EPFL

Thèse n° 8999

Practical Byzantine-resilient Stochastic Gradient Descent

Présentée le 24 février 2022

Faculté informatique et communications
Laboratoire de calcul distribué
Programme doctoral en informatique et communications

pour l'obtention du grade de Docteur ès Sciences

par

Sébastien Louis Alexandre ROUAULT

Acceptée sur proposition du jury

Prof. V. Kuncak, président du jury
Prof. R. Guerraoui, directeur de thèse
Dr M. Seeger, rapporteur
Dr M. Á. Veganzones, rapporteur
Prof. M. Jaggi, rapporteur

École
polytechnique
fédérale
de Lausanne

2022

Acknowledgements

To be honest, more than four years ago, I had no intention of beginning a thesis after my second master. It is my advisor, Rachid Guerraoui, who convinced me then to apply to the PhD program of the faculty. His arguments were on point: at worst it does not change my plans, at best it would open me another door. During my master project at Cisco, Dr. Patrick Marlier, Dr. Sergio Mena and Dr. Pedro Ramalhete convinced me that the PhD is not a mere academic curriculum choice, but an exceptional opportunity to grow in many ways. I am grateful for their wise words of advice; they were right.
I am thankful for having been accepted in the PhD program and Rachid Guerraoui's laboratory. I am grateful for Rachid's exceptional overall vision and ability to ask the right (and sometimes difficult) questions, the questions that can make one sees his own work from a completely different angle, unveiling new paths, and for constantly pushing me to synthesize and explain potentially complex matters with simple words. I am also grateful to Rachid who from day one, when I wanted to work on shared memory problems, instead invited me to take a look at this then-new line of work, Byzantine machine learning, and introduced me to El Mahdi El Mhamdi.

I cannot overstate how instrumental El Mahdi El Mhamdi has been in this thesis. Our collaboration brought forth many technical discussions, that turned into new formulations, new problems, new ideas and ultimately new contributions to the literature. Mahdi's intellectual sharpness may only be rivaled by Lê Nguyen Hoang, who joined EPFL just after I started my PhD. The three of us had many stimulating philosophical discussions, from epistemic questions to ethical considerations, which really changed the way I think today and made these last four years so exhilarating.
More punctually, I also had fruitful collaborations with several other PhD students, both inside and outside EPFL, all duly acknowledged in Section 1.2.1. The use of the pronoun "we" instead of "I" in the rest of the thesis reflects that fact.

Most importantly, I am grateful to my parents for providing for and educating me in the twenty-three years predating this thesis. Without their sustained efforts and joint dedication to my brother and myself, I would not be presenting this thesis today.
I may not fully realize how lucky I am to have grown up in fully developed countries, legacy of the work of so many before me. Thinking about it, one can question to which extent a thesis may actually be more of a common effort than a personal endeavor.

I stand on the shoulders of my elders. Please consider this section incomplete.

Abstract

Algorithms are everywhere. The recipe for the frangipane cake is an algorithm. If all the listed ingredients are available and the cook is sufficiently deft, after a finite number of small, well-defined steps a delicious dessert will exit the oven.
Now, what if the grocer who sold the cook the ingredients mislabeled some of them? If salt had been *mistakenly* labeled and sold as sugar, those tasting the cake would probably be disgusted and stop eating. If a dangerous drug had been *maliciously* sold as sugar, the consequences could be much more awful.

When it comes to security, machine learning is perhaps closer to cooking than it is to other families of computer programs. Follow the recipe for the Margherita pizza using cream and onions instead of tomatoes and mozzarella, and you may pull a Flammekueche out of the oven. Likewise, what the practitioner gets at the end of the training certainly depends on the relevance of the model and other hyperparameters, but ultimately the output parameters are defined by the inputs: the training set.
While for cookery we simply trust the supply chain and blame the cook for mistakes, we should expect machine learning algorithms to gracefully handle (partially) malformed and malicious datasets and inputs, as for any other computer program.

Byzantine machine learning studies malicious behaviors during the training phase. A particular, growing body of work has been tackling Byzantine failures in the context of distributed Stochastic Gradient Descent (SGD). A central server distributes and safely aggregates gradients from several worker nodes, some of them being adversarial.

On the theoretical side, this thesis tries to advance the state-of-the-art on two fronts. Existing works had always considered a *trusted*, central parameter server. We propose a novel algorithm that has the same proven guarantees as previous works, but does not require any node to be trusted and can operate under network asynchrony.
Two other contributions tackle problems either in the construction or an assumption common to *statistically-robust* Gradient Aggregation Rule (GAR), that made them very vulnerable to attacks in practice. We put a substantial emphasis on devising pragmatic, easy to implement and computationally cheap techniques to address these issues.

On the system side, this thesis implements many of the existing and contributed GARs, integrated into two major machine learning frameworks (PyTorch and TensorFlow). These algorithms are assesses from microbenchmarks on a single GPU to actual networked deployments on many nodes. The associated code has been open-sourced.

Résumé

Les algorithmes sont omniprésents. La recette de la tarte frangipane est un algorithme. Si tous les ingrédients sont disponibles et le patissier suffisament dégourdi, après un nombre fini d'étapes simples et bien définies, un délicieux dessert sortira du four. Maintenant que se passeraît-il si, sans changer l'algorithme, certains ingrédients utilisés par notre cuisinier n'étaient pas les bons ? Replacer *par erreur* le sucre par du sel rendra la tarte infecte, sans plus de conséquence. Replacer *par malice* le sucre par une drogue dangereuse risquerait de s'avérer beaucoup plus grave.

Concernant la sécurité, l'apprentissage machine se rapproche probablement davantage de la cuisine que d'autres types de logiciels. Suivez la recette de la pizza margarita en replaçant la tomate et la mozzarella par de la crème et des oignons, et ce que vous sortirez du four ressemblera à une Flammekueche. Les hyperparamètres choisis influent évidemment sur le résultat de l'apprentissage, mais le comportement final du modèle est avant tout défini par le dataset donné à l'algorithme d'apprentissage.
En cuisine, on peut avoir confiance en l'intégrité de la chaîne d'approvisionnement. Le logiciel lui ne bénéficie naturellement pas des mêmes contrôles et garanties sur les données qui lui sont fournies, et doit donc résister à des entrées malveillantes.

Le champ d'étude des comportement adversariaux durant l'apprentissage porte le nom d'*apprentissage machine Byzantin*. Une application très étudiée est l'algorithme distribué du gradient stochastique : un serveur central distribue et aggrège de manière robuste les gradients calculés par plusieurs nœuds, certains pouvant être adversariaux.

Sur l'aspect théorique, cette thèse se veut d'avancer l'état de l'art dans deux directions. La littérature ayant toujours supposé l'existence d'un serveur central *de confiance*, nous proposons un nouvel algorithme qui conserve les mêmes garanties sans pour autant dépendre d'un serveur de confiance, et est capable de fonctionner en asynchronie. Deux autres contributions s'attaquent à des problèmes soit dans la construction, soit dans une hypothèse commune aux défenses dites *statistiquement robutes*, qui rendaient ces défenses particulièrement vulnérables à certaines attaques. Nos solutions se veulent pragmatiques, simples à implémenter et à faible coût calculatoire.

Sur l'aspect système, cette thèse implémente et intégre de nombreuses défenses à deux des frameworks les plus utilisés en apprentissage machine (PyTorch et TensorFlow). Les performances de ces défenses seront évaluées, de simple microbenchmarks sur GPU à des déploiements réels sur plusieurs nœuds en réseau. Le code est open source.

Contents

Introduction

This thesis is fundamentally about *software security*.

1.1 When Data becomes Code

"Hello, World!" programs are famous in the software development folklore, as being one of the first programs ever written by novice developers. They are (conceptually) extremely simple: they display somehow the text "Hello, world!".

"Hello, World!" programs are interesting for their behavior is *independent* from any input, and so unaffected by adversarial inputs in particular. This is a singular situation in software. The purpose of software system is precisely to (automatically) process information in meaningful ways. Data *must* affect the behavior of software systems.

As software system inputs can be malicious, *software security* strives to "allow intended use of software [and to] prevent unintended use that may cause harm"[1]. Data must only affect software systems in intended ways. Let us take an example: microblogging. Microblogging allows users to publicly publish small pieces of content (text, images, videos, etc). Adversarial submissions to such a system could for instance display arbitrary content under the name of the adversarial user: this is how the software system is intended to operate[2]. Adversarial submissions must not be able to stop the system, modify the content published by other users, etc: this would be a violation of the *security policies*, which precisely define *what* the system can and cannot do.

Machine learning is a whole new paradigm, especially regarding software security. The root problem with machine learning security lies with the security policies: the expected behavior of the trained model is not well-defined. That is actually the purpose of machine learning: to take over when humans cannot (know how to) directly decompose the (intuitively) desired behavior into elementary, programmable steps, because it is too complex or too loosely defined. Machine learning instead considers a

[1] This short definition was given in the master course CS-412 at EPFL, in 2019, by Pr. Mathias Payer.

[2] At least before content moderation is applied; and for moderation to ever happen, the system state must have already been changed, altered by the (malicious) data as intended by the designer/defender.

large space of behaviors, and automatically selects a behavior that *maximizes* some metric. For instance with *reinforcement learning*, the machine learning algorithm must maximize the *reward*. In the case of *supervised learning* (this thesis), the machine learning algorithm must maximize how well the chosen behavior *fits* some input data.

To illustrate, at one end of the spectrum, "Hello, World!" programs do not take any input and so are invulnerable to adversarial inputs. Further into the spectrum, *regular* software systems can *precisely* define how they (should) interact with their inputs, which only affect such systems in restricted ways (if specifications are complete enough and systems precisely follow their specifications). And at the opposite end of the spectrum lies machine learning, where the training data has free reign over which behavior the final model will follow among an arbitrarily large set of possible behaviors.

How to tackle machine learning security then?

The defensive strategies studied in this thesis are techniques that make the training procedure *robust to a minority* of Byzantine inputs. Namely, the result of the training procedure with Byzantine agents should ideally be indistinguishable from the result without Byzantine agents. The validity of this strategy depends on one strong assumption: in the absence of Byzantine agents (e.g. if the training set is only composed of non-adversarial samples), the final model will enforce (our intuition of) the security policies. Unfortunately, this assumption is almost certainly not true in practice: there is a whole branch of the literature highlighting issues with final models, that are trained without Byzantine agents but still exhibit unwanted behaviors (Biggio et al., 2013).

This thesis only proposes incomplete answers to the question of machine learning security; and the road ahead this challenging topic looks particularly long.

1.2 Organization

Chapter 2 will introduce the technical background of this thesis, concluding with a critical eye on the applicability and limitations of the established formalizations.

Chapter 3 will identify a flaw, that affects a whole subfamily of defenses. This flaw stems from a particular defensive construction, that suffers the *curse of dimensionality*. We will formally study to what extent an attacker can bypass three affected defenses. Using this knowledge, we will build and experiment with an actual attack, and observe its (substantial) impact on the training. Finally, we will propose a new, pragmatic, *composite*, Byzantine resilient Gradient Aggregation Rule (GAR) defeating this attack.

Chapter 4 will clear the assumption of a *trusted*, central parameter server out of distributed, Byzantine SGD. The threat model of Byzantine SGD (Section 2.2) aims at shielding the training from *arbitrary* faults, but this threat model had always arbitrar-

ily assumed the faults would be limited to the worker nodes only. Additionally, the threat model never (needed to) consider asynchronous communications, which, if not supported by the defender, can be another source of vulnerabilities in practice. We will propose a distributed algorithm that is built to work in asynchrony and supports multiple parameter servers, some of which being Byzantine. We will prove that our algorithm nevertheless achieves the same theoretical guarantees as standard Byzantine SGD. Finally, we will experimentally assess the slowdown induced by our solution.

Chapter 5 will tackle two state-of-the-art attacks, targeting any *statistically-robust* Byzantine resilient GARs with devastating effects. The main identified weakness in this family of GARs is their requirement for a sufficiently low *variance-norm* ratio, a requirement intuitively capturing how informative non-Byzantine gradients must be. We will propose a practical method which, despite increasing the variance, reduces the *variance-norm* ratio, mitigating the identified weakness. Besides theoretical support, we will assess the effectiveness of our method over several thousands runs, seeded for confidence and reproducibility purposes, spawning small to fairly large models, four image classification tasks and testing every combination of five other hyperparameters.

On the system side, Chapter 6 will implement and assess Byzantine-resilient GAR in actual, datacenter-scale distributed settings. We will strive to write optimized, parallelized and *specialized* GAR implementations. This chapter will then explore the challenges behind deploying Byzantine resilient GARs inside a modern machine learning framework, and to what extend existing defense can affect the training performances.

Chapter 7 will then implement these Byzantine-resilient GARs on GPUs, revealing whether gradient aggregation costs could be marginalized in actual deployments. We will see that fusing or otherwise specializing our implementations can lead to drastic performance improvements; even beating PyTorch's *Median* implementation on GPU.

1.2.1 Associated Publications

Substantial portions of the following publications[3] will be included in this thesis[4]:

1. The Hidden Vulnerability of Distributed Learning in Byzantium
   El-Mahdi El-Mhamdi, Rachid Guerraoui, Sébastien Rouault
   ICML 2018 — 35th International Conference on Machine Learning
   Stockholm, Sweden, July 10–15, 2018
   🎖 *Accepted with a "Long Talk"*

2. AGGREGATHOR: Byzantine Machine Learning via Robust Gradient Aggregation
   Georgios Damaskinos, El-Mahdi El-Mhamdi, Rachid Guerraoui, Arsany Guirguis,

[3] Please note that the author names are always ordered alphabetically.
[4] This publication list is ordered by date of publication, not by order of appearance in the thesis.

Sébastien Rouault
MLSys 2019 — 1st Conference on Machine Learning and Systems
Palo Alto, CA, USA, March 31–April 2, 2019

3. Genuinely Distributed Byzantine Machine Learning
El-Mahdi El-Mhamdi, Rachid Guerraoui, Arsany Guirguis, Lê Nguyen Hoang, Sébastien Rouault
PODC 2020 — ACM 39th Symposium on Principles of Distributed Computing
Selerno, Italy, August 3–7, 2020

4. Fast and Robust Distributed Learning in High Dimension
El-Mahdi El-Mhamdi, Rachid Guerraoui, Sébastien Rouault
SRDS 2020 — IEEE 39th International Symposium on Reliable Distributed Systems
Shanghai, China, September 21–24, 2020

5. Distributed Momentum for Byzantine-resilient Stochastic Gradient Descent
El-Mahdi El-Mhamdi, Rachid Guerraoui, Sébastien Rouault
ICLR 2021 — 9th International Conference on Learning Representations
Vienna, Austria, May 4–8, 2021

The following publications will not appear or only partially appear in this thesis:

6. On The Robustness of a Neural Network
El-Mahdi El-Mhamdi, Rachid Guerraoui, Sébastien Rouault
SRDS 2017 — IEEE 36th Symposium on Reliable Distributed Systems
Hong Kong, China, September 26–29, 2017

7. AKSEL: Fast Byzantine SGD
Amine Boussetta, El-Mahdi El-Mhamdi, Rachid Guerraoui, Alexandre Maurer, Sébastien Rouault
OPODIS 2020 — 24th International Conference on Principles of Distributed Systems
Strasbourg, France, December 14–16, 2020
🎖️ *Best Student Paper award*

8. GARFIELD: System Support for Byzantine Machine Learning
Rachid Guerraoui, Arsany Guirguis, Jérémy Max Plassmann, Anton Alexandre Ragot, Sébastien Rouault
DSN 2021 — 51st IEEE/IFIP International Conference on Dependable Systems and Networks
Taipei, Taiwan, June 21–24, 2021

9. Differential Privacy and Byzantine Resilience in SGD: Do They Add Up?
Rachid Guerraoui, Nirupam Gupta, Rafaël Pinot, Sébastien Rouault, John Stephan
PODC 2021 — ACM 40th Symposium on Principles of Distributed Computing
Selerno, Italy, July 26–30, 2021

10. Collaborative Learning in the Jungle
    El-Mahdi El-Mhamdi, Sadegh Farhadkhani, Rachid Guerraoui, Arsany Guirguis, Lê Nguyên Hoang, Sébastien Rouault
    NeurIPS 2021 — 35th Conference on Neural Information Processing Systems
    Virtual only, December 7–10, 2021

1.2.2 Notation

Symbol	Description	1st reference
t	Current step number	
n	Total number of workers in a distributed deployment	
f	Maximum number of Byzantine workers among the n workers in a distributed deployment	
$i, j, k, \ldots$	Worker/server identifiers in a distributed deployment	
d	Dimension of the parameter vector (usually $d \gg 1$)	
$\|\cdot\|_p$	ℓ_p norm. Without p, $\|\cdot\|$ represents the Euclidean norm.	
$\mathbf{1}_y(x)$	Indicator function, equals 1 when $x = y$ else equals 0	
$\theta_t^{(i)} \in \mathbb{R}^d$	Parameter vector held by server node i at step t; when there is only one server, we note this vector θ_t instead	Section 2.1.1
$Q : \mathbb{R}^d \to \mathbb{R}$	Loss function to minimize	Section 2.1.1
η_t	Learning rate used at step t	Equation (2.2)
μ	Momentum factor	Equation (2.3)
$g_t^{(i)}$	Gradient *sampled* by worker i at step t; as with $\theta_t^{(i)}$, when there is only one worker involved, we note g_t	Equation (2.5)
$G_t^{(i)}$	Gradient *submitted* by worker i at step t to the Byzantine-resilient gradient aggregation rule	Section 2.1.2
$F : \left(\mathbb{R}^d\right)^n \to \mathbb{R}^d$	An unspecified Gradient Aggregation Rule (GAR)	Equation (2.6)
G_t	Aggregated gradient at step t	Equation (2.6)

Table 1.1: Notation used throughout this thesis.

1.2.3 Contributions

This thesis compiles portions of papers co-published (Section 1.2.1) with the author, although this thesis sometimes features substantial differences from these papers. This section[5] goes through each chapter and points out the contributions of the author. The introduction, Part IV and each *concluding remarks* sections are novel and unpublished.

Chapter 2 is a new redaction of existing, preliminary concepts. The critical analyses expressed in this chapter, and in particular in Section 2.2.2, are new and solely represent the point-of-view of the author.

[5]This section had been requested by a jury member before the oral exam.

Chapter 3. The attack proposed in this chapter is originally the idea of the author. The defense (*Bulyan*) is a joint work with El-Mahdi El-Mhamdi. The proof of resilience of *Bulyan* is the work of El-Mahdi El-Mhamdi, and it is available in the appendix. The proofs included within this chapter are the work of the author, and contributions from El-Mahdi El-Mhamdi are systematically acknowledged with a footnote. The proof in Section 3.2.3 is an improved analysis over the published one (El-Mhamdi et al. (2018), Section B). The experiments (Section 3.4) and the software is the work of the author.

