeta .$$

We can rewrite the above as

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}_H(\boldsymbol{\theta_t})\|^2\right] \leq \frac{9\left(\mathcal{L}_H(\boldsymbol{\theta_1}) - \mathcal{L}_H^*\right)}{\gamma T} + 288L\frac{\sigma^2}{b}\left(6\kappa + \frac{1}{n-f}\right)\gamma + \left(\frac{288L\sigma^2}{b(n-f)}\right)\frac{\gamma}{T} + 36\kappa\zeta .$$

As $\frac{\gamma}{T} \leq \gamma$, from the above we obtain that

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}_H(\boldsymbol{\theta_t})\|^2\right] \leq \frac{9\left(\mathcal{L}_H(\boldsymbol{\theta_1}) - \mathcal{L}_H^*\right)}{\gamma T} + 288L\frac{\sigma^2}{b}\left(6\kappa + \frac{1}{n-f}\right)\gamma + \left(\frac{288L\sigma^2}{b(n-f)}\right)\gamma + 36\kappa\zeta .$$

Merging the 2nd and 3rd terms on the right-hand side concludes the proof. □

Upon considering an optimal choice for the learning rate γ, as per the convergence result presented in Theorem 6.1, we analyze below the robustness guarantee achievable by executing T iterations of Robust DMHB.

Stationary Resilience by Implementing Algorithm 6
Let Δ_H be a real value such that $\mathcal{L}_H(\boldsymbol{\theta_1}) - \mathcal{L}_H^* \leq \Delta_H$. Then, the upper bound obtained in Theorem 6.1 reduces to the following:

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}_H(\boldsymbol{\theta_t})\|^2\right] \leq \frac{9\Delta_H}{\gamma T} + 576L\frac{\sigma^2}{b}\left(3\kappa + \frac{1}{n-f}\right)\gamma + 36\kappa\zeta\,. \tag{6.43}$$

We note that the above upper bound attains its minimum value for learning rate γ that solves the following equation:

$$\frac{9\Delta_H}{\gamma T} = 576L\frac{\sigma^2}{b}\left(3\kappa + \frac{1}{n-f}\right)\gamma\,.$$

That is to say, assuming $\sigma > 0$, an optimal choice for the learning rate is given by

$$\gamma = \frac{1}{8}\sqrt{\left(\frac{n-f}{3\kappa(n-f)+1}\right)\frac{\Delta_H b}{\sigma^2 LT}}\,. \tag{6.44}$$

In light of the above observation, we obtain the following corollary of Theorem 6.1.

Corollary 6.1 *Suppose Assumptions 3.1, 3.2, and 4.1 with $\sigma > 0$. Consider $T \geq 2$ iterations of Algorithm 6 with an (f,κ)-robust averaging F and a batch-size $b \in [1,m)$. If, for all $t \in [T]$, $\gamma_t = \gamma$ such that*

$$\gamma = \min\left\{\frac{1}{24L}, \frac{1}{8}\sqrt{\left(\frac{n-f}{3\kappa(n-f)+1}\right)\frac{\Delta_H b}{\sigma^2 LT}}\right\}$$

and $\beta_t = \beta$, where β is defined by (6.24) with $c = 24$, then, provided that $T \geq \left(\frac{n-f}{3\kappa(n-f)+1}\right)\frac{9L\Delta_H b}{\sigma^2}$, the following holds true:

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}_H(\boldsymbol{\theta_t})\|^2\right] \leq 144\sqrt{\left(3\kappa + \frac{1}{n-f}\right)\frac{\Delta_H L\sigma^2}{Tb}} + 36\kappa\zeta\,.$$

Proof Note that $\gamma \leq \frac{1}{24L}$. Therefore, the result shown in Theorem 6.1 holds true. When $\sigma > 0$ and $T \geq \left(\frac{n-f}{3\kappa(n-f)+1}\right)\frac{9L\Delta_H}{\sigma^2}$, γ is defined by (6.44). Substituting this particular value of γ in (6.43) concludes the proof. □

According to Corollary 6.1, if the server samples a model randomly from the set of all the models generated, then (in expectation) the model approximates a critical point of the average loss function $\mathcal{L}_H(\boldsymbol{\theta})$. Specifically, upon executing T iterations of Algorithm 6, under the conditions stated in Corollary 6.1, if the server outputs $\hat{\boldsymbol{\theta}}$ chosen uniformly from the set of intermediate vectors $(\boldsymbol{\theta_1}, \ldots, \boldsymbol{\theta_T})$, then, despite the presence of f adversarial nodes, we obtain that

$$\mathbb{E}\left[\left\|\nabla \mathcal{L}_H(\hat{\boldsymbol{\theta}})\right\|^2\right] = \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla \mathcal{L}_H(\boldsymbol{\theta_t})\|^2\right] \in O_T\left(\sqrt{\left(\kappa + \frac{1}{n-f}\right)\frac{\sigma^2}{bT}} + \kappa\zeta\right),$$

where the expectation $\mathbb{E}[\cdot]$ in the above also accounts for the randomness in the choice of $\hat{\boldsymbol{\theta}}$, which is independent to the randomness due to the local data sampling by the honest nodes. Hence, upon running T iterations of Algorithm 6 with specifications in Corollary 6.1, and outputting a random model from the set $(\boldsymbol{\theta_1}, \ldots, \boldsymbol{\theta_T})$, we achieve (in expectation) (f, ε)-stationary resilience where

$$\varepsilon \in O_T\left(\sqrt{\left(\kappa + \frac{1}{n-f}\right)\frac{\sigma^2}{bT}} + \kappa\zeta\right). \tag{6.45}$$

From (6.45), we can obtain the training error for a given aggregation rule by substituting the corresponding value of the robust coefficient κ. In Table 6.1, we summarize the stationary resilience properties for the aggregation rules presented and studied in Chaps. 4 and 5. Let us remark on some important differences between the training errors for Robust DMHB and Robust DMGD, in the following.

Comparisons with Robust DMGD

Let us denote by $\varepsilon^{\text{DMGD}}$ the training error achieved upon executing T iterations of the Robust DMGD algorithm (i.e., Algorithm 5) in the presence of up to f adversarial nodes. Recall, from (4.30), that

$$\varepsilon^{\text{DMGD}} \in O_T\left(\sqrt{\frac{\sigma^2}{(n-f)bT}} + \kappa\sigma^2 + \kappa\zeta\right).$$

Upon comparing the above with the training error achievable using Robust DMHB, shown in (6.45), we make the following two critical observations:

Asymptotic error. Local gradient noise does not affect the asymptotic training error, despite the presence of adversarial nodes, when the server updates the global model by using a robust averaging of nodes' local momentums. Moreover, the order of the asymptotic error for Robust DMHB, i.e., $O_\star(\kappa\zeta)$, matches

Table 6.1 Robust DMHB is (f, ε)-stationary resilient in the general non-convex case, for the shown values of training error ε after T iterations, with different robust aggregation rules, provided the conditions of Corollary 6.1 hold true. The training error ε for all the aggregation rules, except CWTM, is obtained by assuming $\frac{f}{n} < \frac{1}{2}$. For CWTM_f, as shown in Sect. 4.3.2, we assume that $\frac{f}{n} \leq \frac{1}{2}\left(\frac{c}{1+c}\right)$ where $c > 0$ is an arbitrary real-valued constant

Aggregation rule	Breakdown point	Training error ε
CWMed	$\frac{1}{2}$	$O_{n,T}\left(\sqrt{\frac{\sigma^2}{bT}} + \zeta\right)$
GeoMed	$\frac{1}{2}$	$O_{n,T}\left(\sqrt{\frac{\sigma^2}{bT}} + \zeta\right)$
CWTM_f	$\frac{f+1}{n}$	$O_{n,T}\left(\sqrt{\left(\frac{f+1}{n-f}\right)\frac{\sigma^2}{bT}} + \frac{f}{n}\zeta\right)$
$\text{Multi-Krum}_{f,1}$	$\frac{f+1}{n}$	$O_{n,T}\left(\sqrt{\frac{\sigma^2}{bT}} + \zeta\right)$
[any of the above] $\circ$ NNM	$\frac{f+1}{n}$	$O_{n,T}\left(\sqrt{\left(\frac{f+1}{n-f}\right)\frac{\sigma^2}{bT}} + \frac{f}{n}\zeta\right)$

the lower bound presented in Sect. 5.3, under the assumption of ζ-gradient dissimilarity.

Rate of Convergence. The Robust DMHB algorithm converges slower than Robust DMGD. Specifically, given an (f, κ)-robust averaging rule F, provided that $\zeta = 0$ (i.e., homogeneous data distribution across the honest nodes), the corresponding Robust DMHB algorithm may require up to $\kappa(n - f) + 1$ times more number of iterations than its DMGD counterpart. The main reason for this is the coupling between the momentum coefficient β and the learning rate γ defined in (6.24). In general, such a degradation of convergence rate is unavoidable due to the presence of adversarial nodes. This however does not significantly inflate the gradient complexity of the algorithm compared to Robust DMHB, specifically when f is small (see Table 6.1).

Next, we study the case when the average loss function is strongly convex. Interestingly, we see how to extend the technique of Lyapunov analysis introduced above for obtaining a tight convergence result in the strongly convex case, where the learning rate diminishes in a scheduled manner.

6.2.3 *Case of Strongly Convex Loss Function*

Recall from the analysis of Robust DMGD, in Sect. 4.4.3, that when Assumption 4.2 holds true for the average loss function $\mathcal{L}_H(\boldsymbol{\theta})$, we can obtain convergence in order $\frac{1}{T}$ upon choosing a scheduled diminishing learning rate. To produce a similar order of convergence for Robust DMHB, however, we must also schedule the momentum

coefficient (in addition to the learning rate), due to the mutual dependence between the momentum coefficient and the learning rate suggested in Sect. 6.2.2. Let us first rework the analysis on the aggregation error, presented in Lemma 6.2, for the more general case when the momentum coefficient is not constant.

Drift Between Momentums Under Changing Momentum Coefficients
Recall, from Lemma 6.1, that the empirical variance of the honest nodes' momentum vectors, i.e., $\frac{1}{|H|}\sum_{i\in H}\mathbb{E}\left[\left\|\mathbf{m_t^{(i)}}-\mathbf{m_t^H}\right\|^2\right]$, can be easily upper bounded, when the momentum coefficient is constant. Consequently, the aggregation error $\boldsymbol{\delta_t}$ could be uniformly bounded for an (f,κ)-robust averaging rule (see Lemma 6.2), which made it easier to analyze the growth of the average loss function in Lemma 6.4. Deriving a similar uniform bound in the more general case, when the rate of momentum is changing, is rather difficult. As a result, it is easier to analyze the relative growth of the empirical variance of nodes' momentum vectors and later incorporate it explicitly in the Lyapunov function.

Lemma 6.6 *Suppose Assumptions 3.2 and 4.1. Consider Algorithm 6 with a batch-size $b \in [1, m)$. Then, for each iteration $t > 1$, we have*

$$\frac{1}{|H|}\sum_{i\in H}\mathbb{E}\left[\left\|\mathbf{m_t^{(i)}}-\mathbf{m_t^H}\right\|^2\right] \leq \beta_t \frac{1}{|H|}\sum_{i,j\in H}\mathbb{E}\left[\left\|\mathbf{m_{t-1}^{(i)}}-\mathbf{m_{t-1}^H}\right\|^2\right] + (1-\beta_t)^2\frac{\sigma^2}{b} + (1-\beta_t)\zeta\,.$$

Proof Consider two arbitrary honest nodes i and j in H, and an iteration $t > 1$. By the definition of momentum vectors in (6.1), we obtain that

$$\mathbf{m_t^{(i)}}-\mathbf{m_t^{(j)}} = \beta_t\left(\mathbf{m_{t-1}^{(i)}}-\mathbf{m_{t-1}^{(j)}}\right) + (1-\beta_t)\left(\mathbf{g_t^{(i)}}-\mathbf{g_t^{(j)}}\right).$$

Taking the squared norm on both sides and expanding the right-hand side, we obtain that

$$\left\|\mathbf{m_t^{(i)}}-\mathbf{m_t^{(j)}}\right\|^2 = \beta_t^2\left\|\mathbf{m_{t-1}^{(i)}}-\mathbf{m_{t-1}^{(j)}}\right\|^2 + (1-\beta_t)^2\left\|\mathbf{g_t^{(i)}}-\mathbf{g_t^{(j)}}\right\|^2 + 2\beta_t(1-\beta_t)\left\langle\mathbf{m_{t-1}^{(i)}}-\mathbf{m_{t-1}^{(j)}},\,\mathbf{g_t^{(i)}}-\mathbf{g_t^{(j)}}\right\rangle.$$

(continued)