Chapter 4. The idea to "use *Krum* to aggregate [parameter vectors]" is from El-Mahdi El-Mhamdi. The algorithm is from the author, except the amortization of the contraction step every T steps. The convergence proof is the work of the author, and technical contributions from Lê Nguyen Hoang are systematically acknowledged with a footnote. The author contributed the code related to the GARs (implementations on CPU/GPU and automated compilation/loading) and miscellaneous helper functions. The remaining code and the experiments are the work of Arsany Guirguis. The text in Section 4.3.2 adds and removes elements compared to the original, published version.

Chapter 5. The idea to "use momentum at the workers" is from the author. The theoretical analyses are from the author, and reviewed by El-Mahdi El-Mhamdi. The author contributed the experiments (including the appendix) and the associated code.

Chapter 6. The system design (except the *LossyMPI* part, c.f. Figure 6.2) is the work of the author, along with the associated text (Section 6.1). The author contributed a large majority of the associated code. The *LossyMPI* design and code are the work of Arsany Guirguis, and the *"corrupted data"* attack is the work of Georgios Damaskinos. The author only ran the experiments of Figure 6.6; the other experiments were run by Arsany Guirguis and Georgios Damaskinos. The text in Section 6.2 is a common effort.

Chapter 7 is entirely the work of the author, without external contribution.

Part I Attacks and Defenses

2 Preliminaries

2.1 Machine Learning

Machine learning (ML) algorithms can be distinguished from other classes of computer algorithms for their behavior is to a large extent defined by data[1], instead of code.

Machine learning uses a *parameterized model* to specify a space of functions, e.g. $M(x) = a\,x + b$ parameterized by $(a, b) \in \mathbb{R}^2$ defines a set of linear functions. *Training a model* consists in finding the parameters that optimize some metric on this model using *data*, e.g. minimizing the quadratic error over some given cloud of points.

The literature contains a wide variety of models, from small support vector machines (Cortes and Vapnik, 1995) to deep neural networks (Schmidhuber, 2015) with billions of parameters. While closed formulas exist to fit a simple linear function, finding optimal parameters for large, complex models requires a different approach.

One key optimization algorithm, which is the workhorse behind many advances and modern results in machine learning, is *Stochastic Gradient Descent*.

2.1.1 Stochastic Gradient Descent

We consider the classical problem of optimizing a *loss function* $Q : \mathbb{R}^d \to \mathbb{R}$, where:

$$Q(\theta_t) \triangleq \mathbb{E}_{X \sim \mathcal{D}} \left[q(\theta_t, X) \right]$$

for a fixed data distribution $\mathcal{D}$. Ideally, we seek θ^* such that $\theta^* = \underset{\theta \in \mathbb{R}^d}{\arg\min} \left(Q(\theta) \right)$.

Remark 1. *Unless otherwise stated, the expectation of random variables is taken over the randomness of the datapoint samplings, i.e. we will simply write:* $Q(\theta_t) \triangleq \mathbb{E}[q(\theta_t)]$.

[1] Data is to be taken in a broad sense, and in particular data includes *feedbacks* from the environment, so as to include *reinforcement learning* and *genetic algorithms* in this definition.

We make the following assumption, standard in the literature (Bottou, 1998; El-Mhamdi, 2020; Su, 2017; Damaskinos, 2020).

Assumption 1 (Derivability of Q)**.**

The loss function Q is differentiable over $\mathbb{R}^d$.

This assumption merely enables us to carry out *Stochastic Gradient Descent* (SGD) and its variant. This assumption is also most reasonable: in practice, models are always made differentiable at least once, enabling *auto-differentiation* as provided by major machine learning frameworks (PyTorch contributors, 2016; TensorFlow contributors, 2015; Abadi et al., 2016).

This thesis will make further assumptions on Q on a per-chapter basis. Unless otherwise noted, we do not make any assumption about the convexity of Q.

Mini-batch SGD optimization.

We employ *mini-batch SGD* optimization to try and seek θ^*.

Starting from an initial parameter $\theta_0 \in \mathbb{R}^d$, at every step $t \geq 0$, b *independent samples* $\left(x_t^{(1)} \dots x_t^{(b)}\right)$ are sampled from $\mathcal{D}$ to estimate one *stochastic gradient*:

$$g_t \triangleq \frac{1}{b} \sum_{k=1}^{b} \nabla q\left(\theta_t, x_t^{(k)}\right) \approx \nabla Q(\theta_t) \tag{2.1}$$

This stochastic gradient is then used to update the *parameters* θ_t, with:

$$\theta_{t+1} = \theta_t - \eta_t \, g_t \tag{2.2}$$

The sequence $\eta_t > 0$ is called the *learning rate*.

Assumption 2 (Convergence of η_t)**.**

The series $\sum_{i=0}^{+\infty} \eta_t$ diverges, and the series $\sum_{i=0}^{+\infty} \eta_t^2$ converges.

Assumption 2 is inherited from (Bottou, 1998) to prove several convergence results.

These two operations (2.1) and (2.2) are repeated for $T \in \mathbb{N}$ steps, generating what we will call the *parameter trajectory* $(\theta_0 \dots \theta_T)$.

In practice, and at least for classifier models[2], the training might stop when a *satisfying* parameter vector θ_t is found. The *top-k cross-accuracy* is a metric commonly used to assess satisfying parameters, and compare different models and training settings.

[2]This thesis will only experiment with classifier models and standard, academic datasets.

Top-*k* cross-accuracy.

In supervised machine learning, and for classification tasks in particular (e.g. assigning names on pictures of people), one relevant metric is the top-k (cross-)accuracy.

Instead of approximating the not-so-intuitive loss function $Q(\theta_t)$ with $\left(x_t^{(1)} \dots x_t^{(b)}\right)$, each sampled datapoint is submitted to the model, which assigns each datapoint a *categorical probability* of belonging to each output class. The top-k accuracy is estimated by counting the number of times the b datapoint labels were in one the k highest categorical probabilities, and dividing this count by b. The top-k accuracy effectively measures how *accurate* the model is when predicting the class of its input.

One important aspect for the practitioner is how well the classifier will *generalize* (i.e. correctly classify) unseen inputs. A metric for this aspect is the top-k ***cross-***accuracy. The only difference with the top-k accuracy is that none of the b sampled datapoints were ever used to update the model (c.f. equations (2.1) and (2.2)). This can be achieved in practice by splitting the labeled dataset in two chunks: the *training* set solely used to update the model, and the *testing* set solely used to estimate the cross-accuracy.

The top-1 cross-accuracy is used extensively throughout the experiments of this thesis. Although the relation between the training loss Q and metrics on the testing set (e.g. testing loss, cross-accuracy) is hardly understood (Belkin et al., 2019), this metric is used extensively in the literature (Roelofs et al., 2019; Zagoruyko and Komodakis, 2016; Krizhevsky et al., 2012; Yamada et al., 2019) to assess/compare model performances.

Momentum SGD.

A field-tested amendment to mini-batch SGD is *momentum* (Polyak, 1964), where each gradient keeps an exponentially-decreasing effect on every subsequent update.

Formally, given $0 < \mu < 1$ the *momentum factor*, the update step (2.2) is replaced with:

$$\theta_{t+1} = \theta_t - \eta_t \sum_{u=0}^{t} \mu^{t-u} g_u \tag{2.3}$$

This formulation will be called *classical momentum*.

Nesterov (1983) proposed another revision. Noting v_t the *velocity vector*, $v_0 = 0$, the gradient computation (2.1) is amended as follows:

$$v_{t+1} = \mu\, v_t + \frac{1}{b} \sum_{k=1}^{b} \nabla q\Big(\theta_t - \eta_t\, \mu\, v_t, x_t^{(k)}\Big) \tag{2.4}$$

The update operation (2.2) remains unchanged, simply using v_{t+1} instead of g_t:

$$\theta_{t+1} = \theta_t - \eta_t \, v_{t+1}$$

This formulation will be called *Nesterov momentum*. Compared to classical momentum, Nesterov momentum estimates the gradient at $\theta_t - \eta_t \, \mu \, v_t$ instead of θ_t.

2.1.2 Distributed Training

Estimating the gradient g_t is the computationally expensive part of the training: it consists in computing $\nabla q\left(\theta_t, x_t^{(k)}\right)$ for each $k \in [1 \,..\, b]$ sampled datapoints $\left(x_t^{(1)} \ldots x_t^{(b)}\right)$. Each computation of $\nabla q\left(\theta_t, x_t^{(k)}\right)$ involves one *forward* pass and one *backpropagation* pass (Hecht-Nielsen, 1992) over the model. Hence, the amount of arithmetic operations to carry out to compute $\nabla q\left(\theta_t, x_t^{(k)}\right)$ is in the order of $\mathcal{O}(b\,d)$. Since $d \gg 1$, estimating the gradient can easily become the bottleneck in Stochastic Gradient Descent.

Fortunately, computing g_t is also embarrassingly parallel: each $\nabla q\left(\theta_t, x_t^{(k)}\right)$ for $k \in [1 \,..\, b]$ can be computed separately. Supposing the batch-size is $b\,n$, we can compute:

$$\begin{aligned} g_t &\triangleq \frac{1}{b\,n} \sum_{k=1}^{b\,n} \nabla q\left(\theta_t, x_t^{(k)}\right) \\ &= \frac{1}{n} \sum_{i=1}^{n} \Bigg(\underbrace{\frac{1}{b} \sum_{k=b\,i+1}^{b\,(i+1)} \nabla q\left(\theta_t, x_t^{(k)}\right)}_{\triangleq \, g_t^{(i)}} \Bigg) \end{aligned} \tag{2.5}$$

The computation of the $g_t^{(i)}$ can be carried out in parallel over n processing units.

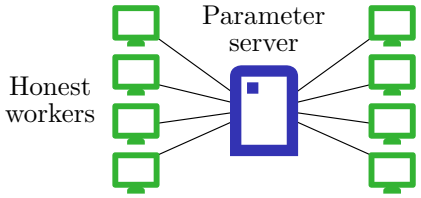


Figure 2.1: Parameter server networked with $n = 8$ worker machines.

Figure 2.1, where a central server holds the parameters θ_t and several workers carry out derivative computations, corresponds to the now standard *parameter server* architecture, used for instance in (Li et al., 2013).

The distributed training algorithm can be summarized as follows. For each step t:

1. The parameter server *broadcasts* the vector θ_t to each worker i.
2. Each worker i computes $g_t^{(i)}$ using its own b sampled datapoints.
3. Each worker i sends back its gradient $G_t^{(i)}$ to the parameter server.
   Unless otherwise specified (c.f. Chapter 5), $G_t^{(i)} = g_t^{(i)}$.
4. The parameter server *aggregates* $G_t^{(1)} \dots G_t^{(n)}$ into one gradient G_t.
5. The parameter server updates θ_t with either Equation (2.1) or (2.3), e.g.:

$$\theta_{t+1} = \theta_t - \eta_t G_t$$

One noteworthy operation in the above algorithm is the aggregation of the n received gradients into one, using a *Gradient Aggregation Rule* (GAR) $F : \left(\mathbb{R}^d\right)^n \to \mathbb{R}^d$:

$$G_t \triangleq F\left(G_t^{(1)}, \dots, G_t^{(n)}\right) \tag{2.6}$$

As an example, the gradient aggregation rule implicitly used by Equation (2.5) is the mere arithmetic mean of the n received gradients $\left(G_t^{(1)} \dots G_t^{(n)}\right) \mapsto \frac{1}{n}\sum_{i=1}^{n} G_t^{(i)}$.

2.2 The Byzantine Model

The root problem we consider in this thesis is when a minority of the processing units (interchangeably called *workers*) behave arbitrarily. In the parlance of distributed computing, that is when f of these n workers are Byzantine.

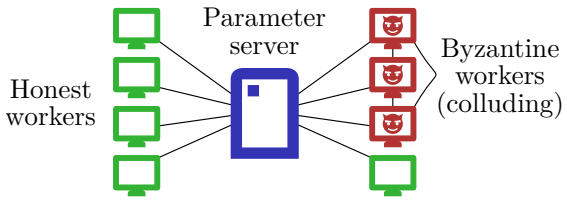


Figure 2.2: Parameter server networked with $n = 8$ worker machines, among which there are $f = 3$ colluding, adversarial machines.

The goal of the adversary is informally to stymie the learning procedure enabled by SGD, to push the parameter vector θ_t into a sub-optimal space, where the model performs poorly. We will consider that a classifier achieving noticeably lower top-k cross-accuracies (and possibly even higher training losses) under attack than without attack performs poorly, and that the adversary fulfilled its objective.

Threat model.

The f Byzantine workers collude. We will then refer to the f Byzantine workers as *the adversary*, as they behave as if they were all under the control of a single entity.

The adversary is omniscient, in the sense that it has a perfect knowledge of the system state at any time. The system state is constituted exhaustively by:

- the full state of the parameter server (data and code), and
- the full state of every non-Byzantine worker (data and code), and
- any data exchanged over any communication channel at any time.

The adversary has arbitrarily fast communication channels and computational capabilities, in the sense that the adversary always has time to carry out its attack and can choose in which order the parameter server will receive the f Byzantine gradients.

The adversary is not omnipotent: it can only send one (or more) messages (e.g. Byzantine gradients) over the network, or choose not to send anything, at any time. The adversary cannot directly modify the memory of non-Byzantine machines. Unless otherwise stated, the adversary cannot impersonate non-Byzantine worker nodes.

Side note 1 (Backdoor attacks)**.**

We considered that higher testing/training losses implies a successful attack: if this happens, the adversary indeed fulfilled its objective to stymie the training. While this consideration is by definition an implication, it is by no mean an equivalence.

A more subtle subclass of attack consists in instilling persistent, unwanted behaviors into the trained model without affecting its performances on both the training and testing sets (Bagdasaryan et al., 2020; Sun et al., 2019). Such a model is said backdoored. *A backdoored classifier could wait for a* cue *in its input (only known by the adversary) to unexpectedly change its output. Wang et al. (2020) theoretically and experimentally supports the existence of backdoors that are difficult to detect, noting in particular that robustness to backdoor attacks implies that the model training procedure is resilient to* adversarial examples *(Biggio et al., 2013; Goodfellow et al., 2014).*

Side note 2 (Security topics in machine learning)**.**

Adversarial behaviors during training, as presented in this section, is not the only security concern. The literature in machine learning security distinguishes three topics:

- *Evasion attacks — when the adversary seeks to deceive a trained model, e.g.: the* universal perturbation *(Moosavi-Dezfooli et al., 2017) or a novel malware detection avoidance technique and its defense (Chen et al., 2017a).*
- *Poisoning attack — when the adversary seeks to affect the trained model: this thesis fits in the poisoning topic, considering an adversary controlling gradients.*

- *Privacy protection — when the adversary tries to gain knowledge upon trained parameters or private data by observing the model inputs/outputs (or even parameters/gradients), e.g.: training data extraction (Carlini et al., 2020) and, for a class of defenses, secure aggregation (Bonawitz et al., 2017).*

This thesis strongly focuses on poisoning attacks, although our latest work (Section 1.2.1), briefly presented in the last chapter (Section 9.3), also studies the combination of differential privacy techniques with defenses against poisoning.

2.2.1 Formal Byzantine resilience

The goal of the defense is to prevent the adversary from stymieing the learning process. Blanchard et al. (2017) proposed a formalization for this loosely-stated objective.

Bottou (1998) proved SGD can converge toward a local extremum of the loss function Q. Building upon this proof, Blanchard et al. (2017) established two sufficient conditions a GAR can satisfy to ensure SGD will still converge to a local extremum under Byzantine behaviors (Section 2.2). These two conditions are gathered below, into Definition 1.

Definition 1 ((α, f)-Byzantine resilience)**.**

Let $G_t^{(1)} \dots G_t^{(n-f)} \sim \mathcal{G}_t$ be $n-f$ independent, non-Byzantine gradients following the same distribution $\mathcal{G}_t$, let $G \sim \mathcal{G}_t$, and let $0 \leq \alpha < \frac{\pi}{2}$.

A gradient aggregation rule $F : \left(\mathbb{R}^d\right)^n \to \mathbb{R}^d$ is (α, f)-Byzantine resilient if and only if:

- $\forall \left(G_t^{(n-f+1)} \dots G_t^{(n)}\right) \in \left(\mathbb{R}^d\right)^f$ *the f Byzantine gradients, and*
- $\forall \mathcal{I} : [1\,..\,n] \hookrightarrow [1\,..\,n]$ *an "index-shuffling" function,*

the aggregated gradient:

$$G_t \triangleq F\left(G_t^{(\mathcal{I}(1))}, \dots, G_t^{(\mathcal{I}(n))}\right)$$

satisfies:

1. $\langle \mathbb{E}\, G_t, \mathbb{E}\, G \rangle \geq (1 - \sin\alpha) \left\|\mathbb{E}\, G\right\| > 0$
2. $\forall r \in \{2, 3, 4\}$, $\mathbb{E}\left\|G_t\right\|^r$ *is bounded above by a linear combination of the terms* $\mathbb{E}\left\|G\right\|^{r_1} \dots \mathbb{E}\left\|G\right\|^{r_k}$, *with* $(k, r_1 \dots r_k) \in \left(\mathbb{N}^*\right)^{k+1}$ *and* $r_1 + \dots + r_k = r$.

Remark 2. *The "index-shuffling" function $\mathcal{I}$ is a notation trick, only to prevent gradient aggregation rules from using the index i of each received gradient $G_t^{(i)}$. For instance, the GAR that outputs $\frac{1}{n-f}\sum_{i=1}^{n-f} G_t^{(i)}$ would spuriously satisfy Definition 1 without $\mathcal{I}$.*

Alternative formalizations.

The literature offers at least two alternative formalizations for Byzantine resilience.

Karimireddy et al. (2020) proposes (δ_{max}, c)-*robust aggregator*. A GAR is a $(\delta_{\max}, c)$-robust aggregator when the expected distance between its output and the average of the non-Byzantine gradients is bounded below a factor of the non-Byzantine gradients' variance. The factor depends on the GAR itself, and the fraction $\frac{f}{n}$ of Byzantine workers.

Liu et al. (2021) proposes, (f, ϵ)-*resilience*, a fundamentally different definition in the sense that a GAR cannot be (f, ϵ)-resilient alone. Only the optimization algorithm *as a whole* can be (f, ϵ)-resilient. An optimization algorithm is (f, ϵ)-resilient when its "output" is, despite Byzantine workers, at most ϵ away (using the ℓ_2 norm) from θ^*.

Side note 3 (Alternative adversarial models)**.**

The Byzantine model and its threat model (Section 2.2) make a particularly powerful adversary. The literature also studies different adversarial models (Liu, 2021). For instance, Charikar et al. (2017) looks at the problem of adversarial data: the adversary can inject fake data, and the defense tries to learn despite this poisoning.

With a slightly different threat model, the literature also contains defenses fundamentally different than gradient filtering. *For instance, Chen et al. (2018) proposes a defense based on* gradient redundancy*, and Rajput et al. (2019) further hybrids the redundancy approach with statistical filtering. We can also cite the* suspicion-based *approach (Xie et al., 2019b; Xie, 2019), assuming the parameter server can also sample from $\mathcal{D}$.*

2.2.2 Applicability and Limitations

From a theoretical standpoint, (α, f)-Byzantine resilience and comparable formalizations all guarantee some form of *convergence* of the parameter vector θ_t to an *optimum*. In the case of convex optimization, this optimum would be θ^*. For a non-convex loss Q, the optimum would be any extremum of the loss (i.e. any θ^* for which $\nabla Q(\theta^*) = 0$). These are the same guarantees Bottou (1998) provided in Byzantine-free settings.

From a practical standpoint, numerous gradient aggregation rules have been developed satisfying (α, f)-Byzantine resilience[3]. A non-exhaustive list is: the coordinate-wise median (Yin et al., 2018), Krum (Blanchard et al., 2017), MDA (El-Mhamdi et al., 2018), GeoMed (Chen et al., 2017b), MeaMed (Xie et al., 2018a), Phocas (Xie et al., 2018c), Aksel (Boussetta et al., 2021), CenteredClip (Karimireddy et al., 2020). All these GARs have been implemented in practice and, notably, several of them run efficiently on both CPUs and GPUs (El-Mhamdi et al., 2020c; Guerraoui et al., 2021a).

[3]Most (α, f)-Byzantine resilient GARs may also be $(\delta_{\max}, c)$-robust, as these formalizations are close.

The Byzantine model presented in Section 2.2 is most suited to settings where the training computations are distributed, but any machine learning pipeline where there are n identified sources of datapoints would fit. This configuration would occur for instance in pipelines where the dataset is generated by a base of n identified users, among which up to f can be Byzantine. This is also a limitation. A different, *weaker* adversarial model may be more suitable if the adversary is not able to generate *arbitrary* Byzantine gradients. The model also implicitly assumes the set of Byzantine nodes is fixed, and promising recent work (El-Mhamdi et al., 2020b; Karimireddy et al., 2020) may actually fail when the identities of the Byzantine nodes change over time. And such a change would most likely be the norm in actual, practical scenarios: the adversary would move as it takes control over new identities/devices/accesses while losing others. Finally, the model assumes the datapoints are sampled independently from the same distribution $\mathcal{D}$. When the dataset is aggregated from many different sources, this assumption may be difficult to sustain (e.g. human labeling may vary across different cultures). Several approaches have already been proposed to tackle Byzantine attacks and heterogeneous datasets (Li et al., 2019; He et al., 2020; El-Mhamdi et al., 2020a).

An important critic one can emit about this model is actually about its formalization, more precisely how much *actionable* the theoretical guarantees of (α, f)-Byzantine resilience (and $(\delta_{\max}, c)$-robustness or (f, ϵ)-resilience alike) are in practice. Namely, and at least in the case of a non-convex loss Q, (α, f)-Byzantine resilience solely enables SGD to converge: *eventually*, a local extremum will be found. Even if we assume that any local extremum always yields satisfying performances, the core problem remains in practice: the final parameters will be found *eventually*. So when the practitioner tries to optimize a non-convex loss while under attack, say for $20\,000$ steps, few formal guarantees about θ_{20000} actually exist. The takeaway is that theoretical guarantees, like ergodic proofs of convergence despite Byzantine workers, will only remain loosely correlated with any low loss/high cross-accuracy[4] measured in actual experiments.

[4]Even without attack only a loose correlation exists, and the relation between low training losses and and low testing losses is in itself only partially understood (Belkin et al., 2019).

3 The Hidden Vulnerability

Statistically-robust aggregation rules are the subset of Byzantine resilient GARs that only use their inputs to derive an aggregated gradient, i.e.: they are all stateless. This chapter focuses on statistically-robust rules based on a *distance-minimization scheme*.

Such stateless aggregations can only compare and combine input gradients with each other, to try and derive a gradient which would satisfy the conditions for (α, f)-Byzantine resilience (Definition 1). As an instance of a distance-minimization scheme, a GAR may measure every pairwise ℓ_2-distances between the input gradients, and output the gradient minimizing its distances with half of its closest neighbors.

These GARs can operate on high-dimensional gradients in $\mathbb{R}^d$, with $d \gg 1000$, and are oblivious to the effect each coordinate actually has on the model. These two observations enable us to build an attack that, while not necessarily preventing convergence, can make SGD converge to *ineffective* parameters.

We also propose a new *composite* GAR that, at the cost of tolerating at best a quarter of Byzantine workers, entirely forestalls the proposed attack.

3.1 Statistically-robust Defenses

We will study two existing statistically-robust, Byzantine resilient defenses, *Krum* (Blanchard et al., 2017) and *Geomed* (Chen et al., 2017b), and devise a third one, *MDA*, inspired from the *Minimum Volume Ellipsoid* (Rousseeuw, 1985). As we will see below, these GARs are all based on a distance-minimization scheme.

The work presented in this chapter follows exactly the distributed setting presented in Section 2.1.2. The attack model is also the one presented in Section 2.2. We recall there are n workers among which f are Byzantine.

3.1.1 Prior Art: Krum and Geomed

These two GARs have a common structure, which we will present first. Their common requirement is that $n \geq 2f + 3$.

Let $k \in \{1,\ 2\}$, let $m \in [1\,..\,n-f-2]$, and let $\|\cdot\|$ be the ℓ_2-norm.

First, a score is assigned to each input gradient $G_t^{(i)}$ with $score(i) \triangleq \sum_{\odot i} \left\|G_t^{(i)} - G_t^{(j)}\right\|^k$, where $\odot i$ designates the set of $n-f-2$ closest input gradients to $G_t^{(i)}$.

Then, the aggregated gradient is arithmetic mean of the m input gradients with the smallest scores. *Geomed* uses $k = 1$, approximating the minimizer of the distances: a "trimmed-median". *Krum* uses $k = 2$, approximating the minimizer of the squared-distances: a "trimmed-mean".

For the remaining of this chapter, we will set $m = n - f - 2$ for both *Krum* and *Geomed*. Blanchard et al. (2017) originally calls simply *Krum* the variant with $m = 1$, while the variant with $m > 1$ is called *Multi-Krum*. Here both variants will be called *Krum*.

3.1.2 MDA: Minimum Diameter Averaging

MDA requires that $n \geq 2f + 1$.

Informally, *MDA* selects the $n - f$ *most clumped* gradients among the submitted ones, and average them as final output. It is reminiscent of the *Minimal Volume Ellipsoid* estimator, introduced by Rousseeuw (1985) and proven to have the optimal *breakdown point* of 50%.

Formally, let:

- $\mathcal{Q} = \left\{G_t^{(1)} \dots G_t^{(n)}\right\}$ be the set of input gradients,
- $\mathcal{R} = \{\mathcal{X} \mid \mathcal{X} \subset \mathcal{Q},\ |\mathcal{X}| = n - f\}$ be the set of all the subsets $\mathcal{X}$ of $\mathcal{Q}$ with a cardinality of $n - f$, and
- $\mathcal{S} = \underset{\mathcal{X}\in\mathcal{R}}{\arg\min} \left(\underset{\left(G_t^{(i)},\, G_t^{(j)}\right)\in\mathcal{X}^2}{\max} \left\|G_t^{(i)} - G_t^{(j)}\right\| \right)$ the set of $n - f$ input gradients with the smallest *diameter*; in case of equality $\mathcal{S}$ can be any of the candidate sets.

Then, the aggregated gradient is given by $G_t = \frac{1}{n-f} \sum\limits_{x\in\mathcal{S}} x$.

As a side note, this rule can hardly be used in large-deployment cases, as $|\mathcal{R}| = \frac{n!}{f!(n-f)!}$. For instance, with $n = 57$ workers and $f = 27$, we have $|\mathcal{R}| \approx 1.4 \cdot 10^{16}$. Even with 10^9

measured subsets $\mathcal{X}$ per second, aggregating these 57 gradients would take more than 5 months.

Since we use *MDA* as a benchmark when experimenting with small amount of workers in Section 3.4.2, we also prove below its (α, f)-Byzantine resilience.

Proof of (α, f)-Byzantine resilience.

We assume the non-Byzantine gradients all follow i.i.d. the distribution $\mathcal{G}_t$. Let $\lambda_t \triangleq \|\mathbb{E}\, G\|$ for $G \sim \mathcal{G}_t$, and $\bar{\sigma}_t \triangleq \mathbb{E}\,\|G - H\|$ for $H \sim \mathcal{G}_t$ independent from G. Under the assumption that $2\, f\, \bar{\sigma}_t < (n-f)\, \lambda_t$, we will prove that *MDA* is (α, f)-Byzantine resilient.

Without loss of generality (*MDA* is agnostic to the indexing), the $n - f$ first input gradients will be the honest ones, and the Byzantine gradients will be noted $B_t^{(1)} \dots B_t^{(f)}$. That is, the set of input gradients can be noted $\mathcal{Q} = \Big\{ \underbrace{G_t^{(1)} \dots G_t^{(n-f)}}_{\text{honest}}, \underbrace{B_t^{(1)} \dots B_t^{(f)}}_{\text{Byzantine}} \Big\}$.

Trivial case: $\forall i \in [1 \,..\, f]\,, B_t^{(i)} \notin \mathcal{S}$.
As the aggregated gradient G_t is directly the arithmetic mean of the non-Byzantine gradients, $\mathbb{E}\, G_t = \mathbb{E}\, G$, and points 1. and 2. of Definition 1 are trivially satisfied.

Otherwise, without loss of generality, let $b \in [1 \,..\, f]$, $\mathcal{S} = \Big\{ G_t^{(1)} \dots G_t^{(n-f-b)}, B_t^{(1)} \dots B_t^{(b)} \Big\}$, and let $\bar{\mathcal{R}} = \mathcal{R} \setminus \mathcal{S}$. Since $\mathcal{S}$ is the subset with the smallest diameter, it holds:

$$
\begin{aligned}
&\forall \bar{\mathcal{S}} \in \bar{\mathcal{R}}, \\
&\exists X_i \in \bar{\mathcal{S}} \setminus \mathcal{S}, \\
&\exists X_j \in \bar{\mathcal{S}} \setminus \{X_i\}\,, \\
&\forall X_k \in \mathcal{S}, \forall X_l \in \mathcal{S} \setminus \{X_k\}\,, \\
&\|X_k - X_l\| < \|X_i - X_j\|
\end{aligned}
$$

We can also notice that:

$$\exists \mathcal{V} \in \bar{\mathcal{R}},\ \forall i \in [1 \,..\, f]\,,\ B_t^{(i)} \notin \mathcal{V}$$

Basically, since $\mathcal{S}$ contains at least one Byzantine gradient, the (unique) set $\mathcal{V}$ that contains only the $n - f$ non-Byzantine gradients is indeed in $\mathcal{R}$.

Then, by combining this observation with the previous one:

$$
\begin{aligned}
\forall a \in [1 \,..\, b]\,, B_t^{(a)} \in \mathcal{S}& \\
&\Rightarrow \exists\, (x_a, y_a) \in [1 \,..\, n-f]^2\,,\ x_a \neq y_a, \\
&\quad\ \forall k \in [1 \,..\, n-f-b]\,,
\end{aligned}
$$

$$\left\|B_t^{(a)} - G_t^{(k)}\right\| \leq \left\|G_t^{(x_a)} - G_t^{(y_a)}\right\| \tag{3.1}$$

This formalization translates the fact that, $\mathcal{S}$ being the set with the smallest diameter, the distances between Byzantine gradients and non-Byzantine gradients in $\mathcal{S}$ can always be bounded above by the distance between some two non-Byzantine workers (in $\mathcal{V}$). This last observation will be reused in the following.

We can compute the aggregated gradient:

$$G_t = \frac{1}{n-f}\left(\sum_{i=1}^{n-f-b} G_t^{(i)} + \sum_{i=1}^{b} B_t^{(i)}\right)$$

and compare it with the average of the non-Byzantine ones:

$$\begin{aligned}
\widehat{G}_t &= \frac{1}{n-f}\sum_{i=1}^{n-f} G_t^{(i)} \\
G_t - \widehat{G}_t &= \frac{1}{n-f}\left(\sum_{i=1}^{b} B_t^{(i)} - \sum_{i=n-f-b+1}^{n-f} G_t^{(i)}\right) \\
&= \frac{1}{n-f}\sum_{i=1}^{b} B_t^{(i)} - G_t^{(i+n-f-b)} \\
\left\|G_t - \widehat{G}_t\right\| &\leq \frac{1}{n-f}\sum_{i=1}^{b}\left\|B_t^{(i)} - G_t^{(i+n-f-b)}\right\| \\
&\leq \frac{1}{n-f}\sum_{i=1}^{b}\left(\left\|B_t^{(i)} - G_t^{(1)}\right\| + \left\|G_t^{(1)} - G_t^{(i+n-f-b)}\right\|\right) \\
\textit{using (3.1)} \rightarrow \quad &\leq \frac{1}{n-f}\sum_{i=1}^{b}\left(\left\|G_t^{(x_i)} - G_t^{(y_i)}\right\| + \left\|G_t^{(1)} - G_t^{(i+n-f-b)}\right\|\right)
\end{aligned}$$

We can then compute the expected value of this distance, and with $\mathbb{E}\,\widehat{G}_t = \mathbb{E}\,G$ and the Jensen's inequality:

$$\begin{aligned}
\|\mathbb{E}\,G_t - \mathbb{E}\,G\| &\leq \mathbb{E}\left\|G_t - \widehat{G}_t\right\| \\
&\leq \frac{1}{n-f}\sum_{i=1}^{b}(\bar{\sigma}_t + \bar{\sigma}_t) \\
&\leq \frac{2\,b\,\bar{\sigma}_t}{n-f} \\
&\leq \frac{2\,f\,\bar{\sigma}_t}{n-f}
\end{aligned}$$

Under the assumption that $2f\bar{\sigma}_t < (n-f)\,\lambda_t$, we verify that $\|\mathbb{E}\,G_t - \mathbb{E}\,G\| < \|\mathbb{E}\,G\|$,

and so: $\langle \mathbb{E}\, G_t, \mathbb{E}\, G \rangle > 0$. Point *1.* of Definition 1 is satisfied.

Point *2.* can also be verified formally[1], $\forall r \in \{2,3,4\}$:

$$\mathbb{E}[\|G_t\|^r] \leq \frac{n-f-b}{n-f}\,\mathbb{E}[\|G\|^r] + \frac{1}{n-f}\sum_{i=1}^{b}\mathbb{E}\Big[\Big\|B_t^{(i)}\Big\|^r\Big]$$

Then reusing (3.1), by using the binomial theorem twice:

$$\Big\|B_t^{(i)}\Big\|^r \leq \sum_{r_1+r_2=r}\binom{r}{r_1}\Big\|B_t^{(i)} - G_t^{(k)}\Big\|^{r_1}\Big\|G_t^{(k)}\Big\|^{r_2}$$
$$\text{for some } k \in [1\,..\,n-f-d]$$

$$\begin{aligned}\Big\|B_t^{(i)} - G_t^{(k)}\Big\|^{r_1} &\leq \Big\|G_t^{(x_i)} - G_t^{(y_i)}\Big\|^{r_1}\\ &\leq \sum_{r_3+r_4=r_1}\binom{r_1}{r_3}\Big\|G_t^{(x_i)}\Big\|^{r_3}\Big\|G_t^{(y_i)}\Big\|^{r_4}\end{aligned}$$

Finally, as $\left(G_t^{(1)}\ldots G_t^{(n-f)}\right)$ are *independent, identically distributed* random variables following the same distribution $\mathcal{G}_t$, we have that:

$$\begin{aligned}&\forall\,(\beta,\gamma) \in \{2,3,4\}^2\,,\\ &\forall\,(i,j) \in [1\,..\,n-f]^2\,,\ i \neq j,\\ &\mathbb{E}\Big[\Big\|G_t^{(i)}\Big\|^\beta\Big\|G_t^{(j)}\Big\|^\gamma\Big] = \mathbb{E}\Big[\|G\|^\beta\Big]\cdot\mathbb{E}\Big[\|G\|^\gamma\Big]\end{aligned}$$

and so $\mathbb{E}\Big[\Big\|B_t^{(i)}\Big\|^r\Big]$ is bounded as described in point *2.* of Definition 1.

□

3.2 The Curse of Dimensionality

This section presents the intuition, along a more formal analysis, behind the inherent vulnerability suffered by statistically-robust, Byzantine resilient GARs based on a distance minimization scheme.

3.2.1 Intuition

In high dimensions, the distance function between two vectors $\|X-Y\|$ cannot answer this core question: *do X and Y "disagree" **a bit** on each coordinate, or do they disagree **a lot** on only one?* SGD has proven its ability to accommodate "small errors" from

[1]The key idea of using the binomial theorem is from El Mahdi El Mhamdi.

the gradient estimation. Such "errors" are often *beneficial*, as they may allow the descent process to leave sub-optimal local minima (Bottou, 2012). In Byzantine-free distributed setups, gradient estimations *"disagree" **a bit** on each coordinate*[2].

In a vector space of dimension $d \gg 1$, the "bit of disagreement" on each coordinate translates into a distance $\|X - Y\| = \mathcal{O}\left(\sqrt{d}\right)$. For the omniscient adversary described in Section 2.2, it translates into an opportunity to submit f Byzantine gradients that "disagree" **a lot**, for instance as much as $\mathcal{O}\left(\sqrt{d}\right)$, on only one coordinate with at least one non-Byzantine gradient. This coordinate could for instance be a *bias* in one of the latest linear activation layer of a neural network, potentially having a substantial impact on the output of the model. As the ℓ_2 norm[3] cannot answer the core question mentioned in the above paragraph, such Byzantine gradient could then be *selected* by a GAR based on such distance-minimization schemes.

The gradient aggregation rules presented in Section 3.1 all perform a linear combination of the selected gradient(s). Thus the final aggregated gradient might for instance have one unexpectedly high coordinate. Depending on the learning rate (Figure 3.5), updating the model with such gradient may push and keep the parameter vector in a sub-space *rarely reached* with the usual, Byzantine-free distributed setup.

The experiments gathered in Section 3.4 clearly show this dependency on the learning rate and indicate that, even if convergence can be achieved, this sub-space only offers sub-optimal to utterly *ineffective* models.

3.2.2 The Attack

The adversary defined in Section 2.2 is omniscient and has arbitrary fast computation and transmission throughput. So for each round, every time the $n - f$ non-Byzantine gradients, are produced, the adversary reads them and chooses the other f gradients the parameter server receives. Based on that capability, for each round, the adversary waits for $n - f$ non-Byzantine gradients to be received. Then it attacks by sending f times the same Byzantine gradient B_t.

Formally, let:

- $\mathcal{Q} \triangleq \left\{ G_t^{(1)} \dots G_t^{(n-f)} \right\}$ be the set of submitted, non-Byzantine gradients (in $\mathbb{R}^d$),
- $E \triangleq (0 \dots 0, 1, 0 \dots 0) \in \mathbb{R}^d$ be any coordinate to attack,
- $\mathcal{B}_t(\gamma) = \gamma E + \frac{1}{n-f} \sum_{X \in \mathcal{Q}} X$ a function generating an attack gradient.

[2] This has been observed during the experiments.

[3] The same reasoning could be made with a ℓ_p norm with p finite, and when p is very large or infinite we also propose an attack (Section 3.2.2).

By a simple search, we estimate the highest value of γ, noted γ_m, such that $B_t = \mathcal{B}_t(\gamma_m)$ is selected by the aggregation rule. Finally, B_t is submitted by every Byzantine worker.