Applying the conditional expectation $\mathbb{E}_t[\cdot]$ in the above, and noting that $\mathbb{E}_t\left[\mathbf{g_t^{(i)}}\right] = \nabla\mathcal{L}_i(\boldsymbol{\theta_t})$ for all $i \in H$, we obtain that

$$\mathbb{E}_t\left[\left\|\mathbf{m_t^{(i)}} - \mathbf{m_t^{(j)}}\right\|^2\right] = \beta_t^2\left\|\mathbf{m_{t-1}^{(i)}} - \mathbf{m_{t-1}^{(j)}}\right\|^2 + (1-\beta_t)^2\mathbb{E}_t\left[\left\|\mathbf{g_t^{(i)}} - \mathbf{g_t^{(j)}}\right\|^2\right] + 2\beta_t(1-\beta_t)\left\langle\mathbf{m_{t-1}^{(i)}} - \mathbf{m_{t-1}^{(j)}},\ \nabla\mathcal{L}_i(\boldsymbol{\theta_t}) - \nabla\mathcal{L}_j(\boldsymbol{\theta_t})\right\rangle. \tag{6.46}$$

Note that we can write

$$\mathbf{g_t^{(i)}} - \mathbf{g_t^{(j)}} = \left(\mathbf{g_t^{(i)}} - \nabla\mathcal{L}_i(\boldsymbol{\theta_t})\right) - \left(\mathbf{g_t^{(j)}} - \nabla\mathcal{L}_j(\boldsymbol{\theta_t})\right) + \left(\nabla\mathcal{L}_i(\boldsymbol{\theta_t}) - \nabla\mathcal{L}_j(\boldsymbol{\theta_t})\right).$$

Therefore,

$$\left\|\mathbf{g_t^{(i)}} - \mathbf{g_t^{(j)}}\right\|^2 = \left\|\mathbf{g_t^{(i)}} - \nabla\mathcal{L}_i(\boldsymbol{\theta_t})\right\|^2 + \left\|\mathbf{g_t^{(j)}} - \nabla\mathcal{L}_j(\boldsymbol{\theta_t})\right\|^2 + \left\|\nabla\mathcal{L}_i(\boldsymbol{\theta_t}) - \nabla\mathcal{L}_j(\boldsymbol{\theta_t})\right\|^2 - 2\left\langle\mathbf{g_t^{(i)}} - \nabla\mathcal{L}_i(\boldsymbol{\theta_t}),\ \mathbf{g_t^{(j)}} - \nabla\mathcal{L}_j(\boldsymbol{\theta_t})\right\rangle + 2\left\langle\mathbf{g_t^{(i)}} - \nabla\mathcal{L}_i(\boldsymbol{\theta_t}) - \mathbf{g_t^{(j)}} + \nabla\mathcal{L}_j(\boldsymbol{\theta_t}),\ \nabla\mathcal{L}_i(\boldsymbol{\theta_t}) - \nabla\mathcal{L}_j(\boldsymbol{\theta_t})\right\rangle.$$

As $\mathbb{E}_t\left[\mathbf{g_t^{(i)}}\right] = \nabla\mathcal{L}_i(\boldsymbol{\theta_t})$ and $\mathbb{E}_t\left[\left\|\mathbf{g_t^{(i)}} - \nabla\mathcal{L}_i(\boldsymbol{\theta_t})\right\|^2\right] \leq \frac{\sigma^2}{b}$ for all $i \in H$ (the later due to Assumption 3.2), applying the conditional expectation $\mathbb{E}_t[\cdot]$ in the above, we obtain that

$$\mathbb{E}_t\left[\left\|\mathbf{g_t^{(i)}} - \mathbf{g_t^{(j)}}\right\|^2\right] \leq 2\frac{\sigma^2}{b} + \left\|\nabla\mathcal{L}_i(\boldsymbol{\theta_t}) - \nabla\mathcal{L}_j(\boldsymbol{\theta_t})\right\|^2.$$

Substituting from the above in (6.46), we obtain that

$$\mathbb{E}_t\left[\left\|\mathbf{m_t^{(i)}} - \mathbf{m_t^{(j)}}\right\|^2\right] \leq \beta_t^2\left\|\mathbf{m_{t-1}^{(i)}} - \mathbf{m_{t-1}^{(j)}}\right\|^2 + (1-\beta_t)^2\left\|\nabla\mathcal{L}_i(\boldsymbol{\theta_t}) - \nabla\mathcal{L}_j(\boldsymbol{\theta_t})\right\|^2 + 2\beta_t(1-\beta_t)\left\langle\mathbf{m_{t-1}^{(i)}} - \mathbf{m_{t-1}^{(j)}},\ \nabla\mathcal{L}_i(\boldsymbol{\theta_t}) - \nabla\mathcal{L}_j(\boldsymbol{\theta_t})\right\rangle + 2(1-\beta_t)^2\frac{\sigma^2}{b}.$$

(continued)

Note that we can rewrite the above as

$$\mathbb{E}_t\left[\left\|\mathbf{m_t^{(i)}}-\mathbf{m_t^{(j)}}\right\|^2\right] \leq \Big\|\beta_t\left(\mathbf{m_{t-1}^{(i)}}-\mathbf{m_{t-1}^{(j)}}\right) + (1-\beta_t)\left(\nabla\mathcal{L}_i(\boldsymbol{\theta_t})-\nabla\mathcal{L}_j(\boldsymbol{\theta_t})\right)\Big\|^2 + 2(1-\beta_t)^2\frac{\sigma^2}{b}\,.$$

Applying Jensen's inequality, from the above, we obtain that

$$\mathbb{E}_t\left[\left\|\mathbf{m_t^{(i)}}-\mathbf{m_t^{(j)}}\right\|^2\right] \leq \beta_t\left\|\mathbf{m_{t-1}^{(i)}}-\mathbf{m_{t-1}^{(j)}}\right\|^2 + (1-\beta_t)\left\|\nabla\mathcal{L}_i(\boldsymbol{\theta_t})-\nabla\mathcal{L}_j(\boldsymbol{\theta_t})\right\|^2 + 2(1-\beta_t)^2\frac{\sigma^2}{b}\,.$$

Taking the total expectation $\mathbb{E}[\cdot]$ on both sides, we have

$$\mathbb{E}\left[\left\|\mathbf{m_t^{(i)}}-\mathbf{m_t^{(j)}}\right\|^2\right] \leq \beta_t\,\mathbb{E}\left[\left\|\mathbf{m_{t-1}^{(i)}}-\mathbf{m_{t-1}^{(j)}}\right\|^2\right] + (1-\beta_t)\,\mathbb{E}\left[\left\|\nabla\mathcal{L}_i(\boldsymbol{\theta_t})-\nabla\mathcal{L}_j(\boldsymbol{\theta_t})\right\|^2\right] + 2(1-\beta_t)^2\frac{\sigma^2}{b}\,.$$

Recall that i and j in the above are chosen arbitrarily from H. Therefore, by taking the average on both sides over all the pairs $(i, j) \in H$, we obtain that

$$\frac{1}{|H|^2}\sum_{i,j\in H}\mathbb{E}\left[\left\|\mathbf{m_t^{(i)}}-\mathbf{m_t^{(j)}}\right\|^2\right] \leq \beta_t\frac{1}{|H|^2}\sum_{i,j\in H}\mathbb{E}\left[\left\|\mathbf{m_{t-1}^{(i)}}-\mathbf{m_{t-1}^{(j)}}\right\|^2\right] + (1-\beta_t)\frac{1}{|H|^2}\sum_{i,j\in H}\mathbb{E}\left[\left\|\nabla\mathcal{L}_i(\boldsymbol{\theta_t})-\nabla\mathcal{L}_j(\boldsymbol{\theta_t})\right\|^2\right] + 2(1-\beta_t)^2\frac{\sigma^2}{b}\,.$$

Recall, from Remark 4.5, that for a set of m vectors $\mathbf{v_1}, \ldots, \mathbf{v_m}$, we have $\sum_{i,j\in[m]}\left\|\mathbf{v_i}-\mathbf{v_j}\right\|^2 = 2m\sum_{i\in[m]}\left\|\mathbf{v_i}-\frac{1}{m}\sum_{j\in[m]}\mathbf{v_j}\right\|^2$. Using this in the

(continued)

above, we obtain that

$$\frac{1}{|H|}\sum_{i\in H}\mathbb{E}\left[\left\|\mathbf{m}_{\mathbf{t}}^{(\mathbf{i})}-\mathbf{m}_{\mathbf{t}}^{\mathbf{H}}\right\|^2\right]\leq\beta_t\frac{1}{|H|}\sum_{i,j\in H}\mathbb{E}\left[\left\|\mathbf{m}_{\mathbf{t-1}}^{(\mathbf{i})}-\mathbf{m}_{\mathbf{t-1}}^{\mathbf{H}}\right\|^2\right]$$
$$+(1-\beta_t)\frac{1}{|H|}\sum_{i\in H}\mathbb{E}\left[\|\nabla\mathcal{L}_i(\boldsymbol{\theta}_{\mathbf{t}})-\nabla\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t}})\|^2\right]+(1-\beta_t)^2\frac{\sigma^2}{b}\,.$$

Due to Assumption 4.1, we have $\frac{1}{|H|}\sum_{i\in H}\mathbb{E}\left[\|\nabla\mathcal{L}_i(\boldsymbol{\theta}_{\mathbf{t}})-\nabla\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t}})\|^2\right]\leq\zeta$. Using this in the above concludes the proof. □

Varying Momentum Coefficients and a New Lyapunov Function

From the above analysis, we remark that, in order to obtain a diminishing effect of gradient noise on the aggregation error (similar to the non-convex case), we must choose $1-\beta_t$ proportional to the learning rate γ_t that itself can be chosen to be diminishing. Specifically, while ensuring that $\gamma_t\leq\frac{1}{c}$ for the entire iterative procedure, where c is a positive real value, we choose

$$\beta_t=1-c\gamma_{t-1}\,,\quad\forall t\,. \tag{6.47}$$

For the first iteration, we choose $\beta_1=0$. The varying of the momentum coefficients, under a scheduled diminishing learning rate, makes it rather difficult to obtain a useful uniform bound for the aggregation error. To overcome this issue, we consider a different Lyapunov function in order to analyze the convergence of Robust DMHB in the strongly convex case. Specifically, we consider the following Lyapunov function:

$$V_t\coloneqq\left(\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t}})-\mathcal{L}_H^*\right)+\frac{1}{64L}\left(\left\|\boldsymbol{\xi}_{\mathbf{t}}\right\|^2+\kappa\frac{1}{|H|}\sum_{i\in H}\left\|\mathbf{m}_{\mathbf{t}}^{(\mathbf{i})}-\mathbf{m}_{\mathbf{t}}^{\mathbf{H}}\right\|^2\right). \tag{6.48}$$

Recall that $\boldsymbol{\xi}_{\mathbf{t}}\coloneqq\mathbf{m}_{\mathbf{t}}^{\mathbf{H}}-\nabla\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t}})$. Observe that in comparison to the Lyapunov function used in the non-convex case, we also include the drift between the momentum vectors, i.e., $\frac{1}{|H|}\sum_{i\in H}\left\|\mathbf{m}_{\mathbf{t}}^{(\mathbf{i})}-\mathbf{m}_{\mathbf{t}}^{\mathbf{H}}\right\|^2$, whose instantaneous growth for a changing rate of momentum was analyzed above in Lemma 6.6. In Theorem 6.2, we analyze the growth of this particular Lyapunov function.