For each presented GAR, we derive in Section 3.2.3 the relation between a *rough* estimation of γ_m and a few hyper-parameters. We study these approximations of γ_m within the *minimal quorum* cases, where the proportion of Byzantine workers is maximized, respectively: $n = 2f + 1$ for *MDA* and $n = 2f + 3$ for *Krum*/*Geomed*.

As a side-note, an adversary does not necessarily need to know the submitted, non-Byzantine gradients $\mathcal{Q}$ with this attack. Indeed non-Byzantine gradients are assumed to be unbiased, so by the law of large numbers we have: $\lim_{|\mathcal{Q}|\to+\infty} \mathcal{B}_t(\gamma) = \mathbb{E}\,X + \gamma\,E$ for $X \sim \mathcal{G}_t$. It indicates that, for this attack to succeed as well, the adversary may only need to compute an unbiased gradient estimate by itself (without the need to "spy" on the other workers) then add $\gamma\,E$ to it.

Attack on the ℓ_p norm with p large.

With $d \gg 1$ fixed: $\lim_{p\to+\infty} \sqrt[p]{d} = 1$. Basically, the *curse of dimensionality* exploited in the above attack seems not to exist any longer with p large enough, or *infinite*.

We propose a simple tweak for such cases. One effective attack consists simply in changing the vector $E = (0\ldots0, 1, 0\ldots0)$ introduced in the previous subsection for $E = (1\ldots1)$. The idea is that modifying non-maximal coordinates of a given vector does not *substantially* affect[4] the distance to the unbiased gradient for the modified vector. From this change on E, we proceed as described in the attack above.

We empirically observe how powerful this attack scenario can become in Section 3.4.2.

3.2.3 Leeway of Attack

In the previous section, we claim that the adversary can build an attack gradient B_t with one unexpectedly large coordinate, and have it pass the Byzantine resilient GAR. Here we try to estimate how large this attacked coordinate can become.

Prior conventions and assumptions.

We will note the arithmetic mean of the non-Byzantine gradients:

$$\overline{G} \triangleq \frac{1}{n-f}\sum_{i=1}^{n-f} G_t^{(i)}$$

[4] It may not affect the infinite norm at all for small-enough γ.

Note that, without loss of generality (since the studied GARs are agnostic to the indexing), the non-Byzantine gradients are indexed between 1 and $n-f$.

Let $e \in [1\,..\,d]$ be the attacked coordinate, and let $E \in \mathbb{R}^d$ be the attack vector such that: $\forall i \in [1\,..\,d]\,,\ E[i] = \mathbf{1}_e(i)$.

Then with $\gamma_m \geq 0$ and $B_t = \overline{G} + \gamma_m\, E$, and $\forall\,(i,j) \in [1\,..\,n-f]^2\,,\ i \neq j$, taking the expectation over the randomness of the non-Byzantine gradient sampling we have:

$$\begin{aligned}
\mathbb{E}\left\|B_t - G_t^{(i)}\right\| &= \mathbb{E}\left\|\overline{G} + \gamma_m\, E - G_t^{(i)}\right\| \\
&= \mathbb{E}\left\|\frac{1}{n-f}\sum_{\substack{j=1\\ j\neq i}}^{n-f}\left(G_t^{(j)} - G_t^{(i)}\right) + \gamma_m\, E\right\| \\
&\leq \frac{1}{n-f}\,\mathbb{E}\left\|\sum_{\substack{j=1\\ j\neq i}}^{n-f}\left(G_t^{(j)} - G_t^{(i)}\right)\right\| + \gamma_m \\
&\leq \frac{1}{n-f}\sum_{\substack{j=1\\ j\neq i}}^{n-f}\mathbb{E}\left\|G_t^{(j)} - G_t^{(i)}\right\| + \gamma_m \\
&\leq \frac{n-f-1}{n-f}\,\mathbb{E}\left\|G_t^{(j)} - G_t^{(i)}\right\| + \gamma_m
\end{aligned}$$

Under the same conditions as above, we also have:

$$\begin{aligned}
\mathbb{E}\left[\left\|B_t - G_t^{(i)}\right\|^2\right] &= \mathbb{E}\left[\left\|\frac{1}{n-f}\sum_{\substack{j=1\\ j\neq i}}^{n-f}\left(G_t^{(j)} - G_t^{(i)}\right) + \gamma_m\, E\right\|^2\right] \\
&= \mathbb{E}\left[\left\|\frac{1}{n-f}\sum_{\substack{j=1\\ j\neq i}}^{n-f}\left(G_t^{(j)} - G_t^{(i)}\right)\right\|^2 + \|\gamma_m\, E\|^2\right. \\
&\qquad \left. + 2\left\langle\frac{1}{n-f}\sum_{\substack{j=1\\ j\neq i}}^{n-f}\left(G_t^{(j)} - G_t^{(i)}\right),\ \gamma_m\, E\right\rangle\right] \\
\text{since } \mathbb{E}\,G_t^{(i)} = \mathbb{E}\,G_t^{(j)} \quad\rightarrow\quad &= \mathbb{E}\left[\left\|\frac{1}{n-f}\sum_{\substack{j=1\\ j\neq i}}^{n-f}\left(G_t^{(j)} - G_t^{(i)}\right)\right\|^2\right] + \|\gamma_m\, E\|^2
\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{(n-f)^2}\,\mathbb{E}\left[\left\|\sum_{\substack{j=1\\j\neq i}}^{n-f}\left(G_t^{(j)}-G_t^{(i)}\right)\right\|^2\right] + {\gamma_m}^2\\
&= \frac{1}{(n-f)^2}\sum_{\substack{j=1\\j\neq i}}^{n-f}\mathbb{E}\left[\left\|G_t^{(j)}-G_t^{(i)}\right\|^2\right] + {\gamma_m}^2\\
&= \frac{n-f-1}{(n-f)^2}\,\mathbb{E}\left[\left\|G_t^{(j)}-G_t^{(i)}\right\|^2\right] + {\gamma_m}^2
\end{aligned}
$$

Attack against *MDA*.

We only study the *worst case* scenario, where $n = 2f+1$, maximizing the proportion of Byzantine workers.

We *expect* all the f submissions of B_t to be selected by *MDA* if, with $\gamma_m \geq 0$:

$$
\begin{aligned}
&\exists\,(i,j) \in [1\,..\,n-f]^2\,,\ i \neq j,\\
&\forall k \in [1\,..\,n-f]\,,\ \mathbb{E}\left\|B_t - G_t^{(k)}\right\| \leq \mathbb{E}\left\|G_t^{(i)} - G_t^{(j)}\right\|\\
\Leftarrow\ &\frac{n-f-1}{n-f}\,\mathbb{E}\left\|G_t^{(i)} - G_t^{(j)}\right\| + \gamma_m \leq \mathbb{E}\left\|G_t^{(i)} - G_t^{(j)}\right\|\\
\Leftarrow\ &\gamma_m \leq \frac{1}{n-f}\,\mathbb{E}\left\|G_t^{(i)} - G_t^{(j)}\right\|
\end{aligned}
$$

This is a sufficient condition *in expectation*, for all the (identical) attack gradients to become aggregated in the output gradient of *MDA*. It is only to give broad insights on the relation between some hyper-parameters and γ_m. In particular, we can reasonably expect $\mathbb{E}\left\|G_t^{(i)} - G_t^{(j)}\right\|$ to grow as $\sqrt{d}$, and so we have with *MDA*: $\gamma_m = \mathcal{O}\left(\sqrt{d}\right)$.

Attack against *Krum*/*Geomed*.

We only study the *worst case* scenario, where $n = 2f+3$, maximizing the proportion of Byzantine workers.
Let $q \in \{\,1,2\,\}$, $q=1$ for *Geomed* and $q=2$ for *Krum*.

Since all the Byzantine submissions are identical, only the neighbor, non-Byzantine gradients will account for non-zero distances in the score of any Byzantine gradient. Namely, since there are $n = 2f+3$ workers and so $n-f-2 = f+1$ neighbors, each Byzantine gradient has two non-Byzantine neighbors. Its expected score is bounded:

$$
\forall k \in [1\,..\,n-f]\,,\ \mathbb{E}[s(B_t)] \leq 2\ \mathbb{E}\left[\left\|B_t - G_t^{(k)}\right\|^q\right]
$$

Since the Byzantine submissions are equal, a non-Byzantine gradient either has no (i.e. 0) Byzantine gradient in its neighbor, or have them all (i.e. f). Thus, $\forall\,(i,j) \in [1\,..\,n-f]^2\,,\ i \neq j$, the *expected* score of the non-Byzantine gradient $G_t^{(i)}$ is either:

$$\mathbb{E}\Big[s\Big(G_t^{(i)}\Big)\Big] = (f+1)\,\mathbb{E}\Big[\Big\|G_t^{(j)} - G_t^{(i)}\Big\|^q\Big]$$

or when all the f identical Byzantine gradients belong to the neighbor of $G_t^{(i)}$:

$$\mathbb{E}\Big[s\Big(G_t^{(i)}\Big)\Big] = f\,\mathbb{E}\Big[\Big\|B_t - G_t^{(i)}\Big\|^q\Big] + \mathbb{E}\Big[\Big\|G_t^{(j)} - G_t^{(i)}\Big\|^q\Big]$$

So we *expect* all f gradients B_t to be selected by *Geomed* if $\gamma_m \geq 0$ both satisfies:

$$\begin{aligned}
&\forall i \in [1\,..\,n-f]\,,\ \mathbb{E}[s(B_t)] \leq \mathbb{E}\Big[s\Big(G_t^{(i)}\Big)\Big] \\
\Leftarrow\ &\begin{cases} 2\ \mathbb{E}\left\|B_t - G_t^{(i)}\right\| \leq (f+1)\,\mathbb{E}\left\|G_t^{(j)} - G_t^{(i)}\right\| \\ 2\ \mathbb{E}\left\|B_t - G_t^{(i)}\right\| \leq f\ \mathbb{E}\left\|B_t - G_t^{(i)}\right\| + \mathbb{E}\left\|G_t^{(j)} - G_t^{(i)}\right\| \end{cases} \\
\Leftarrow\ &\begin{cases} 2\ \left(\frac{n-f-1}{n-f}\,\mathbb{E}\left\|G_t^{(j)} - G_t^{(i)}\right\| + \gamma_m\right) \leq (f+1)\,\mathbb{E}\left\|G_t^{(j)} - G_t^{(i)}\right\| \\ (2-f)\ \left(\frac{n-f-1}{n-f}\,\mathbb{E}\left\|G_t^{(j)} - G_t^{(i)}\right\| + \gamma_m\right) \leq \mathbb{E}\left\|G_t^{(j)} - G_t^{(i)}\right\| \end{cases} \\
\Leftarrow\ &\begin{cases} 2\,\gamma_m \leq \left(f + 1 - 2\,\frac{n-f-1}{n-f}\right)\mathbb{E}\left\|G_t^{(j)} - G_t^{(i)}\right\| \\ (2-f)\,\gamma_m \leq \left((2-f)\left(\frac{1}{n-f} - 1\right) + 1\right)\mathbb{E}\left\|G_t^{(j)} - G_t^{(i)}\right\| \end{cases}
\end{aligned}$$

if $f = 1$:

$$\Leftarrow \gamma_m \leq \frac{1}{n-1}\,\mathbb{E}\left\|G_t^{(j)} - G_t^{(i)}\right\|$$

if $f = 2$:

$$\Leftarrow \gamma_m \leq \left(\frac{1}{2} + \frac{1}{n-2}\right)\mathbb{E}\left\|G_t^{(j)} - G_t^{(i)}\right\|$$

if $f > 2$:

$$\Leftarrow \begin{cases} \gamma_m \leq \left(\frac{f-1}{2} + \frac{1}{n-f}\right)\mathbb{E}\left\|G_t^{(j)} - G_t^{(i)}\right\| \\ \gamma_m \geq \Big(\underbrace{\frac{1}{n-f} - 1 - \frac{1}{f-2}}_{<0}\Big)\,\mathbb{E}\left\|G_t^{(j)} - G_t^{(i)}\right\| \end{cases}$$

The above three cases can actually be joined into:

$$\Leftarrow \gamma_m \leq \left(\frac{f-1}{2} + \frac{1}{n-f}\right)\mathbb{E}\left\|G_t^{(j)} - G_t^{(i)}\right\|$$

And for *Krum*, we *expect* all f gradients B_t to be selected if:

$$\forall i \in [1\,..\,n-f]\,,\ \mathbb{E}[s(B_t)] \leq \mathbb{E}\Big[s\Big(G_t^{(i)}\Big)\Big]$$

$$\Leftarrow \begin{cases} 2\,\mathbb{E}\left[\left\|B_t - G_t^{(i)}\right\|^2\right] \leq (f+1)\,\mathbb{E}\left[\left\|G_t^{(j)} - G_t^{(i)}\right\|^2\right] \\ 2\,\mathbb{E}\left[\left\|B_t - G_t^{(i)}\right\|^2\right] \leq f\,\mathbb{E}\left[\left\|B_t - G_t^{(i)}\right\|^2\right] + \mathbb{E}\left[\left\|G_t^{(j)} - G_t^{(i)}\right\|^2\right] \end{cases}$$

$$\Leftarrow \begin{cases} 2\left(\frac{n-f-1}{(n-f)^2}\,\mathbb{E}\left[\left\|G_t^{(j)} - G_t^{(i)}\right\|^2\right] + {\gamma_m}^2\right) \leq (f+1)\,\mathbb{E}\left[\left\|G_t^{(j)} - G_t^{(i)}\right\|^2\right] \\ (2-f)\left(\frac{n-f-1}{(n-f)^2}\,\mathbb{E}\left[\left\|G_t^{(j)} - G_t^{(i)}\right\|^2\right] + {\gamma_m}^2\right) \leq \mathbb{E}\left[\left\|G_t^{(j)} - G_t^{(i)}\right\|^2\right] \end{cases}$$

$$\Leftarrow \begin{cases} 2\,{\gamma_m}^2 \leq \left(f+1-2\,\frac{n-f-1}{(n-f)^2}\right)\mathbb{E}\left[\left\|G_t^{(j)} - G_t^{(i)}\right\|^2\right] \\ (2-f)\,{\gamma_m}^2 \leq \left((2-f)\left(\frac{1}{(n-f)^2} - \frac{1}{n-f}\right)+1\right)\mathbb{E}\left[\left\|G_t^{(j)} - G_t^{(i)}\right\|^2\right] \end{cases}$$

The same case disjunction made for *Geomed* can be made here, yielding:

$$\Leftarrow {\gamma_m}^2 \leq \left(\frac{f+1}{2} - \frac{1}{n-f} + \frac{1}{(n-f)^2}\right)\mathbb{E}\left[\left\|G_t^{(j)} - G_t^{(i)}\right\|^2\right]$$

$$\Leftarrow \gamma_m \leq \sqrt{\left(\frac{f+1}{2} - \frac{1}{n-f} + \frac{1}{(n-f)^2}\right)\mathbb{E}\left[\left\|G_t^{(j)} - G_t^{(i)}\right\|^2\right]}$$

Assuming that $\sqrt{\mathbb{E}\left[\left\|G_t^{(j)} - G_t^{(i)}\right\|^2\right]}$ will grow as $\sqrt{d}$, we observe that for both *Geomed* and *Krum* the attack coordinate will grow as: $\gamma_m = \mathcal{O}\left(\sqrt[q]{f}\sqrt{d}\right)$. The added dependency in $\sqrt[q]{f}$ compared to *MDA* comes from the structure of the score function in *Geomed* and *Krum*, which decreases the score of the Byzantine gradients as they all are identical.

3.3 Mitigating the Curse

In addition to being Byzantine resilient in the sense that it ensures convergence, our algorithm *Bulyan* also ensures that each coordinate is *agreed on* by a majority of gradient vectors that were *selected* by a Byzantine resilient aggregation rule F. This rule F can for example be *MDA*, *Krum*, *Geomed*, a medoid, and more generally any Byzantine resilient GAR that *designates* at least one input gradient as non-Byzantine.

As *Bulyan* relies on an underlying (α, f)-Byzantine resilient GAR to operate, in this sense we call *Bulyan* a *composite* gradient aggregation rule.

3.3.1 Bulyan: a Composite GAR

Let F be any (α, f)-Byzantine resilient aggregation rule, that requires $n \geq r_F(f)$. *Bulyan* of F requires $n \geq 2f + r_F(f)$ received gradients. For instance with *MDA*, $r_F(f) \triangleq 2f+1$ and so *Bulyan* of *MDA* would require $n \geq 4f+1$.

Bulyan of F operates in two steps. The first step is to *recursively* use F to select $\theta = n - 2f$ gradients, from the set of *received* gradients $\mathcal{R}$ (which initially contains the n input gradients), into a *selection* set $\mathcal{S}$ (initially empty). Namely:

1. With F, choose among $\mathcal{R}$ the next selected vector; for *Krum* we choose the highest scoring gradient, with a medoid the selected gradient would be the output, etc.
2. Remove the chosen gradient from $\mathcal{R}$ and add it into $\mathcal{S}$.
3. Loop back to 1. as long as $|\mathcal{S}| < \theta$.

With $n \geq 2f + r_F(f)$ and $\theta = n - 2f$, we ensure that there always is a *quorum* of workers, i.e. $r_F(f)$, for each iterative use of F.

Since $\theta = n - 2f \geq 2f + 3$, this selection $\mathcal{S} = \{S_1 \dots S_\theta\}$ contains a majority of non-Byzantine gradients. Hence for each $i \in [1 \,..\, d]$, the median of the θ coordinates i of the selected gradients is always bounded by coordinates from non-Byzantine submissions. With $\beta = \theta - 2f \geq 1$, the second step is to *generate* the resulting gradient $G_t = (G_t[1] \dots G_t[d])$, so that for each of its coordinates $G_t[\cdot]$:

$$\forall i \in [1 \,..\, d]\,,\; G[i] = \frac{1}{\beta} \sum_{X \in \mathcal{M}[i]} X[i] \tag{3.2}$$

where:

$$\mathcal{M}[i] = \underset{\mathcal{X} \subset \mathcal{S},\, |\mathcal{X}| = \beta}{\arg\min} \left(\sum_{X \in \mathcal{X}} |X[i] - Median[i]| \right)$$

and:

$$Median[i] = \underset{m = Y[i],\, Y \in \mathcal{S}}{\arg\min} \left(\sum_{Z \in \mathcal{S}} |Z[i] - m| \right)$$

Each i^{th} coordinate of G_t is equal to the average of β of the i^{th} coordinates of the gradients in $\mathcal{S}$, that are closest to the i^{th} coordinate of the median of the gradients in $\mathcal{S}$.

(α, f)-Byzantine resilience of *Bulyan* of F.

The intuition is that, due to the additional requirement of $2f$ non-Byzantine input gradients compared to $r_F(f)$, every gradient selected by F satisfies the requirements for (α, f)-Byzantine resilience. And so will the final aggregation of these gradients.

The formalization[5] is available in the appendix, Section B.1.

[5] This formalization is the work of my co-author, El Mahdi El Mhamdi.