Theorem 6.2 *Suppose Assumptions 3.1, 3.2, 4.1, and 4.2. Consider Algorithm 6 with an (f,κ)-robust averaging F, a batch-size $b\in[1,m)$, $\gamma_t\leq\frac{1}{144L}$ for all t, and momentum coefficients defined by (6.47) with $c=144L$. Then, for V_t defined*

in (6.48)*, for all* $t > 1$*, we have*

$$\mathbb{E}\left[V_t\right] \leq \left(1 - \frac{\mu}{4}\gamma_{t-1}\right)\mathbb{E}\left[V_{t-1}\right] + 324L\left(\kappa + \frac{1}{n-f}\right)\frac{\sigma^2}{b}\,\gamma_{t-1}^2 + \frac{9}{4}\kappa\zeta\,\gamma_{t-1}\,.$$

Proof Let us denote $z \coloneqq \frac{1}{64L}$, and

$$M_t^H = \frac{1}{|H|}\sum_{i\in H}\left\|\mathbf{m_t^{(i)}} - \mathbf{m_t^H}\right\|^2 .$$

Consider an arbitrary iteration $t > 1$. By definition of V_t in (6.48), we have

$$V_t = \left(\mathcal{L}_H(\boldsymbol{\theta_t}) - \mathcal{L}_H^*\right) + z\left(\left\|\boldsymbol{\xi_t}\right\|^2 + \kappa M_t^H\right) . \tag{6.49}$$

We next separately obtain upper bounds (in expectations) on the three terms $\mathcal{L}_H(\boldsymbol{\theta_t}) - \mathcal{L}_H^*$, $\left\|\boldsymbol{\xi_t}\right\|^2$, and M_t^H using Lemmas 6.4, 6.3, and 6.6, respectively.
Bound on $\mathcal{L}_H(\boldsymbol{\theta_t}) - \mathcal{L}_H^*$. From Lemma 6.4, we have

$$\mathcal{L}_H(\boldsymbol{\theta_t}) - \mathcal{L}_H^* \leq \left(\mathcal{L}_H(\boldsymbol{\theta_{t-1}}) - \mathcal{L}_H^*\right) - \gamma_{t-1}\left(\frac{1}{2} - 2\gamma_{t-1}L\right)\left\|\nabla\mathcal{L}_H(\boldsymbol{\theta_{t-1}})\right\|^2$$
$$+ \gamma_{t-1}\left(1 + 2\gamma_{t-1}L\right)\left\|\boldsymbol{\xi_{t-1}}\right\|^2 + \gamma_{t-1}\left(1 + \gamma_{t-1}L\right)\left\|\boldsymbol{\delta_{t-1}}\right\|^2 .$$

As the aggregation rule F is assumed to be (f,κ)-robust averaging, by Definition 4.3, we have that $\left\|\boldsymbol{\delta_t}\right\|^2 \leq \frac{\kappa}{|H|}\sum_{i\in H}\left\|\mathbf{m_t^{(i)}} - \mathbf{m_t^H}\right\|^2 = \kappa M_t^H$ for all $t \in [T]$. Using these in the above, we obtain that

$$\mathcal{L}_H(\boldsymbol{\theta_t}) - \mathcal{L}_H^* \leq \left(\mathcal{L}_H(\boldsymbol{\theta_{t-1}}) - \mathcal{L}_H^*\right) - \gamma_{t-1}\left(\frac{1}{2} - 2\gamma_{t-1}L\right)\left\|\nabla\mathcal{L}_H(\boldsymbol{\theta_{t-1}})\right\|^2$$
$$+ \gamma_{t-1}\left(1 + 2\gamma_{t-1}L\right)\left\|\boldsymbol{\xi_{t-1}}\right\|^2 + \gamma_{t-1}\left(1 + \gamma_{t-1}L\right)\kappa M_{t-1}^H .$$

Note that, as $\gamma_t \leq \frac{1}{144L}$, for all $t \in [T]$, we have $1 + 2\gamma_{t-1}L \leq 2$, and $1 + \gamma_{t-1}L \leq 2$. Using these in the above, and taking total expectation $\mathbb{E}\left[\cdot\right]$, we obtain that

$$\mathbb{E}\left[\mathcal{L}_H(\boldsymbol{\theta_t}) - \mathcal{L}_H^*\right] \leq \mathbb{E}\left[\mathcal{L}_H(\boldsymbol{\theta_{t-1}}) - \mathcal{L}_H^*\right] + 2\gamma_{t-1}\,\mathbb{E}\left[\left\|\boldsymbol{\xi_{t-1}}\right\|^2\right]$$
$$+ 2\kappa\,\gamma_{t-1}\,\mathbb{E}\left[M_{t-1}^H\right] - \gamma_{t-1}\left(\frac{1}{2} - 2\gamma_{t-1}L\right)\mathbb{E}\left[\left\|\nabla\mathcal{L}_H(\boldsymbol{\theta_{t-1}})\right\|^2\right] . \tag{6.50}$$

(continued)

Bound on $\left\|\boldsymbol{\xi}_{\mathbf{t}}\right\|^2$. From Lemma 6.3, we have

$$\mathbb{E}_t\left[\left\|\boldsymbol{\xi}_{\mathbf{t}}\right\|^2\right] \leq (1+5L\gamma_{t-1}+4L^2\gamma_{t-1}^2)\beta_t^2\left\|\boldsymbol{\xi}_{\mathbf{t-1}}\right\|^2 + (1-\beta_t)^2\frac{\sigma^2}{b(n-f)}$$
$$+ 2L\gamma_{t-1}(1+\gamma_{t-1}L)\beta_t^2\left\|\boldsymbol{\delta}_{\mathbf{t-1}}\right\|^2 + 4L\gamma_{t-1}(1+\gamma_{t-1}L)\beta_t^2\left\|\nabla\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t-1}})\right\|^2 .$$

Similarly to above, as $\gamma_t \leq \frac{1}{144L}$ for all $t \in [T]$, we have $1+5L\gamma_{t-1}+4L^2\gamma_{t-1}^2 \leq 1+5L\gamma_{t-1}+L\gamma_{t-1} = 1+6L\gamma_{t-1}$ and $1+\gamma_{t-1}L \leq 2$. Using these in the above, we obtain that

$$\mathbb{E}_t\left[\left\|\boldsymbol{\xi}_{\mathbf{t}}\right\|^2\right] \leq (1+6L\gamma_{t-1})\beta_t^2\left\|\boldsymbol{\xi}_{\mathbf{t-1}}\right\|^2 + (1-\beta_t)^2\frac{\sigma^2}{b(n-f)}$$
$$+ 4L\gamma_{t-1}\beta_t^2\left\|\boldsymbol{\delta}_{\mathbf{t-1}}\right\|^2 + 8L\gamma_{t-1}\beta_t^2\left\|\nabla\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t-1}})\right\|^2 .$$

Recall that, as F is (f,κ)-robust averaging, $\left\|\boldsymbol{\delta}_{\mathbf{t}}\right\|^2 \leq \kappa M_t^H$ for all $t \in [T]$. Using these in the above, and taking total expectation $\mathbb{E}[\cdot]$, we obtain that

$$\mathbb{E}\left[\left\|\boldsymbol{\xi}_{\mathbf{t}}\right\|^2\right] \leq (1+6L\gamma_{t-1})\beta_t^2\,\mathbb{E}\left[\left\|\boldsymbol{\xi}_{\mathbf{t-1}}\right\|^2\right] + (1-\beta_t)^2\frac{\sigma^2}{b(n-f)} \tag{6.51}$$
$$+ 4L\kappa\beta_t^2\gamma_{t-1}\,\mathbb{E}\left[M_{t-1}^H\right] + 8L\gamma_{t-1}\beta_t^2\,\mathbb{E}\left[\left\|\nabla\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t-1}})\right\|^2\right] .$$

Bound on $\mathbb{E}\left[M_t^H\right]$. Lastly, from Lemma 6.6, we have

$$\mathbb{E}\left[M_t^H\right] \leq \beta_t\,\mathbb{E}\left[M_{t-1}^H\right] + (1-\beta_t)^2\frac{\sigma^2}{b} + (1-\beta_t)\zeta . \tag{6.52}$$

Upon applying the total expectation $\mathbb{E}[\cdot]$ in (6.49), and then substituting from (6.50), (6.51), and (6.52), we obtain that

$$\mathbb{E}[V_t] \leq \mathbb{E}\left[\mathcal{L}_H(\boldsymbol{\theta}_{\mathbf{t-1}}) - \mathcal{L}_H^*\right] + 2\gamma_{t-1}\,\mathbb{E}\left[\left\|\boldsymbol{\xi}_{\mathbf{t-1}}\right\|^2\right]$$
$$+ 2\kappa\,\gamma_{t-1}\,\mathbb{E}\left[M_{t-1}^H\right] + z(1+6L\gamma_{t-1})\beta_t^2\,\mathbb{E}\left[\left\|\boldsymbol{\xi}_{\mathbf{t-1}}\right\|^2\right]$$
$$+ z(1-\beta_t)^2\frac{\sigma^2}{b(n-f)}$$
$$+ z4L\kappa\beta_t^2\gamma_{t-1}\,\mathbb{E}\left[M_{t-1}^H\right] + z\kappa\beta_t\,\mathbb{E}\left[M_{t-1}^H\right]$$

(continued)

$$
\begin{aligned}
&+ z\kappa(1-\beta_t)^2\frac{\sigma^2}{b} + z(1-\beta_t)\kappa\zeta \\
&- \gamma_{t-1}\left(\frac{1}{2} - 2\gamma_{t-1}L\right)\mathbb{E}\left[\|\nabla\mathcal{L}_H(\boldsymbol{\theta_{t-1}})\|^2\right] \\
&+ 8Lz\gamma_{t-1}\beta_t^2\,\mathbb{E}\left[\|\nabla\mathcal{L}_H(\boldsymbol{\theta_{t-1}})\|^2\right].
\end{aligned}
$$

Recall that, due to Assumption 4.2, $\|\nabla\mathcal{L}_H(\boldsymbol{\theta_{t-1}})\|^2 \geq 2\mu\left(\mathcal{L}_H(\boldsymbol{\theta_{t-1}}) - \mathcal{L}_H^*\right)$. Using this and rearranging the terms, we obtain that

$$
\begin{aligned}
\mathbb{E}[V_t] \leq &\left(1 - 2\mu\gamma_{t-1}\left(\frac{1}{2} - 2\gamma_{t-1}L - 8Lz\beta_t^2\right)\right)\mathbb{E}\left[\mathcal{L}_H(\boldsymbol{\theta_{t-1}}) - \mathcal{L}_H^*\right] \\
&+ \left(2\gamma_{t-1} + z\left(1 + 6L\gamma_{t-1}\right)\beta_t^2\right)\mathbb{E}\left[\left\|\boldsymbol{\xi_{t-1}}\right\|^2\right] \\
&+ z\left(1-\beta_t\right)^2\left(\kappa + \frac{1}{n-f}\right)\frac{\sigma^2}{b} \\
&+ \kappa\left(2\gamma_{t-1} + z4L\beta_t^2\gamma_{t-1} + z\beta_t\right)\mathbb{E}\left[M_{t-1}^H\right] + z\kappa\left(1-\beta_t\right)\zeta\,. \qquad (6.53)
\end{aligned}
$$

Let us denote

$$
A_t \coloneqq 2\gamma_{t-1} + z\left(1 + 6L\gamma_{t-1}\right)\beta_t^2\,, \quad \text{and} \quad B_t \coloneqq 2\gamma_{t-1} + z4L\beta_t^2\gamma_{t-1} + z\beta_t\,.
$$

Substituting from above in (6.53), we can rewrite

$$
\begin{aligned}
\mathbb{E}[V_t] \leq &\left(1 - 2\mu\gamma_{t-1}\left(\frac{1}{2} - 2\gamma_{t-1}L - 8Lz\beta_t^2\right)\right)\mathbb{E}\left[\mathcal{L}_H(\boldsymbol{\theta_{t-1}}) - \mathcal{L}_H^*\right] \\
&+ A_t\,\mathbb{E}\left[\left\|\boldsymbol{\xi_{t-1}}\right\|^2\right] + B_t\kappa\,\mathbb{E}\left[M_{t-1}^H\right] + z\left(1-\beta_t\right)^2\left(\kappa + \frac{1}{n-f}\right)\frac{\sigma^2}{b} \\
&+ z\left(1-\beta_t\right)\kappa\zeta\,.
\end{aligned}
$$

Note that, as $z = \frac{1}{64L}$, $\beta_t \leq 1$, and $\frac{1}{144L}$, we have

$$
\frac{1}{2} - 2\gamma_{t-1}L - 8Lz\beta_t^2 \geq \frac{1}{2} - 2\gamma_{t-1}L - \frac{1}{8} \geq \frac{1}{8}.
$$

(continued)

Therefore, from the above, we obtain that

$$\mathbb{E}[V_t] \leq \left(1 - \frac{\mu}{4}\gamma_{t-1}\right) \mathbb{E}\left[\mathcal{L}_H(\boldsymbol{\theta_{t-1}}) - \mathcal{L}_H^*\right] + A_t \, \mathbb{E}\left[\left\|\boldsymbol{\xi_{t-1}}\right\|^2\right] + B_t \kappa \, \mathbb{E}\left[M_{t-1}^H\right] + z\,(1-\beta_t)^2 \left(\kappa + \frac{1}{n-f}\right) \frac{\sigma^2}{b} + z\kappa\,(1-\beta_t)\,\zeta\,. \tag{6.54}$$

Next, we separately analyze the terms A_t and B_t and obtain, respectively, the upper bounds that ultimately show an exponential decay of the Lyapunov function with diminishing perturbations.