3.3.2 Computational Complexity

Let C be the average computational complexity of running F for each step at the master to aggregate the gradients. Then:

1. The average complexity of *Bulyan* of F is at most: $\mathcal{O}((n-2f)\,C + d\,n)$.
2. In particular, if F is either *Krum* or *Geomed*, this complexity is: $\mathcal{O}(n^2 d)$.

We iterate F as much as $\theta = n - 2f$ times to get the selected vectors, then we run *quick-select* to get each median component ($\mathcal{O}(n)$ on each coordinate, i.e. $\mathcal{O}(d\,n)$ times), and another quick-select to get the β closest coordinates (another $\mathcal{O}(d\,n)$). Hence the average computational complexity of *Bulyan* of F is at most $\mathcal{O}((n-2f)\,C + d\,n)$.

However if we know more about how F is computed in particular cases, we may be able to get rid of the factor $(n-2f)$ when iterating F. Concretely, when F relies on distance computations between the proposed vectors (like with *Krum* and *Geomed*), *Bulyan* does not need to re-compute these distances for F, as the input gradients are the same. Since *Krum* and *Geomed* both have an average complexity of $\mathcal{O}(n^2 d)$, *Bulyan* of these GARs can be optimized to a final average computational complexity of $\mathcal{O}(n^2 d + n\,d)$.

Modern models are notoriously large, and $d \gg n$ holds. Therefore *Bulyan* of *Krum* and *Bulyan* of *Geomed* both run in $\mathcal{O}(n^2 d)$, the same as their respective underlying GARs.

3.4 Practical Evaluations

We implemented the three (α, f)-Byzantine resilient gradient aggregation rules presented in Section 3.1, along with the attack introduced in Section 3.2.2. We report in this section on the actual impact this attack can have, on the MNIST and CIFAR-10 classification datasets, despite the use of such aggregation rules. Then, we evaluate the impact of *Bulyan* compared to these gradient aggregation rules. Finally, we exhibit the cost, in terms of *convergence speed*, of using *Bulyan* in a Byzantine-free setup.

3.4.1 Experimental Settings

We will study the following two models.

MNIST We use a fully connected, feed-forward network with 784 inputs, 1 hidden layer of size 100, for a total of $d \approx 8 \cdot 10^4$ free parameters. The hidden layers use rectified linear units only. The output layer uses *softmax*.

CIFAR-10 We use a convolutional network with the following 7-layers architecture: input $32 \times 32 \times 3$, convolutional (kernel-size: 3×3, 16 maps, 1 stride), max-pooling

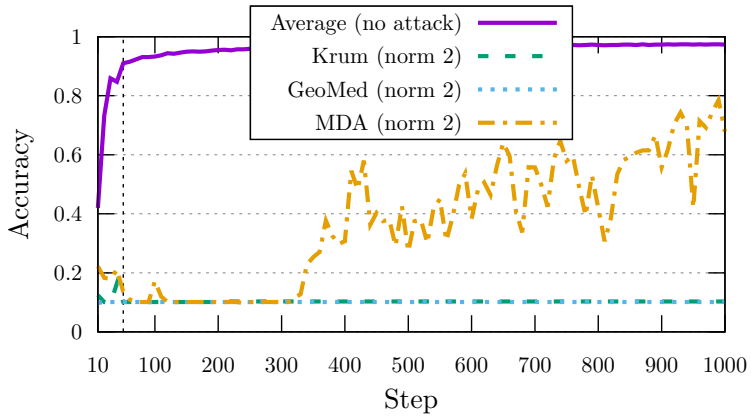


Figure 3.1: MNIST: top-1 cross-accuracy up to step 1000, comparing the presented aggregation rules under our attack. The attack was maintained only up to step 50 (dotted line). The *average* is the reference: it is the accuracy the model would have shown if only non-Byzantine gradients had been selected.

of size 3×3, convolutional (kernel-size: 4×4, 64 maps, 1 stride), max-pooling of size 4×4, two fully connected layers composed of 384 and 192 rectified linear units respectively, and *softmax* is used on the output layer. This model totals 10^6 free parameters. The hidden layers use rectified linear units. The output layer uses *softmax*.

The *maximum cross entropy* loss function is used for both models. L2-regularization of value 10^{-4} is used for both models, and both use the Xavier weight initialization algorithm. We use a fading learning rate $\eta_{step} = \eta_0 \frac{r_\eta}{step+r_\eta}$. The initial learning rate η_0, the fading rate r_η, and the mini-batch size depend on each experiment.

We use $E = (0 \ldots 0, 1)$, attacking the last coordinate, which corresponds to the bias of the last linear layer of the model. The accuracy is always measured on the *testing set*.

We will use *Krum* with *Bulyan* throughout the experiments. The resulting (α, f)-Byzantine resilient rule *Bulyan* of *Krum* will simply be called *Bulyan* in the following.

3.4.2 Experimental Results

Attack on *MDA*, *Krum* and *Geomed*.

Figures 3.3 and 3.4 shows the impact of our attack on the aggregation rules presented in Section 3.1. The *average* rule is the arithmetic mean of its n input gradients.

On MNIST, we use $\eta_0 = 1$, $r_\eta = 10000$, a batch size of 83 images (256 for *MDA*), and for the worker counts:

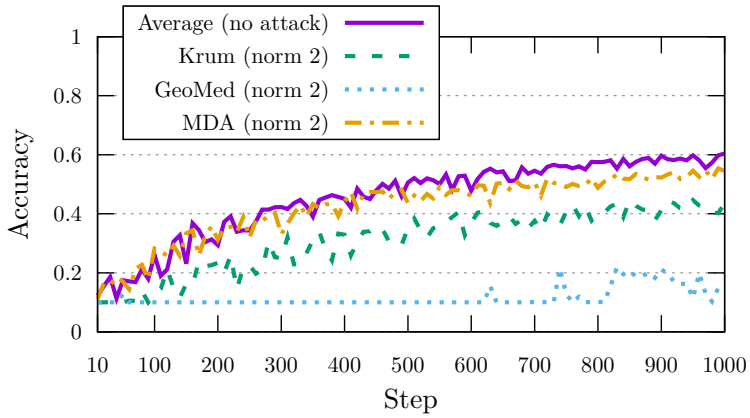


Figure 3.2: CIFAR-10: top-1 cross-accuracy up to step 1000, comparing the presented aggregation rules under our attack. The *average* is the reference: it is the accuracy the model would have shown if only non-Byzantine gradients had been selected.

Krum/*Geomed*	30 non-Byzantines + 27 Byzantines
MDA	6 non-Byzantines + 5 Byzantines
Average	30 non-Byzantines + 0 Byzantines

On CIFAR-10, we use $\eta_0 = 0.1$, $r_\eta = 2000$, a batch size of 128 images (256 for *MDA*), and for the worker counts:

Krum/*Geomed*	21 non-Byzantines + 18 Byzantines
MDA	6 non-Byzantines + 5 Byzantines
Average	21 non-Byzantines + 0 Byzantines

In Figure 3.3, the attack is maintained only up to 50 steps. As shown, and except for *MDA*, this short attack phase at the beginning of the learning process is sufficient to put the parameter vector in a sub-space of *ineffective* models that SGD did not succeed in leaving for at least 950 steps. In Figure 3.4, the attack is never stopped. Only *MDA* preserved the accuracy. *Krum* suffered a 33% decrease at step 1000, and *Geomed* failed to produce a useful model.

Higher learning rates and lower batch sizes increase the impact of our attack, by boosting both its *exploratory* capabilities and the variance of the non-Byzantine gradients.

Attack on *MDA*, *Krum* and *Geomed*: the case of the infinite norm.

On MNIST, here we use $\eta_0 = 1$, $r_\eta = 10000$, a batch size of 83 images (256 for *MDA*), and for the workers:

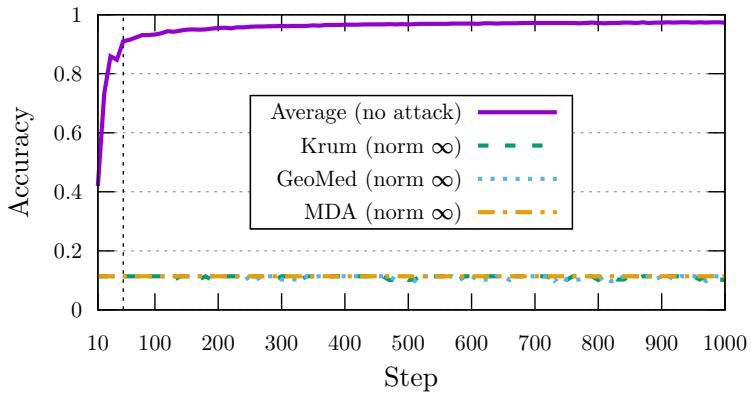


Figure 3.3: MNIST: top-1 cross-accuracy up to step 1000, comparing the presented aggregation rules under our attack. The attack was maintained only up to step 50 (dotted line). The *average* is the reference: it is the accuracy the model would have shown if only non-Byzantine gradients had been selected.

Krum/*Geomed*	30 non-Byzantines + 27 Byzantines
MDA	6 non-Byzantines + 5 Byzantines
Average	30 non-Byzantines + 0 Byzantines

In Figure 3.3, the attack is maintained only up to 50 steps. The attack variant for ℓ_∞ norm-based gradient aggregation rules exhibited a very strong impact. None of the presented defenses prevented the stochastic gradient descent from being *pushed* and remaining in a sub-space of *ineffective* models, and for at least 1000 steps.

The effect of *Bulyan*.

Figures 3.5 and 3.6, respectively for MNIST and CIFAR-10, compares *Krum*, *Geomed* and *Bulyan*.

On MNIST, we use $\eta_0 = 1$ ($\eta_0 = 0.2$ for the upper graph), $r_\eta = 10000$, and a mini-batch size of 83 images. On CIFAR-10, we use $\eta_0 = 0.25$, $r_\eta = 2000$, and a mini-batch size of 128 images. For both MNIST and CIFAR-10, we use 30 non-Byzantines + 9 Byzantines workers. *MDA* cannot be used with that many workers, see Section 3.1.2.

In Figure 3.5, with $\eta_0 = 1$, *Krum* and *Geomed* fail to prevent the attack from *pushing* the model into an *ineffective* state, despite the reduced proportion of Byzantine workers from roughly $1/2$, in Figure 3.3, to roughly $1/4$. With $\eta_0 = 0.2$, *Krum* and *Geomed* support the attack, at the cost of a *uselessly* slower learning process. Here, *Bulyan* is not affected by the attack, and achieves the same accuracy *as if* it averages only the non-Byzantine gradients. In Figure 3.6, we do the same experiment with CIFAR-10. As with MNIST, only *Bulyan* is not affected by our attack.

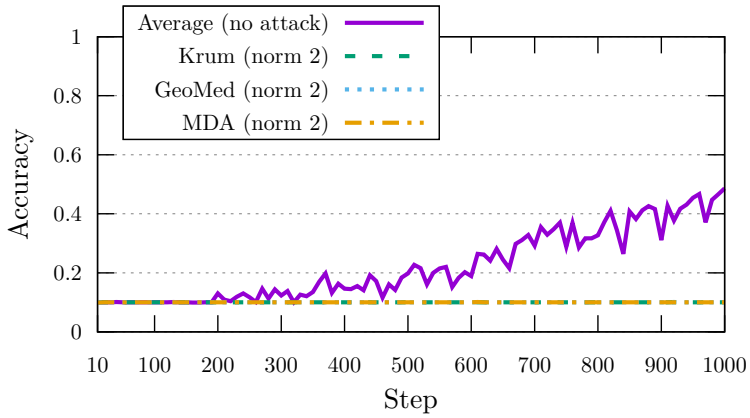


Figure 3.4: CIFAR-10: top-1 cross-accuracy up to step 1000, comparing the presented aggregation rules under our attack. The *average* is the reference: it is the accuracy the model would have shown if only non-Byzantine gradients had been selected.

The cost of *Bulyan*.

In Figure 3.7, we study the cost of using *Bulyan*, in terms of *convergence speed*, when there is actually no adversary. We define the *convergence speed*, for a given mini-batch size, as the accuracy the model reaches at a fixed, arbitrary step. We use the *average*, i.e. the arithmetic mean of the submitted gradients, as the reference aggregation rule.

Without Byzantine workers, the loss in convergence speed induced by *Bulyan* is minimized with a *reasonable* batch size: 24 images/batch for MNIST, and 36 for CIFAR-10.

3.5 Concluding Remarks

In this chapter, we presented a simple attack that can defeat stateless, (α, f)-Byzantine resilient defenses based on a *distance minimization scheme*. The core enabling cause is the so-called *curse of dimensionality*, in particular with respect to what information the ℓ_2 distance between two gradients really provides. Namely with high dimensional $d \gg 1$ machine learning models, for a gradient to be close to another one does not constrain much each of their coordinates: these two gradients may "disagree" *a bit* on each of their coordinates, or "disagree" *a lot* on only a few of them. This offers the attacker an opportunity to inject f Byzantine gradients, all impacting the aggregated gradient, with one fairly large coordinate, as large as $\mathcal{O}\left(f\sqrt{d}\right)$ in the case of *Geomed*. As we have observed in our experiments, this simple attack can have devastating effects.

We call this opportunity the *leeway of attack*. As our theoretical development shows conspicuously, the leeway of attack is independent from the coordinate the adversary chose to attack. This is problematic. In a neural network for instance, the coordinates of the parameter vector are not equivalent, and none of the studied GARs take that fact

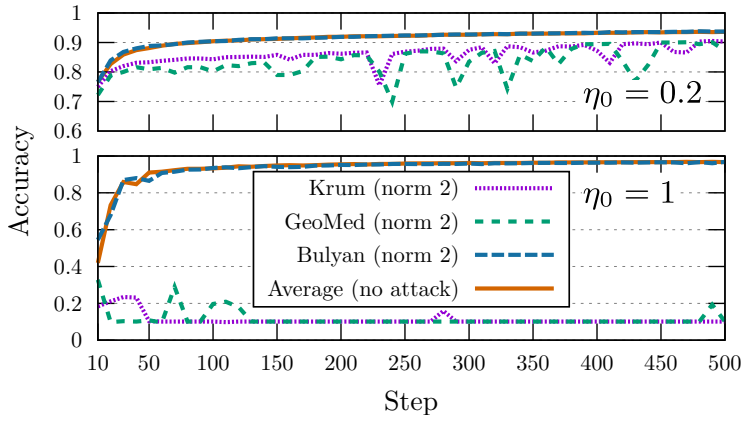


Figure 3.5: MNIST: top-1 cross-accuracy up to 500 steps for *Krum*, *Geomed*, *Bulyan* rules. This graph illustrates the impact of the learning rate, as described in Section 3.2.1.

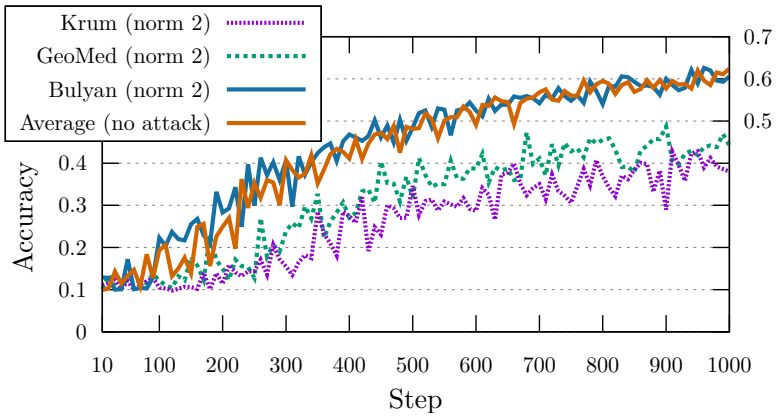


Figure 3.6: CIFAR-10: top-1 cross-accuracy set up to 1000 steps for *Krum*, *Geomed*, *Bulyan* rules. The arithmetic mean of non-Byzantine gradients serves as reference.

into account. One possible future work would then be to design a GAR aware of each coordinate *role*, for instance by weighting each coordinate differently when computing a distance. There would be challenges ahead: how to formalize each coordinate *role*, which complexity would computing this *role* have (e.g. computing a Hessian in $\mathcal{O}(d^2)$ would not be practical), and how do the *roles* of each coordinate change over time (allowing the adversary to attack coordinates just before they become more influential).

In the meantime, the *composite* gradient aggregation rule we propose, *Bulyan*, offers a pragmatic and computationally efficient response to the family of attack we exposed.

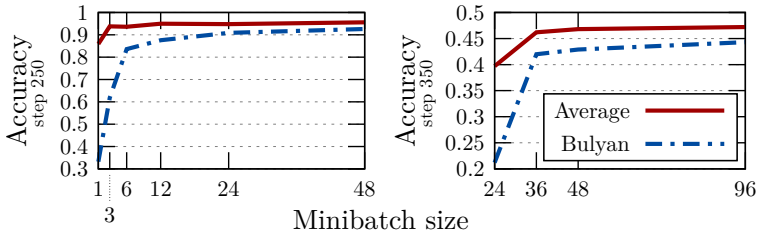


Figure 3.7: MNIST (left), CIFAR-10 (right): top-1 cross-accuracy at step 250 (left) and 350 (right) for *Average* and *Bulyan*. There are $n = 39$ workers and no adversary, but f is declared to 9. This shows the trade-off between the Byzantine robustness of *Bulyan* and the loss in convergence speed it introduces.

Addressing Shortcomings Part II

4 (No) Single Point of Failure

The Byzantine fault model, as originally introduced in distributed computing (Lamport et al., 1982), encompasses crashes, software bugs, hardware defects, message omissions, corrupted data, as well as compromised machines (Biggio et al., 2012; Xiao et al., 2015). Previous work (Alistarh et al., 2018; Xie et al., 2018a; Blanchard et al., 2017; Yin et al., 2018; El-Mhamdi et al., 2018; Xie et al., 2018c) on Byzantine resilient SGD assumed that a fraction of the workers could be Byzantine. But all also assumed the central parameter server (Figure 2.2) always remains *honest* and *failure-free*. In other words, none of the previous approaches considered a genuinely Byzantine resilient distributed setting, where no component is trusted.

We propose a distributed algorithm solving the *general* Byzantine resilient distributed stochastic gradient descent problem, where no individual component is trusted.

4.1 Motivation

Consider a multi-branch organization with sensitive data and needs for high availability, e.g. a hospital or a bank. The failure of the central server, be it because of a crash or malicious attack, could have severe consequences for such critical organizations: from the unavailability of the service to compromised trained models affecting its users. A common and generic strategy to avoid the parameter server from becoming a *single point-of-failure* is to replicate it. The remaining challenge is then to keep the respective states of these many replicas somehow *synchronized*.

The classical synchronization technique, *State Machine Replication* (SMR), enforces a *total order on state changes* for all the replicas through *consensus* (Schneider, 1990), allowing all the replicated states to undergo the exact same updates (and thus remain synchronized). Such an approach would provide the abstraction of a single parameter server, allowing to keep using Byzantine resilient GARs as with one parameter server, while benefiting from the resilience of many replicas. Applying SMR to distributed

SGD would however lead to a potentially huge *overhead*. In order to maintain the same state, replicas would need to agree on a total order of the model updates, inducing frequent exchanges (and especially retransmissions) of gradients and parameter vectors, that can easily be hundreds of megabytes large (Kim, 2012). Given that distributed stochastic gradient descent setups are network-bound (Zhang et al., 2017; Hsieh et al., 2017), SMR appears impractical in this context.