Upper Bound on A_t. Recall that $A_t \coloneqq 2\gamma_{t-1} + z\,(1 + 6L\gamma_{t-1})\,\beta_t^2$. Substituting $z = \frac{1}{64L}$, we obtain that

$$\begin{aligned} A_t &= z\left(128L\gamma_{t-1} + (1 + 6L\gamma_{t-1})\,\beta_t^2\right) \\ &= z\left(128L\gamma_{t-1} + \beta_t^2 + 6L\gamma_{t-1}\beta_t^2\right)\,. \end{aligned}$$

As $\beta_t \leq 1$, we have $\beta_t^2 \leq \beta_t$ and $6L\gamma_{t-1}\beta_t^2 \leq 6L\gamma_{t-1}$. Using these above, we obtain that

$$A_t \leq z\,(134L\gamma_{t-1} + \beta_t)\,.$$

Upon substituting $\beta_t = 1 - 144L\gamma_{t-1}$ and using the fact that when Lipschitz smoothness (i.e., Assumption 3.1) and strong convexity (i.e., Assumption 4.2) satisfy simultaneously, we have $L \geq \mu$ (see Appendix A.2), and we obtain that

$$A_t \leq z\,(1 - 10L\gamma_{t-1}) \leq z\,(1 - 10\mu\gamma_{t-1}) \leq z\left(1 - \frac{\mu}{4}\gamma_{t-1}\right)\,. \tag{6.55}$$

Upper Bound on B_t. Recall that $B_t \coloneqq 2\gamma_{t-1} + z4L\beta_t^2\gamma_{t-1} + z\beta_t$. Similarly to A_t above, we obtain that

$$B_t \leq z\,(132L\gamma_{t-1} + \beta_t) = z\,(1 - 12L\gamma_{t-1}) \leq z\left(1 - \frac{\mu}{4}\gamma_{t-1}\right)\,. \tag{6.56}$$

(continued)

Substituting from (6.55) and (6.56) in (6.54), we obtain that

$$\begin{aligned}\mathbb{E}\left[V_t\right] \leq \left(1-\frac{\mu}{4}\gamma_{t-1}\right)\Big(&\mathbb{E}\left[\mathcal{L}_H(\boldsymbol{\theta_{t-1}})-\mathcal{L}_H^*\right]\\&+z\,\mathbb{E}\left[\left\|\boldsymbol{\xi_{t-1}}\right\|^2\right]+\kappa z\,\mathbb{E}\left[M_{t-1}^H\right]\Big)\\&+z\left(1-\beta_t\right)^2\left(\kappa+\frac{1}{n-f}\right)\frac{\sigma^2}{b}+z\left(1-\beta_t\right)\kappa\zeta\,.\end{aligned}$$

Recall that, by definition of V_t in (6.49), $\mathbb{E}\left[\mathcal{L}_H(\boldsymbol{\theta_{t-1}})-\mathcal{L}_H^*\right]+z\,\mathbb{E}\left[\left\|\boldsymbol{\xi_{t-1}}\right\|^2\right]+\kappa z\,\mathbb{E}\left[M_{t-1}^H\right]=\mathbb{E}\left[V_{t-1}\right]$. Therefore, the above can be written as

$$\begin{aligned}\mathbb{E}\left[V_t\right] \leq \left(1-\frac{\mu}{4}\gamma_{t-1}\right)&\mathbb{E}\left[V_{t-1}\right]\\&+z\left(1-\beta_t\right)^2\left(\kappa+\frac{1}{n-f}\right)\sigma^2+z\left(1-\beta_t\right)\kappa\zeta\,.\end{aligned}$$

Substituting $\beta_t = 1-144L\gamma_{t-1}$ and $z=\frac{1}{64L}$ in the above concludes the proof. □

Convergence Result

Note that the growth of the sequence $(\mathbb{E}\left[V_t\right])_{t\in[T+1]}$ presented in Theorem 6.2 corresponds to that of $(U_t)_{t\in[T+1]}$ in Lemma 3.1 with $U_t := \mathbb{E}\left[V_t\right]$ and $\alpha_1, \alpha_2, \alpha_3$ set as follows:

$$\alpha_1 := \frac{\mu}{4}\,,\ \alpha_2 := 324L\left(\kappa+\frac{1}{n-f}\right)\frac{\sigma^2}{b} \text{ and } \alpha_3 := \frac{9}{4}\kappa\zeta\,. \tag{6.57}$$

Upon observing that $\mathcal{L}_H\left(\boldsymbol{\theta_t}\right)-\mathcal{L}_H^* \leq V_t$ for each iteration t, owing to Theorem 6.2 above, we can thus obtain the sublinear convergence of Algorithm 6 in the strongly convex case similar to that of Algorithm 5. Essentially, upon combining the result of Theorem 6.2 with that of Lemma 3.1, we obtain the following corollary. For simplicity, we will state the corollary in terms of parameters α_2 and α_3 defined above in (6.57).

Corollary 6.2 *Suppose Assumptions 3.1, 3.2, 4.1, and 4.2. Let T be an even integer greater than 2. Consider T iterations of Algorithm 6 with batch-size $b \in [1, m)$, an (f, κ)-robust aggregation rule F, a sequence of learning rates $(\gamma_t)_{t\in[T]}$ defined by*

$$\gamma_t = \begin{cases}\gamma_o & ,\ t < T/2\\ \frac{8}{\mu(s_o+t-\frac{T}{2})} & ,\ t \geq T/2\end{cases}\,, \tag{6.58}$$

where γ_o and s_o are positive real values such that $\gamma_o = \frac{8}{\mu s_o} \leq \frac{1}{144L}$, and a sequence of momentum coefficients $(\beta_t)_{t\in[T]}$ defined by (6.47) with $c = 144L$. Then, the following holds true:

$$\mathbb{E}\left[\mathcal{L}_H\left(\boldsymbol{\theta}_{\mathbf{T+1}}\right) - \mathcal{L}_H^*\right] \leq \frac{\Delta_o e^{-\frac{\mu}{8}\gamma_o(T-2)}}{T^2} + \frac{16(\alpha_2\gamma_o + \alpha_3)(s_o-1)^2}{\mu T^2} + \frac{128\alpha_2}{\mu^2 T} + \frac{8\alpha_3}{\mu},$$

where Δ_o is a positive real value such that

$$4(s_o-1)^2\left(\mathcal{L}_H(\boldsymbol{\theta}_1) - \mathcal{L}_H^* + \frac{1}{64L}\left(\left(\kappa + \frac{1}{n-f}\right)\frac{\sigma^2}{b} + \kappa\zeta\right)\right) \leq \Delta_o\,,$$

and constants α_2, α_3 are defined in (6.57).

Proof Similarly to Corollary 3.2 and Theorem 4.2, we make use of Lemma 3.1 presented in Chap. 3. First, consider the sequence of learning rates $(\gamma_t)_{t\in[T]}$ defined in (6.58). As $\mu \leq L$ when Assumptions 4.1 and 4.2 hold true simultaneously, we have $\frac{1}{144L} \leq \frac{1}{\mu} \leq \frac{4}{\mu}$. This implies that setting $\alpha_1 \coloneqq \frac{\mu}{4}$, we have

$$\gamma_o = \frac{8}{\mu s_o} = \frac{2}{\alpha_1 s_o} \leq \frac{1}{144L} \leq \frac{4}{\mu} = \frac{1}{\alpha_1}\,.$$

From the above, we note that the sequence $(\gamma_t)_{t\in[T]}$ satisfies the condition stated in Lemma 3.1.

Second, recall from Theorem 6.2 that for $t \in [T]$, we have

$$\mathbb{E}\left[V_t\right] \leq \left(1 - \frac{\mu}{4}\gamma_{t-1}\right)\mathbb{E}\left[V_{t-1}\right] + 324L\left(\kappa + \frac{1}{n-f}\right)\frac{\sigma^2}{b}\,\gamma_{t-1}^2 + \frac{9}{4}\kappa\zeta\,\gamma_{t-1}\,.$$

Similar to Theorem 6.1, note that the above is true even for $t = T+1$, once we apply the definitions of local stochastic gradients and the corresponding momentum vectors in Algorithm 6 also to $t = T+1$. Let us denote $U_t \coloneqq \mathbb{E}\left[V_t\right]$. The above implies that, for all $t \in [T]$,

$$U_{t+1} \leq \left(1 - \frac{\mu}{4}\gamma_t\right)U_t + 324L\left(\kappa + \frac{1}{n-f}\right)\frac{\sigma^2}{b}\,\gamma_t^2 + \frac{9}{4}\kappa\zeta\,\gamma_t\,.$$

(continued)

Therefore, we note that both the sequences $(\gamma_t)_{t\in[T]}$ and $(U_t)_{t\in[T+1]}$ satisfy the condition stated in Lemma 3.1, with a value for α_1 as defined above, and for α_2, α_3 defined as

$$\alpha_2 = 324L\left(\kappa + \frac{1}{n-f}\right)\frac{\sigma^2}{b} \text{ and } \alpha_3 = \frac{9}{4}\kappa\zeta\,. \tag{6.59}$$

Lastly, owing to Lemma 3.1, we obtain that

$$\begin{aligned} U_{T+1} \leq (s_o-1)^2 U_1 \frac{4e^{-\alpha_1\gamma_o\frac{(T-2)}{2}}}{T^2} \\ + \frac{4(\alpha_2\gamma_o+\alpha_3)(s_o-1)^2}{\alpha_1 T^2} \\ + \frac{8\alpha_2}{\alpha_1^2 T} + \frac{2\alpha_3}{\alpha_1}\,. \end{aligned}$$

As $U_1 = \mathbb{E}[V_1]$, from the definition of V_t in (6.48), we have

$$\begin{aligned} U_1 = &\left(\mathcal{L}_H(\boldsymbol{\theta_1}) - \mathcal{L}_H^*\right) \\ &+ \frac{1}{64L}\left(\mathbb{E}\left[\left\|\boldsymbol{\xi_1}\right\|^2\right] + \kappa\frac{1}{|H|}\sum_{i\in H}\mathbb{E}\left[\left\|\mathbf{m_1^{(i)}} - \boldsymbol{m_1^{\mathbf{H}}}\right\|^2\right]\right)\,. \end{aligned}$$

Simply using Lemmas 6.3 and 6.6 in the above with $\beta_1 = 0$, we get

$$U_1 \leq \left(\mathcal{L}_H(\boldsymbol{\theta_1}) - \mathcal{L}_H^*\right) + \frac{1}{64L}\left(\frac{\sigma^2}{b(n-f)} + \kappa\left(\frac{\sigma^2}{b} + \zeta\right)\right) = \frac{\Delta_0}{(s_o-1)^2}\,.$$

Replacing α_1 in the above, and then noting that $\mathbb{E}\left[\mathcal{L}_H(\boldsymbol{\theta_{T+1}}) - \mathcal{L}_H^*\right] \leq U_{T+1} \coloneqq \mathbb{E}[V_{T+1}]$ (by definition of V_t in (6.48)), concludes the proof. □

Resilience of Algorithm 6

Similarly to the non-convex case, we use the above corollary to deduce the resilience properties of Robust DMHB. Recall that, in the case of strong convexity, the output model denoted by $\hat{\boldsymbol{\theta}}$ is taken to be equal to the last iterate of the algorithm, i.e., $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta_{T+1}}$. According to Corollary 6.1, upon executing T iterations of Algorithm 6, under the conditions stated in Corollary 6.2, if the server outputs $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta_{T+1}}$, despite

the presence of f adversarial nodes, we obtain that

$$\mathbb{E}\left[\mathcal{L}_H\left(\hat{\boldsymbol{\theta}}\right) - \mathcal{L}_H^*\right] \leq \frac{\Delta_o e^{-\frac{\mu}{8}\gamma_o(T-2)}}{T^2} + \frac{16(\alpha_2\gamma_o + \alpha_3)(s_o-1)^2}{\mu\, T^2} + \frac{128\alpha_2}{\mu^2\, T} + \frac{8\alpha_3}{\mu}\,.$$

Upon substituting the values of α_2 and α_3 from 6.57 in the above, and rearranging the terms, we have that, if the server outputs $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}_{\mathbf{T+1}}$, Algorithm 6 achieves (f, ε)-resilience where ε is smaller than

$$\frac{\Delta_o e^{-\frac{\mu}{8}\gamma_o(T-2)}}{T^2} + \frac{162\left(\kappa + \frac{1}{n-f}\right)K_{\mathcal{L}_H}\sigma^2}{b}\left(\frac{2\gamma_0\tilde{s_o}}{T^2} + \frac{1}{\mu T}\right) + \frac{9\kappa\zeta}{\mu}\left(\frac{\kappa\zeta\tilde{s_o}}{4\mu\, T^2} + 2\right),$$

with $K_{\mathcal{L}_H} \coloneqq \frac{L}{\mu}$ is the condition number of the loss function $\mathcal{L}_H$, and $\tilde{s_o} = (s_o - 1)^2$. In the general setting when there is non-zero gradient noise, i.e., $\sigma > 0$, by ignoring the higher-order (exponential and quadratic) terms in T, and the constants (including Δ_o), we obtain that

$$\varepsilon \in O_T\left(K_{\mathcal{L}_H}\left(\kappa + \frac{1}{n-f}\right)\frac{\sigma^2}{bT} + \kappa\zeta\right).$$

We summarize, in Table 6.2, the above training error for the aggregation rules described in Chaps. 4 and 5. We note some important differences between the training errors of Robust DMHB and Robust DMGD, in the following.

Comparisons with Robust DMGD

Let us denote by $\varepsilon^{\text{DMGD}}$ the training error achieved upon executing T iterations of the Robust DMGD algorithm (i.e., Algorithm 5), provided there are at most f adversarial nodes. Recall, from (4.36), that

$$\varepsilon^{\text{DMGD}} \in O_T\left(K_{\mathcal{L}_H}\frac{\sigma^2}{(n-f)bT} + \kappa\frac{\sigma^2}{b} + \kappa\zeta\right).$$

Upon comparing the above with the training error achievable using Robust DMHB, we make the following two critical observations:

Asymptotic error. Similarly to the non-convex case, the local gradient noise does not affect the asymptotic training error when the server updates the global model by using a robust averaging of nodes' local momentums. The order of the

Table 6.2 Robust DMHB is (f, ε)-resilient in the strongly convex case for the shown training error ε, after T iterations, with different robust aggregation rules, provided the conditions of Corollary 6.2 hold true. The training errors ε for all the aggregation rules, except CWTM, are obtained by assuming $\frac{f}{n} < \frac{1}{2}$. For CWTM$_f$, as shown in Sect. 4.3.2, we assume that $\frac{f}{n} \leq \frac{1}{2}\left(\frac{c}{1+c}\right)$ where $c > 0$ is an arbitrary real-valued constant

Aggregation rule	Breakdown point	Training error ε
CWMed	$\frac{1}{2}$	$O_{n,T}\left(\frac{K_{\mathcal{L}_H}\sigma^2}{bT} + \zeta\right)$
GeoMed	$\frac{1}{2}$	$O_{n,T}\left(\frac{K_{\mathcal{L}_H}\sigma^2}{bT} + \zeta\right)$
CWTM$_f$	$\frac{f+1}{n}$	$O_{n,T}\left(\left(\frac{f+1}{n-f}\right)\frac{K_{\mathcal{L}_H}\sigma^2}{bT} + \frac{f}{n}\zeta\right)$
Multi-Krum$_{f,1}$	$\frac{f+1}{n}$	$O_{n,T}\left(\frac{K_{\mathcal{L}_H}\sigma^2}{bT} + \zeta\right)$
[Any of the above] $\circ$ NNM	$\frac{f+1}{n}$	$O_{n,T}\left(\left(\frac{f+1}{n-f}\right)\frac{K_{\mathcal{L}_H}\sigma^2}{bT} + \frac{f}{n}\zeta\right)$

asymptotic error for Robust DMHB, i.e., $O_\star(\kappa\zeta)$, still matches the lower bound presented in Sect. 5.3, under the assumption of ζ-gradient dissimilarity.

Rate of Convergence. Similarly to the non-convex case, Robust DMHB converges slower than Robust DMGD. Specifically, given an (f, κ)-robust averaging rule F, provided that $\zeta = 0$, the corresponding Robust DMHB algorithm may require up to $\kappa(n - f) + 1$ times more iterations than its DMGD counterpart. In general, such a degradation of convergence rate is unavoidable due to the presence of adversarial nodes. This however does not significantly inflate the gradient complexity of the algorithm, specifically when f is small (see Table 6.2).

6.3 Chapter Notes

In this chapter, we have introduced the method of distributed momentum for reducing the gradient complexity of Robust DMGD, thereby obtaining an order-optimal robustness at a reduced computational cost. As we had seen in Chap. 5 that although the composition of a robust averaging with the pre-aggregation scheme of NNM yields an order-optimal training error, the resulting gradient complexity can be orders of magnitude higher than a traditional (non-robust) machine learning algorithm. Hence, Robust DMGD imposes a prohibitively high gradient complexity for achieving an order-optimal training error. We have shown that an efficient alternative to Robust DMGD is Robust DMHB, i.e., the robust variant of the *distributed mini-batch heavy-ball* method. Essentially, in Robust DMHB, the server uses the robust aggregation of local *gradient momentums* to update a model. This

method yields order-optimal training error in the presence of adversarial nodes, while imposing minimal overhead on gradient complexity (of honest nodes). We have shown that the incorporation of local gradient momentum diminishes the impact of gradient noise on the asymptotic convergence of the learning procedure. This holds true for both non-convex and strongly convex loss functions. For proving this result, we introduced a general technique of analyzing the convergence of an optimization algorithm, specifically the *Lyapunov analysis*. Further details on this proof technique can be found in [5]. In general, Lyapunov analysis is widely used for characterizing the stability of non-linear dynamical systems [12, 14]. In the specific context of robust machine learning, these proofs are inspired from our recent work [3, 4, 8].

The fundamental limitation of *memoryless* robust aggregation is due to their *permutation invariance property*. That is to say, the trajectory of the model parameter vector over the course of Robust DMGD, using a memoryless aggregation, remains invariant to arbitrary shuffling of the nodes across iterations. The use of *history* of past gradients breaks this permutation invariance property and has been shown to be effective for Byzantine resilience, e.g., see [1, 2]. Related adaptations using *Polyak's momentum* were proposed and analyzed under standard machine learning assumptions in [3, 8, 11]. The effectiveness of distributed momentum has also been demonstrated empirically [7, 13]. The key property of Polyak's momentum that makes it effective against adversarial nodes is variance reduction with *controllable bias*. Other variance-reduction techniques, specifically SAGA [6] and VR-MARINA [9], have also been shown to be effective [10, 15]. However, these variance-reduction techniques induce higher gradient complexity compared to Polyak's momentum.

References

1. Alistarh D, Allen-Zhu Z, Li J (2018) Byzantine stochastic gradient descent. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems, pp 4618–4628
2. Allen-Zhu Z, Ebrahimianghazani F, Li J, Alistarh D (2021) Byzantine-resilient non-convex stochastic gradient descent. In: International Conference on Learning Representations
3. Allouah Y, Farhadkhani S, Guerraoui R, Gupta N, Pinot R, Stephan J (2023) Fixing by mixing: a recipe for optimal byzantine ML under heterogeneity. In: Proceedings of the 26th International Conference on Artificial Intelligence and Statistics, vol. 206. Proceedings of Machine Learning Research. PMLR, pp 1232–1300
4. Allouah Y, Guerraoui R, Gupta N, Pinot R, Stephan J (2023) On the privacy-robustness-utility trilemma in distributed learning. In: Krause A, Brunskill E, Cho K, Engelhardt B, Sabato S, Scarlett J (eds) International Conference on Machine Learning, ICML 2023, 23–29 July 2023, Honolulu, Hawaii, USA, vol. 202. Proceedings of Machine Learning Research. PMLR, pp 569–626
5. Bottou L (1999) On-line learning and stochastic approximations. In: Saad D (ed) On-Line learning in neural networks. Publications of the Newton Institute. Cambridge University Press, Cambridge, pp 9–42

6. Defazio A, Bach F, Lacoste-Julien S (2014) SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In: Advances in Neural Information Processing Systems, p 27
7. El Mhamdi EM, Guerraoui R, Rouault S (2021) Distributed momentum for byzantine-resilient stochastic gradient descent. In: 9th International Conference on Learning Representations, ICLR 2021, Vienna, Austria, May 4–8. OpenReview.net
8. Farhadkhani S, Guerraoui R, Gupta N, Pinot R, Stephan J (2022) Byzantine machine learning made easy by resilient averaging of momentums. In: International Conference on Machine Learning, ICML 2022, 17–23 July 2022, Baltimore, Maryland, USA, vol. 162. Proceedings of Machine Learning Research. PMLR, pp 6246–6283
9. Gorbunov E, Burlachenko KP, Li Z, Richtárik P (2021) MARINA: Faster non-convex distributed learning with compression. In: International Conference on Machine Learning. PMLR, pp 3788–3798
10. Gorbunov E, Horváth S, Richtárik P, Gidel G (2023) Variance reduction is an antidote to byzantines: better rates, weaker assumptions and communication compression as a cherry on the top. In: The 11th International Conference on Learning Representations
11. Karimireddy SP, He L, Jaggi M (2021) Learning from history for byzantine robust optimization. In: International Conference on Machine Learning, vol 139. Proceedings of Machine Learning Research
12. Khalil HK (2002) Nonlinear systems, 3rd edn. Prentice-Hall, Upper Saddle River, NJ
13. Rouault SLA (2022) Practical Byzantine-resilient Stochastic Gradient Descent. Tech. rep. EPFL
14. Vidyasagar M (2002) Nonlinear systems analysis. SIAM, Philadelphia
15. Wu Z, Ling Q, Chen T, Giannakis GB (2020) Federated variance-reduced stochastic gradient descent with robustness to byzantine attacks. IEEE Trans Signal Process 68:4583–4596

Appendix A
General Lemmas

This appendix presents some basic inequalities pertaining to Lipschitz smooth functions and functions that also satisfy the Polyak–Lojasiewicz (PL) inequality. These results are fairly standard in the optimization community and can be found in most optimization textbooks, e.g., see [2, 3].

A.1 Lipschitz Smoothness Inequality

Lemma A.1 *Let $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$ be a loss function satisfying Assumption 3.1. For any $\boldsymbol{\theta}$, $\boldsymbol{\theta}' \in \mathbb{R}^d$, we have*

$$\mathcal{L}(\boldsymbol{\theta}) \leq \mathcal{L}(\boldsymbol{\theta}') + \left\langle \nabla \mathcal{L}(\boldsymbol{\theta}'),\, \boldsymbol{\theta} - \boldsymbol{\theta}' \right\rangle + \frac{L}{2} \left\| \boldsymbol{\theta} - \boldsymbol{\theta}' \right\|^2 .$$

Proof Let $\boldsymbol{\theta}$, $\boldsymbol{\theta}' \in \mathbb{R}^d$. Recall that, by definition of the gradient $\nabla \mathcal{L}$,

$$\mathcal{L}(\boldsymbol{\theta}) - \mathcal{L}(\boldsymbol{\theta}') = \int_0^1 \left\langle \nabla \mathcal{L}(\boldsymbol{\theta}' + t(\boldsymbol{\theta} - \boldsymbol{\theta}')),\, \boldsymbol{\theta} - \boldsymbol{\theta}' \right\rangle dt \,.$$

Using the above, we get

$$\begin{aligned}
&\mathcal{L}(\boldsymbol{\theta}) - \mathcal{L}(\boldsymbol{\theta}') - \left\langle \nabla \mathcal{L}(\boldsymbol{\theta}'),\, \boldsymbol{\theta} - \boldsymbol{\theta}' \right\rangle \\
&= \int_0^1 \left\langle \nabla \mathcal{L}(\boldsymbol{\theta}' + t(\boldsymbol{\theta} - \boldsymbol{\theta}')),\, \boldsymbol{\theta} - \boldsymbol{\theta}' \right\rangle - \left\langle \nabla \mathcal{L}(\boldsymbol{\theta}'),\, \boldsymbol{\theta} - \boldsymbol{\theta}' \right\rangle dt \\
&= \int_0^1 \left\langle \nabla \mathcal{L}(\boldsymbol{\theta}' + t(\boldsymbol{\theta} - \boldsymbol{\theta}')) - \nabla \mathcal{L}(\boldsymbol{\theta}'),\, \boldsymbol{\theta} - \boldsymbol{\theta}' \right\rangle dt .
\end{aligned}$$

R. Guerraoui et al., *Robust Machine Learning*, Machine Learning: Foundations, Methodologies, and Applications, https://doi.org/10.1007/978-981-97-0688-4

Then by monotony of the integration and Cauchy–Schwarz inequality, we get

$$\leq \int_0^1 |\left\langle \nabla \mathcal{L}(\boldsymbol{\theta}' + t(\boldsymbol{\theta} - \boldsymbol{\theta}')) - \nabla \mathcal{L}(\boldsymbol{\theta}'),\, \boldsymbol{\theta} - \boldsymbol{\theta}' \right\rangle| dt.$$

$$\leq \int_0^1 \left\| \nabla \mathcal{L}(\boldsymbol{\theta}' + t(\boldsymbol{\theta} - \boldsymbol{\theta}')) - \nabla \mathcal{L}(\boldsymbol{\theta}') \right\| \left\| \boldsymbol{\theta} - \boldsymbol{\theta}' \right\| dt.$$

Finally, by using Assumption 3.1, we get the following:

$$\leq \int_0^1 L \left\| t(\boldsymbol{\theta} - \boldsymbol{\theta}') \right\| \left\| \boldsymbol{\theta} - \boldsymbol{\theta}' \right\| dt = \frac{L}{2} \left\| \boldsymbol{\theta} - \boldsymbol{\theta}' \right\|^2.$$

The above inequality concludes the proof. □

A.2 Polyak–Lojasiewicz Inequality Under Lipschitz Smoothness

We show in the following that if a function $\mathcal{L}$ satisfies both the μ-PL inequality (see Assumption 3.3) and the condition of Lipschitz smoothness with coefficient L (see Assumption 3.1), then $\mu \leq L$.

As $\mathcal{L}$ is L-Lipschitz smooth, due to Lipschitz inequality (Lemma A.1), for any $\boldsymbol{\theta}, \boldsymbol{\theta}' \in \Theta$, we have

$$\mathcal{L}(\boldsymbol{\theta}') \leq \mathcal{L}(\boldsymbol{\theta}) + \left\langle \nabla \mathcal{L}(\boldsymbol{\theta}),\, \boldsymbol{\theta}' - \boldsymbol{\theta} \right\rangle + \frac{L}{2} \left\| \boldsymbol{\theta}' - \boldsymbol{\theta} \right\|^2.$$

Substituting $\boldsymbol{\theta}' = \boldsymbol{\theta} - \frac{1}{L} \nabla \mathcal{L}(\boldsymbol{\theta})$, we obtain that

$$\mathcal{L}(\boldsymbol{\theta}') \leq \mathcal{L}(\boldsymbol{\theta}) - \frac{1}{2L} \left\| \nabla \mathcal{L}(\boldsymbol{\theta}) \right\|^2.$$

Recall that $\mathcal{L}^*$ denotes the minimum value of $\mathcal{L}(\boldsymbol{\theta})$. Subtracting $\mathcal{L}^*$ on both sides and noting that $\mathcal{L}(\boldsymbol{\theta}') - \mathcal{L}^* \geq 0$, we obtain that

$$\frac{1}{2} \left\| \nabla \mathcal{L}(\boldsymbol{\theta}) \right\|^2 \leq L \left(\mathcal{L}(\boldsymbol{\theta}) - \mathcal{L}^* \right). \tag{A.1}$$

The above is in line with the μ-PL inequality (i.e., Assumption 3.3) only if $\mu \leq L$.