The key insight underlying this work is that the *general* Byzantine SGD problem, even when neither the workers nor the servers are trusted, is easier than consensus; and so total ordering of updates is not required for SGD applications. The final parameter obtained with many parameter servers need only be close to the one obtained in the single parameter server case. We thus follow a different path where we do not require all the servers to maintain the same state. Instead, we allow mildly diverging parameters (which have proven beneficial in other contexts (Zhang et al., 2015; Alistarh et al., 2016)) and present a new way to *contract* them in a distributed manner.

4.1.1 The Case for Asynchrony

The additional need for the parameter servers to synchronize would also increase the attack surface available to the adversary. As Cachin et al. (2011) notes it: "an important part in the characterization of a distributed system is the behavior of its processes and links with respect to the passage of time". Distributed algorithms can either be:

Synchronous The algorithm assumes there is a *known* upper bound delay on both processing and communication. The correctness of the algorithm is not guaranteed if the delay happens to be violated in practice.

Partially synchronous The algorithm makes progress *only* when a known upper bound delay on both processing and communication is upheld, but is built to maintain consistency even when the delay upper bound is violated in practice.

Asynchronous No timing assumption is made and needed to ensure the correctness and actual *progress* of the algorithm. In particular, asynchronous consensus algorithms are impossible under Byzantine failures (Fischer et al., 1985).

Introducing delays in communication and processing is most certainly within the reach of a real adversary. Based on overwhelming a service with requests, *Distributed Denial of Service* (DDoS) attacks are for instance able to slow down a target, possibly to the point of rendering the service effectively inaccessible. Attacking one of the parameter server this way (or any router between parameter servers) appears to be a plausible attack, that would disrupt *synchronous* algorithms. Under the same circumstances, *partially synchronous* algorithms should not introduce inconsistencies

that the adversary can leverage. Nevertheless, the progress of such algorithms would be delayed until a phase of synchrony (Cachin et al., 2011); delay which the adversary controls. So under an adversary capable of inducing delays (which is very reasonable given enough resources), a partially synchronous algorithm may exchange infinitely many messages between non-Byzantine nodes, and still not make any progress.

As long as messages are exchanged between non-Byzantine nodes, an *asynchronous* algorithm would keep making progress. We thus choose to design a distributed, asynchronous algorithm to tackle the problem of general Byzantine resilient SGD.

4.1.2 Updated Distributed Model

There is one change to make compared to the original distributed model presented in Section 2.1.2: instead of one parameter server, there are $n_{ps} \geq 1$ parameter servers.

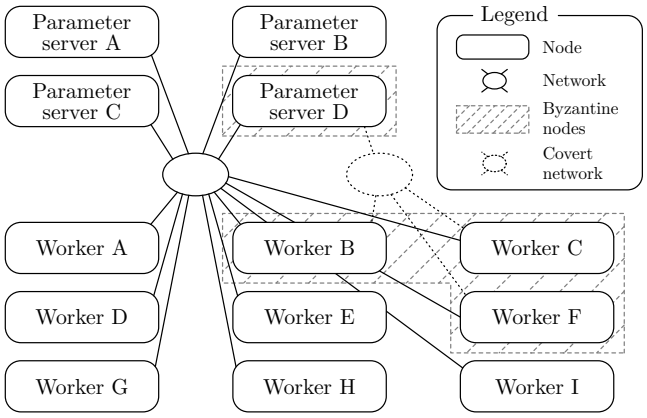


Figure 4.1: $n_{ps} = 4$ parameter servers and $n_w = 9$ workers, including respectively $f_{ps} = 1$ and $f_w = 3$ Byzantine nodes, which can be viewed together as a single adversary.

Figure 4.1 provides an overview of the new model. The objective of the adversary and the threat model both remain identical as the ones presented in Section 2.2. The Byzantine parameter servers and the Byzantine workers can also collude.

The notations are updated accordingly:

Symbol	Description
n_{ps}	Total number of parameter servers in the distributed deployment
f_{ps}	Maximum number of Byzantine parameter servers among the n_{ps} servers
n_w	Total number of workers in the distributed deployment
f_w	Maximum number of Byzantine workers among the n_w workers
$\theta_t^{(i)}$	Parameter vector held by parameter server i at step t

Table 4.1: Updated notations over Table 1.1 for Chapter 4 specifically.

4.2 ByzSGD: General Byzantine SGD

We present here *ByzSGD*, historically the first algorithm to tolerate Byzantine workers and servers without making any assumptions on node relative speeds and communication delays. *ByzSGD* does not add, on average, any communication rounds compared to the standard parameter server communication model. However, periodically[1], *ByzSGD* adds a communication round between the parameter servers to eventually enforce contraction and convergence.

We will first describe the distributed algorithm *ByzSGD*. We will then present its operating assumptions, and finally provide a formal proof of convergence.

4.2.1 Distributed Algorithm

The particularity of our algorithm is that it uses two aggregation rules: one Byzantine resilient rule to aggregate the gradients (as in the single server setting), and one *contraction* rule to bring back parameter vectors closer to each other.

One of the main challenges arising with multiple parameter servers is that nothing prevents the non-Byzantine parameters $\theta_t^{(1)} \dots \theta_t^{(n-f)}$ to drift away from each other as t grows. Parameter drifts can be mitigated without Byzantine nodes, by using a *central authority* which pulls back together the diverging parameters using an *elastic force* (Zhang et al., 2015). With *general* Byzantine faults, there can be no central authority anymore.

Asynchrony and Byzantine faults not only prevent consensus (Fischer et al., 1985) and incidentally SMR, but also prevent non-Byzantine nodes from waiting for *every other* non-Byzantine nodes. Indeed: due to asynchrony, a slow-to-respond non-Byzantine node is indistiguishable from a non-responding Byzantine node. For instance if there are n nodes among which f are Byzantine, if a non-Byzantine node waits for at least $n - f$ *other* nodes, then this non-Byzantine node will necessarily wait for at least one of the Byzantine nodes. Consequently, if the f Byzantine nodes all decide never to respond, the distributed algorithm will remain blocked forever[2].

The challenging question can then be formulated as follows: given that the non-Byzantine parameter servers should not expect to receive more than $n - f - 1$ messages

[1] Our experimental assessment suggests this additional communication round may happen infrequently, e.g. once every 333 training steps yielded satisfying performances in practice (Section 4.3.2). While the algorithm was originally my contribution, the idea to amortize the communication round between parameter servers every T steps, instead of having it every step, was the idea of my co-authors.

[2] Remember that asynchrony forbids the non-Byzantine node from setting a *deadline* after which the algorithm would continue with the received responses. If the non-Byzantine node was waiting for q responses for at most Δt, due to asynchrony, it is possible that the non-Byzantine node receives e.g. only f responses, all coming only from the f Byzantine nodes. In asynchrony, there can be no such deadline.

per round, how to keep the non-Byzantine parameters close to each other, knowing that a fraction of the q received messages could be Byzantine?

Distributed Median-based Contraction.

Our solution to this issue is what we call *Distributed Median-based Contraction* (DMC). Once every T steps, the parameter servers broadcast to each other their respective parameter vectors $\theta_t^{(1)} \dots \theta_t^{(n_{ps})}$. The goal of DMC is to decrease the expected maximum distance between any two honest parameter vectors, hence the term *contraction*.

DMC is a combination of:

- the application of the *coordinate-wise median* (*Median*) on the parameter vectors,
- and *provisionning* enough non-Byzantine servers so that each non-Byzantine server can expect to collect the messages from at least $2f_{ps} + 1$ other parameter servers (i.e. due to asynchrony, the following has to hold: $n_{ps} \geq 3f_{ps} + 2$).

These two points constitute the root of what we call the *contraction effect*. In a nutshell, this contraction effect derives from the fact that, during the contraction step made every T steps, each non-Byzantine server will aggregate (with the *Median*) at least $2f_{ps}+2$ parameter vectors[3]. Since there is a strict majority of non-Byzantine parameter vectors $n_{ps} \geq 2f_{ps} + 1$, each coordinate of the coordinate-wise median will then be bounded between coordinates of the non-Byzantine parameter vectors. And since there is at least one additional non-Byzantine parameter vector $n_{ps} \geq (2f_{ps} + 1) + 1$ over a strict majority, taken over which random subset of non-Byzantine parameter vectors was actually received, the expected maximum distance between any two non-Byzantine parameter vectors is thus decreased after applying the DMC.

The algorithms for both non-Byzantine workers and non-Byzantine parameter servers are detailed in Figure 4.2. Please note that the initial (non-Byzantine) parameter vector is randomly selected using a common seed, so that each non-Byzantine parameter server j starts the algorithm with the same initial parameter vector $\theta_0^{(j)}$.

4.2.2 Operating Assumptions

Since the parameter server uses *MDA* (c.f. Algorithm 2), each parameter server needs to collect $q_w \geq 2f_w + 1$ gradients per step. And so, due to asynchrony, there must be at least $n_w \geq q_w + f_w \geq 3f_w + 1$ workers, among which up to f_w can be Byzantine.

[3]Indeed: each non-Byzantine parameter server i collected at least $2f_{ps} + 1$ other parameter vectors. So adding its *own* parameter vector $\theta_t^{(i)}$ the server will aggregate at least $2f_{ps} + 2$ parameter vectors.

Data: *max_steps*
$t \leftarrow 0$;
while $t <$ *max_steps* **do**
  $\theta_t^{(x)} \dots \theta_t^{(y)} \leftarrow$ receive_from_servers();
  $\overline{\theta}_t^{(i)} \leftarrow \text{Median}\left(\theta_t^{(x)} \dots \theta_t^{(y)}\right)$;
  $g_t^{(i)} \leftarrow \text{estimate\_gradient}\left(\overline{\theta}_t^{(i)}\right)$;
  broadcast_to_servers$\left(g_t^{(i)}\right)$;
  $t \leftarrow t + 1$;
end

Algorithm 1: Worker i

Data: *max_steps, T, seed*
$\theta_0^{(j)} \leftarrow$ seed_parameters(*seed*);
$t \leftarrow 0$;
while $t <$ *max_steps* **do**
  broadcast_to_workers$\left(\theta_t^{(j)}\right)$;
  $G_t^{(x)} \dots G_t^{(y)} \leftarrow$ receive_from_workers();
  $\overline{G}_t^{(j)} \leftarrow \text{MDA}\left(G_t^{(x)} \dots G_t^{(y)}\right)$;
  $\theta_t^{(j)} \leftarrow \theta_t^{(j)} - \eta_t \overline{G}_t^{(j)}$;
  $t \leftarrow t + 1$;
  if $t \bmod T = 0$ **then**
    broadcast_to_servers$\left(\theta_t^{(j)}\right)$;
    $\theta_t^{(x)} \dots \theta_t^{(y)} \leftarrow$ receive_from_servers();
    $\theta_t^{(j)} \leftarrow \text{Median}\left(\theta_t^{(x)} \dots \theta_t^{(y)}\right)$;
  end
end

Algorithm 2: Parameter server j

Figure 4.2: *ByzSGD*: worker and parameter server algorithms. receive_from_workers() always waits for $n_w - f_w$ gradients. On a worker, receive_from_servers() waits for $n_{ps} - f_{ps}$ parameter vectors. On a server, receive_from_servers() waits for $n_{ps} - f_{ps} - 1$ other parameter vectors, as the server's own parameter vector is always delivered.

A similar reasoning applies for n_{ps}. For DMC to work, each parameter server must collect at least $q_{ps} \geq 2f_{ps} + 2$ parameter vectors (including their own) per contraction step. And so, due to asynchrony, there must be at least $n_{ps} \geq q_{ps} + f_{ps} \geq 3f_{ps} + 2$ workers, among which up to f_{ps} can be Byzantine.

To prove the convergence of the non-Byzantine parameter vectors, we assume that:

1. $\forall t \in \mathbb{N},\ g_t^{(1)} \dots g_t^{(n_w - f_w)}$ are mutually independent.
2. $\exists \sigma' \in \mathbb{R}_+,\ \forall (i,t) \in [1 \,..\, n_w - f_w] \times \mathbb{N},\ \mathbb{E}\left\|g_t^{(i)} - \mathbb{E}\, g_t^{(i)}\right\| \leq \sigma'$.
3. Q is positive, and 3-times differentiable with continuous derivatives.
4. $\forall r \in [2 \,..\, 4],\ \exists (A_r, B_r) \in \mathbb{R}^2,\ \forall (i,\theta) \in [1 \,..\, n_w - f_w] \times \mathbb{R}^d$, $g \triangleq \text{estimate\_gradient}(\theta),\ \mathbb{E}\|g\|^r \leq A_r + B_r \|\theta\|^r$.
5. ∇Q is Lipschitz continuous, i.e.: $\exists l > 0,\ \forall (x,y) \in \left(\mathbb{R}^d\right)^2$, $\|\nabla Q(x) - \nabla Q(y)\|_1 \leq l \|x - y\|_1$
6. $\exists D \in \mathbb{R},\ \forall \theta \in \mathbb{R}^d,\ \|\theta\|^2 \geq D,\ \exists (\varepsilon, \beta) \in \mathbb{R}_+ \times [0, \frac{\pi}{2}[$, $\|\nabla Q(\theta)\| \geq \varepsilon,\ \langle \theta, \nabla Q(\theta) \rangle \geq \cos(\beta)\ \|\theta\|\ \|\nabla Q(\theta)\|$.

7. ∇q is "almost Lipschitz continuous", i.e.: $\exists l' > 0,\ \forall (x,y) \in \left(\mathbb{R}^d\right)^2$, $\mathbb{E}_{z\sim\mathcal{D}} \left\|\nabla q(x,z) - \nabla q(y,z)\right\|_1 \leq l' \left\|x-y\right\|_1$
8. $\exists \kappa \in \left]n_{ps}, +\infty\right[,\ \forall (i,t,\theta) \in [1\,..\,n_w - f_w] \times \mathbb{N} \times \mathbb{R}^d$, $\kappa \frac{\sqrt{8}\,f_w}{n_w - f_w} \sqrt{\mathbb{E}\left[\left\|\text{estimate\_gradient}(\theta) - \nabla Q(\theta)\right\|^2\right]} \leq \left\|\nabla Q(\theta)\right\|$.
9. Let[4] $S_j \triangleq \{s \subset [1\,..\,n_{ps} - f_{ps}] - \{j\} \mid |s| \in [q-f-1\,..\,q-1]\}$ the subset of non-Byzantine indexes that node j will receive at any given step. Let $S = \prod_{j\in[1\,..\,n_{ps}-f_{ps}]} S_j$ the set of all non-Byzantine parameter server indexes the $n_{ps} - f_{ps}$ non-Byzantine parameter servers can receive. We assume that: $\exists \rho > 0,\ \forall s \in S,\ P\left(X = s\right) \geq \rho$.

The assumptions made in Point 1 to Point 5 (i.i.d., bounded variance, loss differentiability, bounded statistical moments, and gradient Lipschitz continuity) are the most common ones in classical SGD analysis (Bottou, 1998; Bousquet and Bottou, 2008).

The assumption made in Point 8 is inherited from the theoretical requirements for the (α, f)-Byzantine resilience of *MDA*. $\kappa > n_{ps}$ is a safe bound to eventually tackle both honest gradients being estimates at slightly different parameter vectors, and a *worst-case* scenario where the adversary could push parameters in the wrong direction.

The assumption made in Point 6 was first adapted from Bottou (1998) by Blanchard et al. (2017) to account for Byzantine resilience. It intuitively says that beyond a certain horizon, the loss function is "steep enough" (it has lower bounded gradient norms) and "convex enough" (it has upper bounded angles between the gradients and the parameter vectors). The loss function does not need to be convex: adding a ℓ_2 regularization term to the loss is enough to guarantee this assumption in practice. Indeed, when $\|\theta_t\|$ goes to infinity, the regularization eventually dominates the rest of the loss function and permits the gradient $\nabla Q(\theta_t)$ to point to the same half space as θ_t.

4.2.3 Proof of Convergence

Without loss of generality, let $(1, \ldots, n_w - f_w)$ be the indices of the non-Byzantine workers, and let $(1, \ldots, n_{ps} - f_{ps})$ be the indices of the non-Byzantine parameter servers.

Noting $\theta_t \triangleq \textit{Median}\left(\theta_t^{(1)} \ldots \theta_t^{(n_{ps})}\right)$, we prove that:

$$\lim_{t\to+\infty} \mathbb{E}\left\|\nabla Q(\theta_t)\right\| = 0 \tag{4.1}$$

The expected values are taken over the randomness of both the datapoints sampled by each non-Byzantine worker and the subset of non-Byzantine messages received by each non-Byzantine node, at step t and every previous steps.

[4] This is a improved formulation by Lê Nguyen Hoang over the original formulation of the author.

Lemma 1. *We derive a sufficient condition for Equation* (4.1)*. Namely, we observe that:*

$$\left.\begin{array}{r} \lim\limits_{t\to+\infty} \mathbb{E}\left[\max\limits_{(a,b)\in[1\,..\,n_{ps}-f_{ps}]^2} \left\|\theta_t^{(a)}-\theta_t^{(b)}\right\|\right]=0 \\ \lim\limits_{t\to+\infty} \mathbb{E}\left\|\nabla Q\left(\theta_t^{(1)}\right)\right\|=0 \end{array}\right\} \implies (4.1)$$

The first expression is proven in Lemma 5, and the second one is proven in Lemma 6.