Appendix B
General Exponential Convergence

This appendix presents a general result on the exponential convergence of a large class of sequences. The result is used for deriving the convergence results, for the iterative algorithms presented in the manuscript, in the strongly convex case. The original form of this result can be found in [4].

Lemma B.1 *Let $\alpha_1, \alpha_2, \alpha_3$ be real values such that $\alpha_1 > 0$, $\alpha_2 \geq 0$, and $\alpha_3 \geq 0$. Let $T > 2$ be an even number. Consider two positive real-valued sequences $(\gamma_1, \ldots, \gamma_T)$ and $(U_1, \ldots, U_{T+1})$ such that for all $t \in [T]$,*

$$\gamma_t = \begin{cases} \gamma_o & , \ t < T/2 \\ \frac{2}{\alpha_1 \left(s_o + t - \frac{T}{2}\right)} & , \ t \geq T/2 \end{cases}, \quad \text{and} \quad U_{t+1} \leq (1 - \alpha_1 \gamma_t)\, U_t + \alpha_2 \gamma_t^2 + \alpha_3 \gamma_t \,,$$

where γ_o and s_o are positive real values such that $\gamma_o = \frac{2}{\alpha_1 s_o} < \frac{1}{\alpha_1}$. Then,

$$U_{T+1} \ \leq \ (s_o - 1)^2 U_1 \, \frac{4 e^{-\frac{\alpha_1 \gamma_o}{2}(T-2)}}{T^2} \ + \ \frac{4(\alpha_2 \gamma_o + \alpha_3)(s_o - 1)^2}{\alpha_1 \, T^2} \ + \ \frac{8\alpha_2}{\alpha_1^2 \, T} \ + \ \frac{2\alpha_3}{\alpha_1} \,.$$

Proof We consider the two cases for t separately, i.e., first we consider the case when $t \in \left\{1, \ldots, \frac{T}{2} - 1\right\}$, and then we consider the case when $t \in \left\{\frac{T}{2}, \ldots, T\right\}$. We conclude by combining the results obtained in the two cases.

Case i. Consider an arbitrary $t \in \left\{1, \ldots, \frac{T}{2} - 1\right\}$. In this particular case, $\gamma_t = \gamma_o$. Thus, we have

$$U_{t+1} \leq (1 - \alpha_1 \gamma_o)\, U_t + \alpha_2 \gamma_o^2 + \alpha_3 \gamma_o \,.$$

R. Guerraoui et al., *Robust Machine Learning*, Machine Learning: Foundations, Methodologies, and Applications, https://doi.org/10.1007/978-981-97-0688-4

Therefore, by recursion, we obtain that

$$U_{\frac{T}{2}} \leq (1-\alpha_1\gamma_o)^{\frac{T}{2}-1} U_1 + (\alpha_2\gamma_o^2 + \alpha_3\gamma_o) \sum_{t=1}^{\frac{T}{2}-1} (1-\alpha_1\gamma_o)^{t-1} . \tag{B.1}$$

From (B.1), as $1-\alpha_1\gamma_o < 1$, we obtain that

$$U_{\frac{T}{2}} \leq (1-\alpha_1\gamma_o)^{\frac{T}{2}-1} U_1 + \frac{(\alpha_2\gamma_o+\alpha_3)\gamma_o}{1-(1-\alpha_1\gamma_o)} = (1-\alpha_1\gamma_o)^{\frac{T}{2}-1} U_1 + \frac{\alpha_2\gamma_o+\alpha_3}{\alpha_1} .$$

As, for any positive real value y and $x < 1$, we have $(1-x)^y \leq e^{-xy}$. From above, we obtain that

$$U_{\frac{T}{2}} \leq U_1 e^{-\alpha_1\gamma_o(\frac{T}{2}-1)} + \frac{\alpha_2\gamma_o+\alpha_3}{\alpha_1} = U_1 e^{-\alpha_1\gamma_o\frac{(T-2)}{2}} + \frac{\alpha_2\gamma_o+\alpha_3}{\alpha_1} . \tag{B.2}$$

Case ii. Consider an arbitrary $t \in \{\frac{T}{2}, \ldots, T\}$. In this particular case, $\gamma_t = \frac{2}{\alpha_1(s_o+t-\frac{T}{2})} \leq \gamma_o$. Let $s = s_o - \frac{T}{2}$, thereby $\gamma_t = \frac{2}{\alpha_1(s+t)}$. Recall that, by hypothesis, we have

$$U_{t+1} \leq (1-\alpha_1\gamma_t)\ U_t + \alpha_2\gamma_t^2 + \alpha_3\gamma_t .$$

Thus, replacing γ_t by $\frac{2}{\alpha_1(s+t)}$, we have

$$U_{t+1} \leq \left(1 - \frac{2}{s+t}\right) U_t + \frac{4\alpha_2}{\alpha_1^2(s+t)^2} + \frac{2\alpha_3}{\alpha_1(s+t)}.$$

Upon multiplying both sides in the above by $(s+t)^2$, we obtain that

$$(s+t)^2 U_{t+1} \leq \Big((s+t)^2 - 2(s+t)\Big)\ U_t + \frac{4\alpha_2}{\alpha_1^2} + \frac{2\alpha_3}{\alpha_1}(s+t) .$$

As $U_t \geq 0$, we have $\big((s+t)^2 - 2(s+t)\big)\ U_t \leq (s+t-1)^2\, U_t$. Hence, we also have

$$(s+t)^2 U_{t+1} \leq (s+t-1)^2\, U_t + \frac{4\alpha_2}{\alpha_1^2} + \frac{2\alpha_3}{\alpha_1}(s+t) .$$

As the above holds true for any $t \in \{\frac{T}{2}, \ldots, T\}$, taking the summation for all these values of t and reorganizing the terms, we obtain that

$$(s+T)^2\, U_{T+1} \leq (s + \frac{T}{2} - 1)^2\, U_{\frac{T}{2}} + \frac{4\alpha_2}{\alpha_1^2}\,\frac{T}{2} + \frac{2\alpha_3}{\alpha_1}(s+T)\frac{T}{2} .$$

Dividing both the sides by $(s+T)^2$, we obtain that

$$U_{T+1} \leq \frac{(s+\frac{T}{2}-1)^2}{(s+T)^2} U_{\frac{T}{2}} + \frac{2\alpha_2 T}{\alpha_1^2 (s+T)^2} + \frac{\alpha_3}{\alpha_1} \left(\frac{T}{s+T} \right).$$

Now substituting $s = s_o - \frac{T}{2}$, we get

$$U_{T+1} \leq \frac{(s_o-1)^2}{(s_o+\frac{T}{2})^2} U_{\frac{T}{2}} + \frac{2\alpha_2}{\alpha_1^2 (s_o+\frac{T}{2})^2} + \frac{\alpha_3}{\alpha_1} \left(\frac{T}{s_o+\frac{T}{2}} \right).$$

As $\frac{T}{2} \leq s_o + \frac{T}{2}$, from above we get

$$U_{T+1} \leq \frac{4(s_o-1)^2}{T^2} U_{\frac{T}{2}} + \frac{8\alpha_2}{\alpha_1^2 T} + \frac{2\alpha_3}{\alpha_1}.$$

Substituting $U_{\frac{T}{2}}$ in the above from (B.2), we obtain that

$$U_{T+1} \leq 4(s_o-1)^2 U_1 \frac{4e^{-\alpha_1 \gamma_o \frac{(T-2)}{2}}}{T^2} + \frac{4(\alpha_2\gamma_o+\alpha_3)(s_o-1)^2}{\alpha_1 T^2} + \frac{8\alpha_2}{\alpha_1^2 T} + \frac{2\alpha_3}{\alpha_1}.$$

This concludes the proof. □

Appendix C
Derivation of Robust Coefficients

This appendix presents the derivations for the robustness coefficients of the robust aggregation rules presented in Chap. 4. But first, we present below a fundamental limitation on robustness of any aggregation rule with respect to the condition of (q, κ)-robust averaging expressed in Definition 4.3. These results are merely a rewriting of the constants computations that can be found in [1].

C.1 Limitation on Robust Averaging

Considering the scalar case, i.e., assuming $d = 1$, we obtain the following necessary condition (lower bound) for (q, κ)-robust averaging. This lower bound, denoted by κ_o, on the robustness coefficient in the lemma below is tight, i.e., there exist aggregation rules (e.g., CWTM) that are indeed (q, κ)-robust for $\kappa \in O_\star(\kappa_o)$.[1]

Lemma C.1 *For any integer $q < n$, an aggregation rule $F : (\mathbb{R})^n \to \mathbb{R}$ is an (q, κ)-robust averaging only if $q < \frac{n}{2}$ and $\kappa \geq \frac{q}{n-2q}$.*

Proof The necessity for $q < n/2$ is easy to show. Also, as $\kappa \geq 0$ by definition of robust averaging, the assertion is trivially true when $q = 0$. Hereafter, we will assume that $q > 0$. We employ proof by contradiction. Let us assume that the following assertion holds true:

Assertion A: For $0 < q < n$ and $\kappa < \frac{q}{n-2q}$, there exists an aggregation rule $F : (\mathbb{R})^n \to \mathbb{R}$ that is (q, κ)-strong averaging.

[1] We write $\phi(a) \in O_\star(\psi(a))$ if there exists a constant $c \in \mathbb{R}^+$ such that $\phi(a) \leq c\psi(a)$ for all $a \in \mathbb{R}^+$.

R. Guerraoui et al., *Robust Machine Learning*, Machine Learning: Foundations, Methodologies, and Applications, https://doi.org/10.1007/978-981-97-0688-4

Consider n real values $v_1, \ldots, v_n$ such that $v_1 = \ldots = v_{n-q} = 0$ and $v_{n-q+1} = \ldots = v_n = 1$. Consider a set $S_0 = \{1, \ldots, n-q\}$. Since $|S_0| = n-q$, by **Assertion A** and Definition 4.3 of robust averaging, we have

$$\left|F(v_1, \ldots, v_n) - \bar{v}_{S_0}\right|^2 \leq \kappa \frac{1}{n-q} \sum_{i \in S_0} \left|v_i - \bar{v}_{S_0}\right|^2 = 0\,,$$

where $\bar{v}_{S_0} \coloneqq \frac{1}{|S_0|} \sum_{i \in S_0} v_i = 0$. Thus, $F(v_1, \ldots, v_n) = \bar{v}_{S_0} = 0$. Next, we consider another set $S_1 = \{q+1, \ldots, n\}$. As $\bar{v}_{S_1} = \frac{q}{n-q}$, we obtain that

$$\left|F(v_1, \ldots, v_n) - \bar{v}_{S_1}\right|^2 = \left|0 - \bar{v}_{S_1}\right|^2 = \left(\frac{q}{n-q}\right)^2 . \tag{C.1}$$

Recall that **Assertion A** implies that F is (q, κ)-strong averaging with $\kappa < \frac{q}{n-2q}$. Thus, as $|S_1| = n-q$, by Definition 4.3, we have

$$\left|F(v_1, \ldots, v_n) - \bar{v}_{S_1}\right|^2 \leq \kappa \frac{1}{n-q} \sum_{i \in S_1} \left|v_i - \bar{v}_{S_1}\right|^2 = \kappa \frac{q(n-2q)}{(n-q)^2} < \left(\frac{q}{n-q}\right)^2 .$$

As $q > 0$, the above contradicts (C.1). Hence, **Assertion A** is false, i.e., we cannot have $\kappa < \frac{q}{n-2q}$ when $q > 0$. This concludes the proof. □

In the following, we prove that the aggregation rules presented in Chap. 4 are robust averaging. In the process, we also derive upper bounds on their respective robustness coefficients.

C.2 Coordinate-Wise Trimmed Mean

We first observe that for a coordinate-wise aggregation rule F, if each coordinate of F satisfies the criterion of robust averaging, then F yields robust averaging. Specifically, we have the following lemma. Recall that for any vector $\mathbf{v}$ we denote its k-th coordinate by $[\mathbf{v}]_k$.

Lemma C.2 *Let $q < n/2$ and $\kappa \geq 0$. Suppose that aggregation $F : (\mathbb{R}^d)^n \to \mathbb{R}^d$ is a coordinate-wise function, i.e., there exist d real-valued functions $F_1, \ldots, F_d \colon \mathbb{R}^n \to \mathbb{R}$ such that for all $\mathbf{v_1}, \ldots, \mathbf{v_n} \in \mathbb{R}^d$, $[F(\mathbf{v_1}, \ldots, \mathbf{v_n})]_k = F_k([\mathbf{v_1}]_k, \ldots, [\mathbf{v_n}]_k)$. If for every $k \in [d]$, F_k is (q, κ)-robust averaging, then F is (q, κ)-robust averaging.*

Proof Consider n arbitrary vectors in $\mathbb{R}^d$, $\mathbf{v_1}, \ldots, \mathbf{v_n}$, and an arbitrary set $S \subseteq [n]$ such that $|S| = n - q$. Assume that for each $k \in [d]$, F_k is (q, κ)-robust averaging. Since F is a coordinate-wise aggregation function, we have

$$\|F(\mathbf{v_1}, \ldots, \mathbf{v_n}) - \overline{\mathbf{v}}_\mathbf{S}\|^2 = \sum_{k=1}^{d} |F_k([\mathbf{v_1}]_k, \ldots, [\mathbf{v_n}]_k) - [\overline{\mathbf{v}}_\mathbf{S}]_k|^2 . \tag{C.2}$$

Since for each $k \in [d]$, F_k is a (q, κ)-robust averaging, we have

$$\sum_{k=1}^{d} |F_k([\mathbf{v_1}]_k, \ldots, [\mathbf{v_n}]_k) - [\overline{\mathbf{v}}_\mathbf{S}]_k|^2 \leq \sum_{k=1}^{d} \frac{\kappa}{|S|} \sum_{i \in S} |[\mathbf{v_i}]_k - [\overline{\mathbf{v}}_\mathbf{S}]_k|^2$$

$$= \frac{\kappa}{|S|} \sum_{i \in S} \sum_{k=1}^{d} |[\mathbf{v_i}]_k - [\overline{\mathbf{v}}_\mathbf{S}]_k|^2 = \frac{\kappa}{|S|} \sum_{i \in S} \|\mathbf{v_i} - \overline{\mathbf{v}}_\mathbf{S}\|^2 .$$

Substituting from above in (C.2) concludes the proof. □

We now show an important property in Lemma C.3 below on the sorting of real values that proves essential in obtaining the (q, κ)-robustness guarantee for CWTM.

Lemma C.3 *Let $q < n/2$. Consider n real values $v_1, \ldots, v_n$ such that $v_1 \leq \ldots \leq v_n$. Let $S \subseteq [n]$ of size $|S| = n - q$, and $I \coloneqq \{q + 1, \ldots, n - q\}$. We have that*

$$\sum_{i \in I \setminus S} |v_i - \overline{v}_S|^2 \leq \sum_{i \in S \setminus I} |v_i - \overline{v}_S|^2 .$$

Proof First, note that, as $|I| = n - 2q$ and $|S| = n - q$, $|I \setminus S| \leq q$ and $|S \setminus I| \geq q$. Thus, $|I \setminus S| \leq |S \setminus I|$. To obtain the expected result, we show that there exists an injective mapping $\phi \colon I \setminus S \to S \setminus I$ such that

$$\forall i \in I \setminus S, \ \ |v_i - \overline{v}_S| \leq \left|v_{\phi(i)} - \overline{v}_S\right| . \tag{C.3}$$

As $|I \setminus S| \leq |S \setminus I|$, we have $\sum_{i \in I \setminus S} \left|v_{\phi(i)} - \overline{v}_S\right|^2 \leq \sum_{i \in S \setminus I} |v_i - \overline{v}_S|^2$. Thus, if (C.3) holds true, Lemma C.3 also holds true.

We denote by S^{C} the complement of S in $[n]$, i.e., $S^{\mathrm{C}} = [n] \setminus S$. Therefore, $\left|S^{\mathrm{C}}\right| = q$ and $I \setminus S = I \cap S^{\mathrm{C}}$. We denote $I^+ \coloneqq \{n - q + 1, \ldots, n\}$ and $I^- \coloneqq \{1, \ldots, q\}$ to be the indices of values that are larger (or equal to) and smaller (or equal to) than the values in I, respectively. Let $\left|I \cap S^{\mathrm{C}}\right| = r$. As $S = [n] \setminus S^{\mathrm{C}}$, we obtain that

$$\left|I^+ \cap S\right| = q - \left|I^+ \cap S^{\mathrm{C}}\right| \geq q - \left|S^{\mathrm{C}} \setminus (I \cap S^{\mathrm{C}})\right| = q - (q - r) = r. \tag{C.4}$$

Similarly, we obtain that

$$\left|I^- \cap S\right| \geq r. \tag{C.5}$$

Recall that $r = \left|I \cap S^{\mathrm{c}}\right|$ where $I \setminus S = I \cap S^{\mathrm{c}}$. Therefore, due to (C.4) and (C.5), there exists an injective mapping from $I \setminus S$ to $(I^- \cap S) \times (I^+ \cap S)$. Let ψ be such a mapping. For each $i \in I \setminus S$, we denote $\psi(i) := (\psi^-(i), \psi^+(i))$, where $(\psi^-(i), \psi^+(i)) \in (I^- \cap S) \times (I^+ \cap S)$. Consider an arbitrary $i \in I \setminus S$. By definition of I^- and I^+, we have

$$v_{\psi^-(i)} \leq v_i \leq v_{\psi^+(i)}.$$

Therefore, for any real value u,

$$|v_i - u| \leq \max\left\{\left|v_{\psi^+(i)} - u\right|, \left|v_{\psi^-(i)} - u\right|\right\}.$$

Setting ϕ to be defined as $\phi(i) = \operatorname{argmax}_{j \in \{\psi^+(i), \psi^-(i)\}} \left|v_j - \overline{v}_S\right|$ for all $i \in I \setminus S$, we obtain that (C.3) holds true, hence concluding the proof. □

Using Lemmas C.2 and C.3, we can now show that CWTM_q is a (q, κ)-robust averaging below.

Proposition C.1 *Let $n \in \mathbb{N}^*$ and $q < n/2$. CWTM_q is (q, κ)-robust with*

$$\kappa = \frac{6q}{n-2q}\left(1 + \frac{q}{n-2q}\right).$$

Proof First, note that CWTM_q is a coordinate-wise aggregation, in the precise sense as defined in Lemma C.2. Thus, due to Lemma C.2, it suffices to show that Trimmed Mean is (f, κ)-robust in the scalar domain, i.e., when $d = 1$.

Consider n arbitrary values $v_1, \ldots, v_n \in \mathbb{R}$. Without loss of generality, assume that $v_1 \leq \ldots \leq v_n$. We denote by $I := \{q+1, \ldots, n-q\}$, and let $S \subseteq [n]$ be an arbitrary set of size $n - q$. Recall that by definition of trimmed mean, we have $\mathrm{TM}_q(v_1, \ldots, v_n) = \frac{1}{n-2q}\sum_{i \in I} v_i$. Therefore,

$$\left|\mathrm{TM}_q(v_1, \ldots, v_n) - \overline{v}_S\right|^2 = \left|\frac{1}{n-2q}\sum_{i \in I} v_i - \overline{v}_S\right|^2 = \left|\frac{1}{n-2q}\sum_{i \in I}(v_i - \overline{v}_S)\right|^2$$

$$= \left|\frac{1}{n-2q}\sum_{i \in I}(v_i - \overline{v}_S) - \frac{1}{n-2q}\sum_{i \in S}(v_i - \overline{v}_S)\right|^2 \qquad \left(\because \sum_{i \in S}(v_i - \overline{v}_S) = 0\right)$$

$$= \frac{1}{(n-2q)^2}\left|\sum_{i \in I \setminus S}(v_i - \overline{v}_S) - \sum_{i \in S \setminus I}(v_i - \overline{v}_S)\right|^2.$$

Using Jensen's inequality above, we obtain that

$$\left|\mathrm{TM}_q(v_1,\ldots,v_n)-\overline{v}_S\right|^2 \leq \frac{|I\setminus S|+|S\setminus I|}{(n-2q)^2}\left(\sum_{i\in I\setminus S}|v_i-\overline{v}_S|^2+\sum_{i\in S\setminus I}|v_i-\overline{v}_S|^2\right).$$

Note that $|I\setminus S|=|I\cup S|-|S|\leq n-(n-q)=q$. Similarly, $|S\setminus I|=|I\cup S|-|I|\leq n-(n-2q)=2q$. Accordingly, $|I\setminus S|+|S\setminus I|\leq 3q$. Therefore, we have

$$\left|\mathrm{TM}_q(v_1,\ldots,v_n)-\overline{v}_S\right|^2 \leq \frac{3q}{(n-2q)^2}\left(\sum_{i\in I\setminus S}|v_i-\overline{v}_S|^2+\sum_{i\in S\setminus I}|v_i-\overline{v}_S|^2\right).$$

Recall from Lemma C.3 that $\sum_{i\in I\setminus S}|v_i-\overline{v}_S|^2\leq\sum_{i\in S\setminus I}|v_i-\overline{v}_S|^2$. Using this fact above, we obtain that

$$\begin{aligned}\left|\mathrm{TM}_q(v_1,\ldots,v_n)-\overline{v}_S\right|^2 &\leq \frac{6q}{(n-2q)^2}\sum_{i\in S\setminus I}|v_i-\overline{v}_S|^2\\ &\leq \frac{6q}{(n-2q)^2}\sum_{i\in S}|v_i-\overline{v}_S|^2.\end{aligned}$$

Finally, as $\frac{q}{(n-2q)^2}=\frac{q(n-q)}{(n-2q)^2}\left(\frac{1}{n-q}\right)=\left(\frac{q}{n-2q}+\left(\frac{q}{n-2q}\right)^2\right)\frac{1}{n-q}$, we obtain that

$$\left|\mathrm{TM}_q(v_1,\ldots,v_n)-\overline{v}_S\right|^2 \leq 6\left(\frac{q}{n-2q}+\left(\frac{q}{n-2q}\right)^2\right)\frac{1}{|S|}\sum_{i\in S}|v_i-\overline{v}_S|^2. \tag{C.6}$$

The above concludes the proof. □

C.3 Krum

Given the input vectors $\mathbf{v_1},\ldots,\mathbf{v_n}\in\mathbb{R}^d$, Krum outputs a vector from the set of input vectors that is nearest to its neighbors. Specifically, the output of Multi-Krum$_{q,1}$, or simply Krum$_q$, is a vector that is closest to its $n-q$ neighbors. We denote by N_j the index set of $n-q$ nearest neighbors of $\mathbf{v_j}$ in $\{\mathbf{v_1},\ldots,\mathbf{v_n}\}$,

where the ties can be broken arbitrarily. $\mathrm{Krum}_q(\mathbf{v_1}, \ldots, \mathbf{v_n})$ outputs a vector $\mathbf{v_{k^*}}$ such that

$$k^* \in \operatorname*{argmin}_{j \in [n]} \sum_{i \in N_j} \left\| \mathbf{v_j} - \mathbf{v_i} \right\|^2 .$$

Proposition C.2 *Let $n \in \mathbb{N}^*$ and $q < \frac{n}{2}$. Then, Krum_q is (q, κ)-robust with $\kappa = 6\left(1 + \frac{q}{n-2q}\right)$.*

Proof Let $n \in \mathbb{N}^*$, $q < \frac{n}{2}$, and $\mathbf{v_1}, \ldots, \mathbf{v_n} \in \mathbb{R}^d$. Consider any subset $S \subseteq [n]$ of size $|S| = n - q$. In the following, for every $j \in [n]$, we denote by N_j the set of indices of $n - q$ nearest neighbors of $\mathbf{v_j}$ in $\{\mathbf{v_1}, \ldots, \mathbf{v_n}\}$, with ties arbitrarily broken. Observe that this implies, for all $j \in [n]$, that

$$\sum_{i \in N_j} \left\| \mathbf{v_j} - \mathbf{v_i} \right\|^2 \leq \sum_{i \in S} \left\| \mathbf{v_j} - \mathbf{v_i} \right\|^2 . \tag{C.7}$$

Let $k^* \in [n]$ be the index chosen by Krum_q, i.e., $k^* \in \operatorname{argmin}_{j \in [n]} \sum_{i \in N_j} \left\| \mathbf{v_j} - \mathbf{v_i} \right\|^2$. Therefore, due to (C.7), we have

$$\begin{aligned} \sum_{i \in N_{k^*}} \left\| \mathbf{v_{k^*}} - \mathbf{v_i} \right\|^2 &= \min_{j \in [n]} \sum_{i \in N_j} \left\| \mathbf{v_j} - \mathbf{v_i} \right\|^2 \leq \min_{j \in S} \sum_{i \in N_j} \left\| \mathbf{v_j} - \mathbf{v_i} \right\|^2 \\ &\leq \frac{1}{|S|} \sum_{j \in S} \sum_{i \in N_j} \left\| \mathbf{v_j} - \mathbf{v_i} \right\|^2 \leq \frac{1}{|S|} \sum_{j \in S} \sum_{i \in S} \left\| \mathbf{v_j} - \mathbf{v_i} \right\|^2 . \end{aligned} \tag{C.8}$$