Proof. Indeed, by one of the triangular inequalities and Lipschitz continuity, we have:

$$\forall t\in\mathbb{N},\ \left|\|\nabla Q(\theta_t)\|-\left\|\nabla Q\left(\theta_t^{(1)}\right)\right\|\right|\leq\left\|\nabla Q(\theta_t)-\nabla Q\left(\theta_t^{(1)}\right)\right\|\leq l\left\|\theta_t-\theta_t^{(1)}\right\|$$

Hence, using $|A-B|\leq C\Rightarrow B-C\leq A\leq B+C$, we get:

$$\mathbb{E}\left\|\nabla Q\left(\theta_t^{(1)}\right)\right\|-l\,\mathbb{E}\left\|\theta_t-\theta_t^{(1)}\right\|\leq\mathbb{E}\left\|\nabla Q(\theta_t)\right\|\leq\mathbb{E}\left\|\nabla Q\left(\theta_t^{(1)}\right)\right\|+l\,\mathbb{E}\left\|\theta_t-\theta_t^{(1)}\right\| \quad (4.2)$$

We now recall that, by the coordinate-wise construction of $\theta_t\triangleq \textit{Median}\left(\theta_t^{(1)}\dots\theta_t^{(n_{ps})}\right)$:

$$\forall i\in[1\,..\,d],\ \left|\theta_t[i]-\theta_t^{(1)}[i]\right|\leq\max_{(a,b)\in[1\,..\,n_{ps}-f_{ps}]^2}\left|\theta_t^{(a)}[i]-\theta_t^{(b)}[i]\right|$$

Hence, observing that:

$$\begin{aligned}
&\max_{(a,b)\in[1\,..\,n_{ps}-f_{ps}]^2}\left\|\theta_t^{(a)}-\theta_t^{(b)}\right\|_1\\
&\leq\sum_{i=1}^{d}\left(\max_{(a,b)\in[1\,..\,n_{ps}-f_{ps}]^2}\left|\theta_t^{(a)}[i]-\theta_t^{(b)}[i]\right|\right)\\
&\leq\frac{(n_{ps}-f_{ps})\,(n_{ps}-f_{ps}-1)}{2}\max_{(a,b)\in[1\,..\,n_{ps}-f_{ps}]^2}\left(\underbrace{\sum_{i=1}^{d}\left|\theta_t^{(a)}[i]-\theta_t^{(b)}[i]\right|}_{\left\|\theta_t^{(a)}-\theta_t^{(b)}\right\|_1}\right) \qquad (4.3)
\end{aligned}$$

The equivalence of norms $\|\cdot\|_2\leq\|\cdot\|_1\leq\sqrt{d}\,\|\cdot\|_2$ yields here:

$$\mathbb{E}\left\|\theta_t-\theta_t^{(1)}\right\|\leq\frac{\sqrt{d}\,(n_{ps}-f_{ps})\,(n_{ps}-f_{ps}-1)}{2}\,\mathbb{E}\left[\max_{(a,b)\in[1\,..\,n_{ps}-f_{ps}]^2}\left\|\theta_t^{(a)}-\theta_t^{(b)}\right\|\right] \quad (4.4)$$

Finally, Equation (4.4) and $\lim\limits_{t\to+\infty}\mathbb{E}\left[\max\limits_{(a,b)\in[1\,..\,n_{ps}-f_{ps}]^2}\left\|\theta_t^{(a)}-\theta_t^{(b)}\right\|\right]=0$ yield:

$$\lim_{t\to+\infty}\mathbb{E}\left\|\theta_t-\theta_t^{(1)}\right\|=0 \quad (4.5)$$

Combining Equation (4.5) with Equation (4.2) and $\lim_{t\to+\infty} \mathbb{E}\left\|\nabla Q\left(\theta_t^{(1)}\right)\right\| = 0$ concludes:

$$\lim_{t\to+\infty} \mathbb{E}\left\|\nabla Q(\theta_t)\right\| = 0$$

□

Lemma 2. *Convergence of* $\sum_{i=0}^{n} k^{n-i}\,\eta_i$.

Let $0 < k < 1$ *and* $\eta_i > 0$ *be the general term of a sequence such that* $\lim_{i\to+\infty} \eta_i = 0$.

Then:
$$\lim_{n\to+\infty} \left(\sum_{i=0}^{n} k^{n-i}\,\eta_i\right) = 0$$

Proof. First, we observe that:

$$\begin{aligned}\lim_{t\to+\infty} \eta_t = 0 &\Rightarrow \forall \varepsilon > 0,\ \exists \tau \in \mathbb{N},\ \forall t \geq \tau,\ \eta_t < \varepsilon \\ &\Rightarrow \exists \tau \in \mathbb{N},\ \forall t \geq \tau,\ \eta_t < 1 \end{aligned} \tag{4.6}$$

Then, reusing τ from (4.6), it holds $\forall n \geq 2\,\tau$:

$$\begin{aligned}\sum_{i=0}^{n} k^{n-i}\,\eta_i &= \sum_{i=0}^{\tau-1} k^{n-i}\,\eta_i + \sum_{i=\tau}^{\lfloor n/2\rfloor} k^{n-i}\,\eta_i + \sum_{i=\lfloor n/2\rfloor+1}^{n} k^{n-i}\,\eta_i \\ &< k^{n-\tau}\left(\sum_{i=0}^{\tau-1} k^{\tau-i}\,\eta_i\right) + \sum_{i=\tau}^{\lfloor n/2\rfloor} k^{n-i} + \sum_{i=\lfloor n/2\rfloor+1}^{n} k^{n-i}\,\eta_i \\ &< k^{n-\tau}\left(\sum_{i=0}^{\tau-1} k^{\tau-i}\,\eta_i\right) + k^{\lceil n/2\rceil} \sum_{i=\tau}^{\lfloor n/2\rfloor} k^{\lfloor n/2\rfloor-i} + \max_{i\in\left[\frac{n}{2}+1\,..\,n\right]}(\eta_i) \sum_{i=\lfloor n/2\rfloor+1}^{n} k^{n-i} \\ &< k^{n-\tau}\left(\sum_{i=0}^{\tau-1} k^{\tau-i}\,\eta_i\right) + \frac{1}{1-k}\left(k^{\lceil n/2\rceil} + \max_{i\in\left[\frac{n}{2}+1\,..\,n\right]}(\eta_i)\right)\end{aligned}$$

Finally, since $0 < k < 1$ and $\lim_{n\to+\infty}\left(\max_{i\in\left[\frac{n}{2}+1\,..\,n\right]}(\eta_i)\right) = \lim_{n\to+\infty}(\eta_n) = 0$, we conclude:

$$\lim_{n\to+\infty} \left(\sum_{i=0}^{n} k^{n-i}\,\eta_i\right) = 0$$

□

Lemma 3. *MDA bounded deviation from majority.*

Let $(d, f) \in (\mathbb{N} - \{0\})^2$, *let* $q \in \mathbb{N}$ *such that* $q \geq 2\,f + 1$.

Let note $H_1 \triangleq [1\,..\,q - f]$ *and* $H_2 \triangleq [q + 1\,..\,2\,q - f]$.

We will show that, $\forall p \in \mathbb{N}_+ \cup \{+\infty\}$*:*

$$\exists c \in \mathbb{R}_+,\ \forall (x_1 \dots x_{2q}) \in \left(\mathbb{R}^d\right)^{2q},$$
$$\|MDA(x_1 \dots x_q) - MDA(x_{q+1} \dots x_{2q})\|_p \leq c \max_{(i,j)\in(H_1 \cup H_2)^2} \|x_i - x_j\|_p$$

Proof. We will proceed by construction of *MDA* (Section 3.1.2).

Reusing the notation from Section 3.1.2, we recall that $MDA(x_1 \dots x_q) \triangleq \frac{1}{q-f} \sum_{x\in\mathcal{S}_1} x$.

Since $\mathcal{S}_1$ is a subset of size $q-f$ of smallest diameter in $\{x_1 \dots x_q\}$, the following holds:

$$\exists (i,j) \in {H_1}^2,\ \forall (y,z) \in \mathcal{S}_1^2,\ \|y-z\| \leq \|x_i - x_j\|$$

Then, observing that $q - f > f \Rightarrow \exists k \in H_1,\ x_k \in \mathcal{S}_1$, we can compute:

$$\begin{aligned}
\|MDA(x_1 \dots x_q) - x_k\| &= \left\| \left(\frac{1}{q-f} \sum_{x\in\mathcal{S}_1} x \right) - x_k \right\| \\
&= \frac{1}{q-f} \left\| \sum_{x\in\mathcal{S}_1} (x - x_k) \right\| \\
&\leq \frac{1}{q-f} \sum_{x\in\mathcal{S}_1} \|x - x_k\| \\
&\leq \frac{1}{q-f} \sum_{x\in\mathcal{S}_1} \left(\max_{(i,j)\in{H_1}^2} \|x_i - x_j\| \right) \\
&\leq \max_{(i,j)\in{H_1}^2} \|x_i - x_j\|
\end{aligned}$$

Using the same construction, with $l \in H_2$:

$$\|MDA(x_{q+1} \dots x_{2q}) - x_l\| \leq \max_{(i,j)\in{H_2}^2} \|x_i - x_j\|$$

Finally, reusing k and l, we can compute:

$$\begin{aligned}
&\|MDA(x_1 \dots x_q) - MDA(x_{q+1} \dots x_{2q})\| \\
=\ &\|MDA(x_1 \dots x_q) - x_k + x_k - x_l + x_l - MDA(x_{q+1} \dots x_{2q})\| \\
\leq\ &\|MDA(x_1 \dots x_q) - x_k\| + \|x_k - x_l\| + \|MDA(x_{q+1} \dots x_{2q}) - x_l\| \\
\leq\ &3 \cdot \max_{(i,j)\in(H_1\cup H_2)^2} \|x_i - x_j\|
\end{aligned}$$

And we conclude using the equivalence of ℓ_p norms. □

Lemma 4. *Coordinate-wise Median contraction effect.*

Let $(d, f) \in (\mathbb{N} - \{0\})^2$*, and let* $(n, q) \in \mathbb{N}^2$ *such that* $n \geq 3f + 1$ *and* $q = n - f$*.*
We recall that $x[i]$ *designates the* i^{th} *coordinate of the vector* $x \in \mathbb{R}^d$*.*

Let $\rho > 0$*, and let* $S \sim \overline{S}$ *a random variable following a random distribution* $\overline{S}$*, with support* $R_S \triangleq \{X \mid X \subset \{1..q\}\,,\ |X| = q - f,\ P(S_1 = X_1,\ \ldots,\ S_q = X_q) \geq \rho\}$*. That is,* S *represents a set of* $q - f$ *indexes selected randomly among the* q *non-Byzantine indexes.*

We will then formally prove that $\exists m \in [0, 1[$ *such that:*

$$
\begin{aligned}
&S_1 \sim \overline{S},\ \ldots,\ S_q \sim \overline{S} \\
&\forall \left(x_1 \ldots x_q\right) \in \left(\mathbb{R}^d\right)^q, \\
&\forall \left(z_1^{(1)} \ldots z_f^{(1)}, \ldots, z_1^{(q)} \ldots y_f^{(q)}\right) \in \left(\mathbb{R}^d\right)^{q\,f}, \\
&\forall i \in [1\,..\,q]\,, \\
&y^{(i)} \triangleq \textit{Median}\left(x_{S_i[1]} \ldots x_{S_i[q-f]}, z_1^{(i)} \ldots z_f^{(i)}\right)
\end{aligned}
$$

we have:

$$
\mathbb{E}\left[\sum_{k=1}^{d} \max_{(i,j)\in[1\,..\,q]^2} \left|y^{(i)}[k] - y^{(j)}[k]\right|\right] \leq m \sum_{k=1}^{d} \max_{(i,j)\in[1\,..\,q]^2} |x_i[k] - x_j[k]| \tag{4.7}
$$

Proof. Since $q - f \geq f + 1$, there is a strict majority of non-Byzantine coordinates when computing the *Median*. So by construction, *Median* is bounded above and below by non-Byzantine coordinates, formally: $\forall i \in [1\,..\,q]\,,\ \min_{j\in[1\,..\,q]} x_j[k] \leq y^{(i)}[k] \leq \max_{j\in[1\,..\,q]} x_j[k]$.
Thus we first observe that:

$$
\begin{aligned}
&\forall k \in [1\,..\,d]\,,\ \forall \left(s_1, \ldots, s_q\right) \in {R_S}^q,\ \forall (i, j) \in [1\,..\,q]^2\,, \\
&y^{(i)}[k] \triangleq \textit{Median}\left(x_{s_i[1]}[k] \ldots x_{s_i[q-f]}[k]\,, z_1^{(i)}[k] \ldots z_f^{(i)}[k]\right), \\
&\left|y^{(i)}[k] - y^{(j)}[k]\right| \leq \max_{(a,b)\in[1\,..\,q]^2} |x_a[k] - x_b[k]|
\end{aligned} \tag{4.8}
$$

We will now consider two cases:

1. $x_1 = x_2 = \ldots = x_q$, or
2. at least two non-Byzantine parameter vectors are different.

If $x_1 = x_2 = \ldots = x_q$, then we have:

$$
\max_{(i,j)\in[1\,..\,q]^2} |x_i[k] - x_j[k]| = \max_{(i,j)\in[1\,..\,q]^2} |x[k] - x[k]| = 0
$$

And so:

$$\forall (i,j) \in [1\,..\,q]^2, \forall k \in [1\,..\,d], \left| y^{(i)}[k] - y^{(j)}[k] \right| = 0$$
$$\Rightarrow \sum_{k=1}^{d} \max_{(i,j)\in[1\,..\,q]^2} \left| y^{(i)}[k] - y^{(j)}[k] \right| = 0$$

Which concludes for Equation (4.7) for any $m \in [0,1[$.

Otherwise, if at least two non-Byzantine parameter vectors are different, then let[5]:

- $x_{min} \triangleq \min_{i\in[1\,..\,q]} x_i[k]$ and $x_{max} \triangleq \max_{i\in[1\,..\,q]} x_i[k]$
- $A \triangleq \left\{ i \in [1\,..\,q] \,\middle|\, x_i[k] \geq \frac{1}{2}(x_{max} + x_{min}) \right\}$ and $B \triangleq \{1..q\} - A$

We observe that $|A| > 0$ and $|B| > 0$, and either $|A| \geq \lceil \frac{q}{2} \rceil$ or $|B| \geq \lceil \frac{q}{2} \rceil$.

Suppose $|A| \geq \lceil \frac{q}{2} \rceil$ and let s the set composed of all the indexes of A completed by $\max(0, q - f - |A|)$ indexes from B. We observe that $s \in R_S$, so $P(S_y = s) \geq \rho$.

If $S_1 = s, \ldots, S_q = s$, since $|A| \geq \lceil \frac{q}{2} \rceil$, we have by construction of *Median*:

$$\begin{cases} x_{min} & < & \frac{1}{2}(x_{max} + x_{min}) & \leq & y^{(1)}[k] & \leq & x_{max} \\ \vdots & & \vdots & & \vdots & & \vdots \\ x_{min} & < & \frac{1}{2}(x_{max} + x_{min}) & \leq & y^{(1)}[k] & \leq & x_{max} \end{cases}$$

The probability associated to this event is by assumption $\mathrm{P}(S_1 = s, \ldots, S_q = s) \geq \rho > 0$, and so, using Equation (4.8), we can bound $\forall (i,j) \in [1\,..\,q]^2$:

$$\mathbb{E}\left| y^{(i)}[k] - y^{(j)}[k] \right| \leq (1-\rho)(x_{max} - x_{min}) + \rho\left(x_{max} - \frac{1}{2}(x_{max} + x_{min})\right)$$
$$\leq \left(\underbrace{1 - \frac{\rho}{2}}_{0<\cdot<1}\right)(x_{max} - x_{min})$$

The case $|B| \geq \lceil \frac{q}{2} \rceil$ is symmetrical and yields the exact same conclusion.

We then have $\exists m' \in [1 - \frac{\rho}{2}, 1[$ such that:

$$\mathbb{E}\left[\max_{(i,j)\in[1\,..\,q]^2} \left| y^{(i)}[k] - y^{(j)}[k] \right| \right] \leq m' \max_{(i,j)\in[1\,..\,q]^2} |x_i[k] - x_j[k]|$$

And so, using Equation (4.8) on every other coordinates $l \neq k$ and by linearity of the expected value, we can finally conclude on the existence of $m < 1$ in Equation (4.7).

□

[5] The idea to split the non-Byzantine indexes this way was originally proposed by Lê Nguyen Hoang.

Lemma 5. *Almost-sure contraction of the correct parameter vectors.*

Using some of the assumptions made in Section 4.2.2, we will prove that:

$$\lim_{t\to+\infty} \mathbb{E}\left[\max_{(a,b)\in[1\,..\,n_{ps}-f_{ps}]^2} \left\|\theta_t^{(a)} - \theta_t^{(b)}\right\|\right] = 0$$

Proof. For concision in the following development[6], we will note:

$$\operatorname*{inner\,max}_{(a,b)\in[1\,..\,q]^2} \left\|x^{(a)} - x^{(b)}\right\|_1 \triangleq \sum_{k=1}^{d} \max_{(a,b)\in[1\,..\,q]^2} \left|x^{(a)}[k] - x^{(b)}[k]\right|$$

Equation (4.3) displays what we call the *equivalence of innermax*:

$$\max_{(a,b)\in[1\,..\,q]^2} \left\|x^{(a)}-x^{(b)}\right\|_1 \leq \operatorname*{inner\,max}_{(a,b)\in[1\,..\,q]^2} \left\|x^{(a)}-x^{(b)}\right\|_1 \leq \frac{q\,(q-1)}{2} \max_{(a,b)\in[1\,..\,q]^2} \left\|x^{(a)} - x^{(b)}\right\|_1$$

Using Equation (4.3), we observe in particular that:

$$\lim_{t\to+\infty} \mathbb{E}\left[\operatorname*{inner\,max}_{(a,b)\in[1\,..\,n_{ps}-f_{ps}]^2} \left\|\theta_t^{(a)} - \theta_t^{(b)}\right\|_1\right] = 0$$

$$\Rightarrow \lim_{t\to+\infty} \mathbb{E}\left[\max_{(a,b)\in[1\,..\,n_{ps}-f_{ps}]^2} \left\|\theta_t^{(a)} - \theta_t^{(b)}\right\|\right] = \lim_{t\to+\infty} \mathbb{E}\left[\max_{(a,b)\in[1\,..\,n_{ps}-f_{ps}]^2} \left\|\theta_t^{(a)} - \theta_t^{(b)}\right\|_1\right] = 0$$

Using the assumption that correct gradient estimations' deviation is bounded (Point 2. of Section 4.2.2), since the number of correct workers is also bounded, it holds that:

$$\exists\sigma \in \mathbb{R}_+,\ \forall t \in \mathbb{N},\ \mathbb{E}\left[\operatorname*{inner\,max}_{i\in[1\,..\,n_w-f_w]} \left\|g_t^{(i)} - \mathbb{E}\, g_t^{(i)}\right\|_1\right] \leq \sigma \tag{4.9}$$

We use the same notations as in Figure 4.2. In particular for $t+1 \bmod T \neq 0$:

- $\theta_{t+1}^{(i)} = \theta_t^{(i)} - \eta_t\, \overline{G}_t^{(i)}$, where $\overline{G}_t^{(i)}$ is the output of *MDA* of $g_t^{(x)} \ldots g_t^{(y)}$.
- $\mathbb{E}\, g_t^{(i)} \triangleq \nabla Q\left(\overline{\theta}_t^{(i)}\right)$, where $\overline{\theta}_t^{(i)}$ is the output of *Median* of $\theta_t^{(x)} \ldots \theta_t^{(y)}$.