Note that

$$\begin{aligned} \sum_{j \in S} \sum_{i \in S} \left\| \mathbf{v_j} - \mathbf{v_i} \right\|^2 &= \sum_{i,j \in S} \left\| \mathbf{v_j} - \overline{\mathbf{v}}_\mathbf{S} - (\mathbf{v_i} - \overline{\mathbf{v}}_\mathbf{S}) \right\|^2 \\ &= \sum_{i,j \in S} \left(\left\| \mathbf{v_j} - \overline{\mathbf{v}}_\mathbf{S} \right\|^2 + \left\| \mathbf{v_i} - \overline{\mathbf{v}}_\mathbf{S} \right\|^2 - 2 \left\langle \mathbf{v_j} - \overline{\mathbf{v}}_\mathbf{S},\ \mathbf{v_i} - \overline{\mathbf{v}}_\mathbf{S} \right\rangle \right) \\ &= 2\,|S| \sum_{i \in S} \left\| \mathbf{v_i} - \overline{\mathbf{v}}_\mathbf{S} \right\|^2 - 2 \sum_{i \in S} \left(\sum_{j \in S} \left\langle \mathbf{v_j} - \overline{\mathbf{v}}_\mathbf{S},\ \mathbf{v_i} - \overline{\mathbf{v}}_\mathbf{S} \right\rangle \right) . \end{aligned}$$

In the above, for all $i \in S$, we have $\sum_{j \in S} \left\langle \mathbf{v_j} - \overline{\mathbf{v}}_\mathbf{S},\ \mathbf{v_i} - \overline{\mathbf{v}}_\mathbf{S} \right\rangle = \left\langle \mathbf{0},\ \mathbf{v_i} - \overline{\mathbf{v}}_\mathbf{S} \right\rangle = 0$. Therefore,

$$\sum_{j \in S} \sum_{i \in S} \left\| \mathbf{v_j} - \mathbf{v_i} \right\|^2 \leq 2\,|S| \sum_{i \in S} \left\| \mathbf{v_i} - \overline{\mathbf{v}}_\mathbf{S} \right\|^2 .$$

Substituting from the above in (C.8), we obtain that

$$\sum_{i \in N_{k^*}} \|\mathbf{v_{k^*}} - \mathbf{v_i}\|^2 \leq 2 \sum_{i \in S} \|\mathbf{v_i} - \overline{\mathbf{v}}_\mathbf{S}\|^2 \ . \tag{C.9}$$

Consider an arbitrary $i \in S$. By triangle and Jensen's inequalities, we have

$$\|\mathbf{v_{k^*}} - \overline{\mathbf{v}}_\mathbf{S}\|^2 \leq (\|\mathbf{v_{k^*}} - \mathbf{v_i}\| + \|\mathbf{v_i} - \overline{\mathbf{v}}_\mathbf{S}\|)^2 \leq 2\,\|\mathbf{v_{k^*}} - \mathbf{v_i}\|^2 + 2\,\|\mathbf{v_i} - \overline{\mathbf{v}}_\mathbf{S}\|^2 \ .$$

Upon rearranging the terms in the above, we obtain that

$$\|\mathbf{v_{k^*}} - \mathbf{v_i}\|^2 \geq \frac{1}{2}\,\|\mathbf{v_{k^*}} - \overline{\mathbf{v}}_\mathbf{S}\|^2 - \|\mathbf{v_i} - \overline{\mathbf{v}}_\mathbf{S}\|^2 \ .$$

The above, in conjunction with the fact that $|S \cap N_{k^*}| = |S| + |N_{k^*}| - |S \cup N_{k^*}| \geq (n-q) + (n-q) - n = n - 2q$, implies that

$$\begin{aligned}\sum_{i \in S \cap N_{k^*}} \|\mathbf{v_{k^*}} - \mathbf{v_i}\|^2 &\geq \frac{|S \cap N_{k^*}|}{2}\,\|\mathbf{v_{k^*}} - \overline{\mathbf{v}}_\mathbf{S}\|^2 - \sum_{i \in S \cap N_{k^*}} \|\mathbf{v_i} - \overline{\mathbf{v}}_\mathbf{S}\|^2 \\ &\geq \frac{n-2q}{2}\,\|\mathbf{v_{k^*}} - \overline{\mathbf{v}}_\mathbf{S}\|^2 - \sum_{i \in S \cap N_{k^*}} \|\mathbf{v_i} - \overline{\mathbf{v}}_\mathbf{S}\|^2 \ .\end{aligned}$$

As $\sum_{i \in N_{k^*}} \|\mathbf{v_{k^*}} - \mathbf{v_i}\|^2 \geq \sum_{i \in S \cap N_{k^*}} \|\mathbf{v_{k^*}} - \mathbf{v_i}\|^2$, from above we obtain that

$$\sum_{i \in N_{k^*}} \|\mathbf{v_{k^*}} - \mathbf{v_i}\|^2 \geq \frac{n-2q}{2}\,\|\mathbf{v_{k^*}} - \overline{\mathbf{v}}_\mathbf{S}\|^2 - \sum_{i \in S \cap N_{k^*}} \|\mathbf{v_i} - \overline{\mathbf{v}}_\mathbf{S}\|^2 \ .$$

Substituting from the above in (C.9), we obtain that

$$\frac{n-2q}{2}\,\|\mathbf{v_{k^*}} - \overline{\mathbf{v}}_\mathbf{S}\|^2 - \sum_{i \in S \cap N_{k^*}} \|\mathbf{v_i} - \overline{\mathbf{v}}_\mathbf{S}\|^2 \leq 2 \sum_{i \in S} \|\mathbf{v_i} - \overline{\mathbf{v}}_\mathbf{S}\|^2 \ .$$

Using the fact that $\sum_{i \in S \cap N_{k^*}} \|\mathbf{v_{k^*}} - \mathbf{v_i}\|^2 \leq \sum_{i \in S} \|\mathbf{v_{k^*}} - \mathbf{v_i}\|^2$ in the above yields

$$\frac{n-2q}{2}\,\|\mathbf{v_{k^*}} - \overline{\mathbf{v}}_\mathbf{S}\|^2 \leq 3 \sum_{i \in S} \|\mathbf{v_i} - \overline{\mathbf{v}}_\mathbf{S}\|^2 \ .$$

Therefore, recalling that $|S| = n - q$, from the above, we obtain that

$$\|\mathbf{v_{k^*}} - \overline{\mathbf{v}}_\mathbf{S}\|^2 \leq \frac{6(n-q)}{n-2q}\,\frac{1}{|S|} \sum_{i \in S} \|\mathbf{v_i} - \overline{\mathbf{v}}_\mathbf{S}\|^2 \ .$$

This concludes the proof. □

C.4 Geometric Median

The *geometric median* of $\mathbf{v_1}, \ldots, \mathbf{v_n} \in \mathbb{R}^d$, denoted by GeoMed, is defined to be a vector that minimizes the sum of the Euclidean distances to these vectors. Specifically,

$$\text{GeoMed}(\mathbf{v_1}, \ldots, \mathbf{v_n}) \in \operatorname*{argmin}_{\mathbf{y} \in \mathbb{R}^d} \sum_{i=1}^{n} \|\mathbf{y} - \mathbf{v_i}\| .$$

Proposition C.3 *Let $n \in \mathbb{N}^*$ and $q < \frac{n}{2}$. Geometric median is (q, κ)-robust with $\kappa = 4\left(1 + \frac{q}{n-2q}\right)^2$.*

Proof Let us denote by $\mathbf{v}^* \coloneqq \text{GeoMed}(\mathbf{v_1}, \ldots, \mathbf{v_n})$ the geometric median of the input vectors. Consider any subset $S \subseteq [n]$ of size $|S| = n - q$. Due to the triangle inequality, for all $i \in S$, we have

$$\left\|\mathbf{v}^* - \mathbf{v_i}\right\| \geq \left\|\mathbf{v}^* - \overline{\mathbf{v}}_\mathbf{S}\right\| - \|\mathbf{v_i} - \overline{\mathbf{v}}_\mathbf{S}\| . \tag{C.10}$$

Similarly, for any $i \in [n] \setminus S$, we obtain

$$\left\|\mathbf{v}^* - \mathbf{v_i}\right\| \geq \|\mathbf{v_i} - \overline{\mathbf{v}}_\mathbf{S}\| - \left\|\mathbf{v}^* - \overline{\mathbf{v}}_\mathbf{S}\right\| . \tag{C.11}$$

Summing up (C.10) and (C.11) over all input vectors, we obtain

$$\sum_{i \in [n]} \left\|\mathbf{v}^* - \mathbf{v_i}\right\| \geq (n - 2q) \left\|\mathbf{v}^* - \overline{\mathbf{v}}_\mathbf{S}\right\| + \sum_{i \in [n] \setminus S} \|\mathbf{v_i} - \overline{\mathbf{v}}_\mathbf{S}\| - \sum_{i \in S} \|\mathbf{v_i} - \overline{\mathbf{v}}_\mathbf{S}\| .$$

Rearranging the terms, we obtain that

$$\left\|\mathbf{v}^* - \overline{\mathbf{v}}_\mathbf{S}\right\| \leq \frac{1}{n - 2q} \left(\sum_{i \in [n]} \left\|\mathbf{v}^* - \mathbf{v_i}\right\| - \sum_{i \in [n] \setminus S} \|\mathbf{v_i} - \overline{\mathbf{v}}_\mathbf{S}\| + \sum_{i \in S} \|\mathbf{v_i} - \overline{\mathbf{v}}_\mathbf{S}\| \right) .$$

Note that by the definition of GeoMed, we have

$$\sum_{i \in [n]} \left\|\mathbf{v}^* - \mathbf{v_i}\right\| \leq \sum_{i \in [n]} \|\overline{\mathbf{v}}_\mathbf{S} - \mathbf{v_i}\| .$$

Therefore,

$$\left\|\mathbf{v}^* - \overline{\mathbf{v}}_S\right\| \leq \frac{1}{n-2q}\left(\sum_{i\in[n]}\|\mathbf{v_i} - \overline{\mathbf{v}}_S\| - \sum_{i\in[n]\setminus S}\|\mathbf{v_i} - \overline{\mathbf{v}}_S\| + \sum_{i\in S}\|\mathbf{v_i} - \overline{\mathbf{v}}_S\|\right)$$
$$\leq \frac{2}{n-2q}\sum_{i\in S}\|\mathbf{v_i} - \overline{\mathbf{v}}_S\| \ .$$

Upon squaring both sides and using Jensen's inequality, we obtain that

$$\left\|\mathbf{v}^* - \overline{\mathbf{v}}_S\right\|^2 \leq \frac{4(n-q)}{(n-2q)^2}\sum_{i\in S}\|\mathbf{v_i} - \overline{\mathbf{v}}_S\|^2 = \frac{4(n-q)^2}{(n-2q)^2}\cdot\frac{1}{n-q}\sum_{i\in S}\|\mathbf{v_i} - \overline{\mathbf{v}}_S\|^2$$
$$= 4\left(1+\frac{q}{n-2q}\right)^2\frac{1}{|S|}\sum_{i\in S}\|\mathbf{v_i} - \overline{\mathbf{v}}_S\|^2 \ .$$

This concludes the proof. □

C.5 Coordinate-Wise Median

For input vectors $\mathbf{v_1}, \ldots, \mathbf{v_n}$, their *coordinate-wise median*, denoted by CWMed, is defined to be a vector whose k-th coordinate, for all $k \in [d]$, is defined to be

$$[\text{CWMed}\,(\mathbf{v_1}, \ldots, \mathbf{v_n})]_k \coloneqq \text{Median}\,([\mathbf{v_1}]_k, \ldots [\mathbf{v_n}]_k) \ . \tag{C.12}$$

Proposition C.4 *Let $n \in \mathbb{N}^*$ and $q < \frac{n}{2}$. Coordinate-wise median is (q, κ)-robust with $\kappa = 4\left(1+\frac{q}{n-2q}\right)^2$.*

Proof Given that the geometric median coincides for one-dimensional inputs with their median, it follows from Proposition C.3 that, for one-dimensional inputs, median is (q, κ)-robust with $\kappa = 4\left(1+\frac{q}{n-2q}\right)^2$. Since CWMed applies the median operation independently for each coordinate, the proof concludes upon invoking Lemma C.2. □

References

1. Allouah Y, Farhadkhani S, Guerraoui R, Gupta N, Pinot R, Stephan J (2023) Fixing by mixing: a recipe for optimal byzantine ML under heterogeneity. In: Proceedings of the 26th International Conference on Artificial Intelligence and Statistics, vol. 206. Proceedings of Machine Learning Research. PMLR, pp. 1232–1300
2. Bubeck S (2015) Convex optimization: algorithms and complexity. In: Foundations and Trends® in Machine Learning 8.4, pp 231–357
3. Nesterov Y, et al. (2018) Lectures on convex optimization, vol. 137. Springer, New York
4. Stich SU (2019) Unified optimal analysis of the (stochastic) gradient method. Preprint. arXiv:1907.04232