Then, for $t \geq 0$ and $t+1 \bmod T \neq 0$, we can bound:

$$\begin{aligned}
&\mathbb{E}\left[\operatorname*{inner\,max}_{(a,b)\in[1\,..\,n_{ps}-f_{ps}]^2} \left\|\theta_{t+1}^{(a)} - \theta_{t+1}^{(b)}\right\|_1\right] \\
&= \mathbb{E}\left[\operatorname*{inner\,max}_{(a,b)\in[1\,..\,n_{ps}-f_{ps}]^2} \left\|\left(\theta_t^{(a)} - \eta_t\, \overline{G}_t^{(a)}\right) - \left(\theta_t^{(b)} - \eta_t\, \overline{G}_t^{(b)}\right)\right\|_1\right]
\end{aligned}$$

[6] The idea to analyze the contraction of each coordinate separately, here formalized with the *innermax*, is the idea of Lê Nguyen Hoang, and it was an "unblocker" to the original redaction of this proof.

$$\leq \mathbb{E}\left[\underset{(a,b)\in[1\,..\,n_{ps}-f_{ps}]^2}{\text{inner max}} \left\|\theta_t^{(a)} - \theta_t^{(b)}\right\|_1\right] + \eta_t\ \mathbb{E}\left[\underset{(a,b)\in[1\,..\,n_{ps}-f_{ps}]^2}{\text{inner max}} \left\|\overline{G}_t^{(a)} - \overline{G}_t^{(b)}\right\|_1\right]$$

Using Lemma 3 and the *equivalence of innermax*, $\exists c > 0$ so that we can bound:

$$\mathbb{E}\left[\underset{(a,b)\in[1\,..\,n_{ps}-f_{ps}]^2}{\text{inner max}} \left\|\overline{G}_t^{(a)} - \overline{G}_t^{(b)}\right\|_1\right] \leq c\ \mathbb{E}\left[\underset{(a,b)\in[1\,..\,n_w-f_w]^2}{\text{inner max}} \left\|g_t^{(a)} - g_t^{(b)}\right\|_1\right]$$

The distance between two gradients $g_t^{(a)}$ and $g_t^{(b)}$ can be split into:

$$\left\|g_t^{(a)} - g_t^{(b)}\right\|_1 = \left\|\left(g_t^{(a)} - \nabla Q\left(\overline{\theta}_t^{(a)}\right)\right) - \left(g_t^{(b)} - \nabla Q\left(\overline{\theta}_t^{(b)}\right)\right) + \left(\nabla Q\left(\overline{\theta}_t^{(a)}\right) - \nabla Q\left(\overline{\theta}_t^{(b)}\right)\right)\right\|_1$$

And so, using Equation (4.9), Point 5. of Section 4.2.2 and Lemma 4:

$$\begin{aligned}\mathbb{E}\left[\underset{(a,b)\in[1\,..\,n_w-f_w]^2}{\text{inner max}} \left\|g_t^{(a)} - g_t^{(b)}\right\|_1\right] &\leq 2\,\sigma + \mathbb{E}\left[\underset{(a,b)\in[1\,..\,n_w-f_w]^2}{\text{inner max}} \left\|\nabla Q\left(\overline{\theta}_t^{(a)}\right) - \nabla Q\left(\overline{\theta}_t^{(b)}\right)\right\|_1\right] \\ &\leq 2\,\sigma + l\,m\ \mathbb{E}\left[\underset{(a,b)\in[1\,..\,n_{ps}-f_{ps}]^2}{\text{inner max}} \left\|\theta_t^{(a)} - \theta_t^{(b)}\right\|_1\right]\end{aligned}$$

Noting $u_t \triangleq \mathbb{E}\left[\underset{(a,b)\in[1\,..\,n_{ps}-f_{ps}]^2}{\text{inner max}} \left\|\theta_t^{(a)} - \theta_t^{(b)}\right\|_1\right]$ and "telescoping" the above inequalities:

$$\forall t \in \mathbb{N},\ t+1 \bmod T \neq 0,\ 0 \leq u_{t+1} \leq (1 + l\,m\,\eta_t\,c)\,u_t + 2\,\sigma\,c\,\eta_t$$

To express the DMC phase every T steps, we define a new learning rate series:

$$\hat{\eta}_t \triangleq \max_{t \leq i < t+T} \eta_i$$

We can then include the DMC phase every T steps, in the series u_t:

$$\forall t \in \mathbb{N},\ t \bmod T = 0,\ 0 \leq u_{t+T} \leq m\,(1 + l\,m\,\hat{\eta}_t\,c)^T\,u_t + 2\,\sigma\,c\,\hat{\eta}_t\,m\,T$$

For readability, we define $v_t \triangleq u_{(T\cdot t)}$, and we will prove that $\lim_{t\to+\infty} v_t = 0$.

Since $\sum_{t\in\mathbb{N}} {\eta_t}^2$ converges (Assumption 2), we have that $\lim_{t\to+\infty} \hat{\eta}_t = \lim_{t\to+\infty} \eta_t = 0$, and so:

$$\begin{aligned}\lim_{t\to+\infty} \hat{\eta}_t = 0 &\Rightarrow \forall \varepsilon > 0,\ \exists \tau \in \mathbb{N},\ \forall t \geq \tau,\ \hat{\eta}_t < \varepsilon \\ &\Rightarrow \exists k \in\ ]m;1[\,,\ \exists \tau \in \mathbb{N},\ \forall t \geq \tau,\ \hat{\eta}_t < \frac{\sqrt[T]{\frac{k}{m}} - 1}{l\,c\,m}\end{aligned} \tag{4.10}$$

Then, using k and τ from Equation (4.10), it holds:

$$\begin{aligned}v_{\tau+1} &\leq m\,(1 + l\,m\,\hat{\eta}_t\,c)^T\,v_\tau + 2\,\sigma\,c\,\hat{\eta}_\tau\,m\,T \\ &\leq k\,v_\tau + 2\,\sigma\,c\,\hat{\eta}_\tau\,m\,T\end{aligned}$$

Similarly for step $\tau + t > \tau$:

$$\begin{aligned} v_{\tau+t} &\leq k\, v_{\tau+t-1} + 2\,\sigma\, c\,\hat{\eta}_{\tau+t-1}\, m\, T \\ v_{\tau+t} &\leq k^2\, v_{\tau+t-2} + k\, 2\,\sigma\, c\,\hat{\eta}_{\tau+t-2}\, m\, T + 2\,\sigma\, c\,\hat{\eta}_{\tau+t-1}\, m\, T \\ &\leq k^t\, v_\tau + 2\,\sigma\, c\, m\, T \sum_{i=1}^{t} k^{i-1}\,\hat{\eta}_{\tau+t-i} \end{aligned}$$

Since $0 < k < 1$ and $\exists r \in \mathbb{R}_+,\ \forall (t,u) \in T\,\mathbb{N} \times [0\,..\,T-1]\,,\ u_{t+u} \leq r\, v_t$, and using Lemma 2, we finally conclude that:

$$\lim_{t\to+\infty} \mathbb{E}\left[\max_{(a,b)\in[1\,..\,n-f]^2} \left\|\theta_t^{(a)} - \theta_t^{(b)}\right\|\right] = \lim_{t\to+\infty} u_t = \lim_{t\to+\infty} v_t = \lim_{t\to+\infty} v_{\tau+t} = 0$$

□

Lemma 6. *Expected convergence of the loss gradient.*

Using some of the assumptions made in Section 4.2.2, we will prove that:

$$\lim_{t\to+\infty} \mathbb{E}\left\|\nabla Q\left(\theta_t^{(1)}\right)\right\| = 0$$

Proof. Using Lemma 5 we will show that, intuitively, the trajectory of any non-Byzantine parameter vector (we have chosen to study $\theta_t^{(1)}$) will be arbitrarily *close* to a "reference" trajectory obtained in the single-server situation with a (α, f)-Byzantine resilient GAR. Such a reference trajectory has been proven to converge (Blanchard et al., 2017).

Formally, we will prove that, after some step $t_c \in \mathbb{N}$, each non-Byzantine estimated gradient $g_t^{(i)},\ i \in [1\,..\,n_{ps} - f_{ps}]$ satisfies the conditions for (α, f)-Byzantine resilient relative to the gradient distribution at the single parameter vector $\theta_t^{(1)}$.

Using the notation of Equation (2.1), the difference between the gradient estimated at $\overline{\theta}_t^{(i)}$ and the gradient estimated at $\theta_t^{(1)}$ **using the same dataset samples** is bounded by:

$$\begin{aligned} &\left\|\text{estimate\_gradient}\left(\overline{\theta}_t^{(i)}\right) - \text{estimate\_gradient}\left(\theta_t^{(1)}\right)\right\|_1 \\ &= \left\|\frac{1}{b}\sum_{k=1}^{b} \nabla q\left(\overline{\theta}_t^{(i)}, x_t^{(k)}\right) - \frac{1}{b}\sum_{k=1}^{b} \nabla q\left(\theta_t^{(1)}, x_t^{(k)}\right)\right\|_1 \\ &= \frac{1}{b}\left\|\sum_{k=1}^{b} \left(\nabla q\left(\overline{\theta}_t^{(i)}, x_t^{(k)}\right) - \nabla q\left(\theta_t^{(1)}, x_t^{(k)}\right)\right)\right\|_1 \\ &\leq \frac{1}{b}\sum_{k=1}^{b} \left\|\nabla q\left(\overline{\theta}_t^{(i)}, x_t^{(k)}\right) - \nabla q\left(\theta_t^{(1)}, x_t^{(k)}\right)\right\|_1 \end{aligned}$$

And so, taking the expected value and using Point 7. of Section 4.2.2:

$$\begin{aligned}
&\mathbb{E}\left\|\text{estimate_gradient}\left(\overline{\theta}_t^{(i)}\right) - \text{estimate_gradient}\left(\theta_t^{(1)}\right)\right\|_1 \\
&\leq \frac{1}{b}\sum_{k=1}^{b}\mathbb{E}\left\|\nabla q\left(\overline{\theta}_t^{(i)}, x_t^{(k)}\right) - \nabla q\left(\theta_t^{(1)}, x_t^{(k)}\right)\right\|_1 \\
&\leq l'\ \mathbb{E}\left\|\overline{\theta}_t^{(i)} - \theta_t^{(1)}\right\|_1
\end{aligned}$$

By construction of *Median*, we recall that:

$$\forall k \in [1\,..\,d]\,,\ \min_{j\in[1\,..\,n_{ps}-f_{ps}]}\theta_t^{(j)}[k] \leq \overline{\theta}_t^{(i)}[k] \leq \max_{j\in[1\,..\,n_{ps}-f_{ps}]}\theta_t^{(j)}[k]$$

Using Lemma 5 and "telescoping" the above inequalities, we then have:

$$\begin{aligned}
&\lim_{t\to+\infty}\mathbb{E}\left\|\text{estimate_gradient}\left(\overline{\theta}_t^{(i)}\right) - \text{estimate_gradient}\left(\theta_t^{(1)}\right)\right\|_1 \\
&= \lim_{t\to+\infty}\mathbb{E}\left\|\overline{\theta}_t^{(i)} - \theta_t^{(1)}\right\|_1 = 0
\end{aligned}$$

Hence, $\forall\varepsilon > 0,\ \exists t_\varepsilon \in \mathbb{N}$ such that $\forall t \geq t_\varepsilon$ we can rewrite:

$$\begin{aligned}
&\forall i \in [1\,..\,n_w - f_w]\,, \\
&\exists\left(\varepsilon_t^{(i)}, e_t^{(i)}\right) \in \mathbb{R}_+ \times \mathbb{R}^d,\ \varepsilon_t^{(i)} \leq \varepsilon,\ \left\|e_t^{(i)}\right\| = 1, \\
&g_t^{(i)} = g_t^{(1)} + \varepsilon_t^{(i)}\, e_t^{(i)}
\end{aligned}$$

Using this new expression and the equivalence of norms, we can bound $\forall i \in [1\,..\,n_w - f_w]$:

$$\begin{aligned}
\mathbb{E}\left[\left\|g_t^{(i)} - \mathbb{E}\, g_t^{(i)}\right\|^2\right] &= \mathbb{E}\left[\left\|g_t^{(i)} - \nabla Q\left(\overline{\theta}_t^{(i)}\right)\right\|^2\right] \\
&= \mathbb{E}\left[\left\|g_t^{(1)} - \nabla Q\left(\overline{\theta}_t^{(1)}\right) + \varepsilon_t^{(i)}\, e_t^{(i)} + \nabla Q\left(\overline{\theta}_t^{(1)}\right) - \nabla Q\left(\overline{\theta}_t^{(i)}\right)\right\|^2\right] \\
&\leq \mathbb{E}\left[\left\|g_t^{(1)} - \nabla Q\left(\overline{\theta}_t^{(1)}\right)\right\|^2\right] + \varepsilon^2 + \mathbb{E}\left[\left\|\nabla Q\left(\overline{\theta}_t^{(1)}\right) - \nabla Q\left(\overline{\theta}_t^{(i)}\right)\right\|^2\right] \\
&\leq \mathbb{E}\left[\left\|g_t^{(1)} - \nabla Q\left(\overline{\theta}_t^{(1)}\right)\right\|^2\right] + \varepsilon^2 + d\, l^2\ \mathbb{E}\left[\left\|\overline{\theta}_t^{(i)} - \overline{\theta}_t^{(1)}\right\|^2\right]
\end{aligned}$$

Since we have by assumption $\kappa > 1$ in Point 8. of Section 4.2.2, using the above decomposition and Lemma 5 again, $\exists t_c \in \mathbb{N}$ such that $\forall t \geq t_c$ it holds that:

$$\begin{aligned}
&\frac{8{f_w}^2}{(n_w - f_w)^2}\ \mathbb{E}\left[\left\|g_t^{(i)} - \nabla Q\left(\overline{\theta}_t^{(i)}\right)\right\|^2\right] \\
&\leq \kappa^2\,\frac{8\,{f_w}^2}{(n_w - f_w)^2}\ \mathbb{E}\left[\left\|\text{estimate_gradient}\left(\theta_t^{(1)}\right) - \nabla Q\left(\theta_t^{(1)}\right)\right\|^2\right] \\
&\leq \left\|\nabla Q\left(\theta_t^{(1)}\right)\right\|^2
\end{aligned}$$

And so for $t \geq t_c$, using the assumption made in Point 4. of Section 4.2.2, each $g_t^{(i)}$ satisfies for $\theta_t^{(1)}$ the condition of (α, f)-Byzantine resilience of *MDA* (Section 3.1.2).

Finally, with $\kappa > n_{ps}$ and using the assumptions made from Point 1. to Point 6. of Section 4.2.2, we can reuse the whole proof of convergence in the single-server setting devised by Blanchard et al. (2017) for any (α, f)-Byzantine resilient GAR to conclude. □

4.3 Experimental Evaluations

We implemented the algorithms presented in Section 4.2.1 on top of TensorFlow (TensorFlow contributors, 2015). We deployed our code on a physically distributed cluster, and in this section we report on the empirical results we obtained.

4.3.1 Evaluation Settings

Testbed.

We use Grid5000[7] as an experimental platform. We employ up to 20 worker nodes and up to 6 parameter servers. Each node in the cluster has:

- 2 Intel Xeon E5-2630 v4, 10 cores per socket
- 256 GiB of RAM
- 2×10 Gbps Ethernet switched networking

Experiments.

We consider the standard, academic image classification task CIFAR-10, due to its wide adoption as a benchmark in the distributed machine learning literature (Abadi et al., 2016; Chilimbi et al., 2014). CIFAR-10 consists in small picture of various common objects (cars, planes, boats, etc). It is composed of of $60\,000$ 32×32 RGB images distributed in 10 classes. Its training set contains $50\,000$ datapoints, and the testing set contains the remaining $10\,000$ datapoints.

As model, we employ the *CifarNet* neural network architecture. It comprises $1\,756\,426$ trainable parameters. Using 32-bit floating point number representation, the size of one parameter vector/gradient is then approximately 7.0 MB. This roughly corresponds to the size of each message exchanged between parameter servers/workers.

[7] https://www.grid5000.fr/. While the code was co-authored by both Arsany Guirguis and me, Arsany Guirguis ran the experiments and produced the original graphs (that are slightly modified in this thesis).

Metrics.

We evaluate the performance of *ByzSGD* with the top-1 cross-accuracy achieved by the model (Section 2.1.1). We report the *top-1 cross-accuracy* function of either the step number t or time (in seconds). The evolution of the accuracy function of the step number reveals whether using Byzantine resilient aggregation rules impact the quality of the training. The evolution of the accuracy function of time allows to compare how much slower *ByzSGD* is compared to the non-Byzantine resilient baseline.

Baseline.

We consider vanilla TensorFlow (shorthand: vanilla TF) as the baseline. Given that such a baseline does not converge in Byzantine environments (Damaskinos et al., 2019), we use it only to estimate the overhead *ByzSGD* incurs in non-Byzantine environments.

4.3.2 Evaluation Results

We assess *ByzSGD*'s performance against the baseline in a non-Byzantine environment (the baseline is not Byzantine resilient). We vary the batch-size and the maximum numbers of Byzantine workers f_w and Byzantine parameter servers f_{ps} *ByzSGD* must tolerate. Changing f_w and f_{ps} affects the number of gradients/parameter servers awaited by each node: due to the asynchrony, no worker/server can await more than $n_w - f_w$ gradients and $n_{ps} - f_{ps}$ parameter vectors (c.f. Section 4.2.1).

In all our experiments we use $T = 333$. El-Mhamdi et al. (2020) discusses the effect of varying T; we use here one of the (many) ideal values reported. The empirical observation is that, from $T \geq 10$, further increasing T does not impact the throughput of *ByzSGD* much. This effect is easily explained by the fact T directly divides the average time spent in the DMC phase. Even with $T = 1$, the cost of the DMC phase is smaller than the gradient exchange phase, as there are more workers than parameter servers in our experiments. The speedup for varying T from T_1 to $T_2 = k\,T_1$, $k > 1$, is then roughly[8] upper bounded by $speedup < \frac{k(T_1+1)}{k\,T_1+1}$ (Amdahl's law (Amdahl, 1967)).

Figure 4.3 shows the convergence of *ByzSGD*, either function of the step number t or run time, when there is no actual attack. We observe that the evolution of the cross-accuracy per step appears identical between the baseline and *ByzSGD*. When $f_w > 0$, *ByzSGD* induces a small loss in final cross-accuracy. Such a loss is more emphasized with a smaller batch size (Figure 4.3) than a larger one (Figure 4.4). This accuracy loss is admitted in previous work (Xie et al., 2018b) and is inherited from using statistical methods (basically, *MDA* in our case) for Byzantine resilience. Regarding the gain

[8] Neglecting the CPU time and assuming the DMC phase takes as long as the gradient exchange phase, which is conservative since the DMC phase is less time-consuming than the gradient exchange phase.

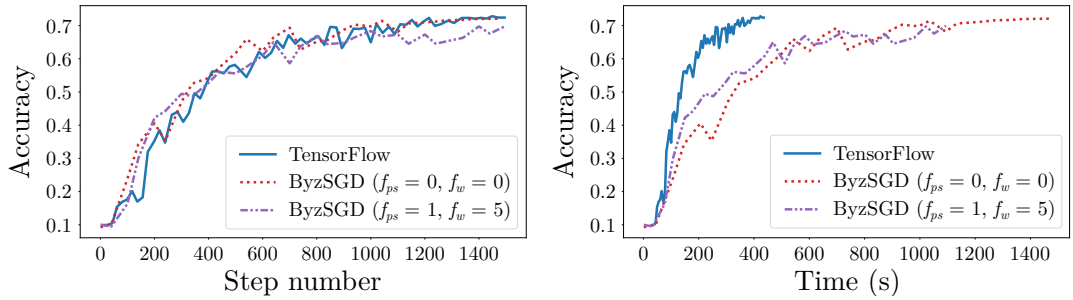


Figure 4.3: Top-1 cross-accuracy function of either the step number t or time, for *ByzSGD* vs the baseline (TF) in a non-Byzantine environment. The batch-size is 100.

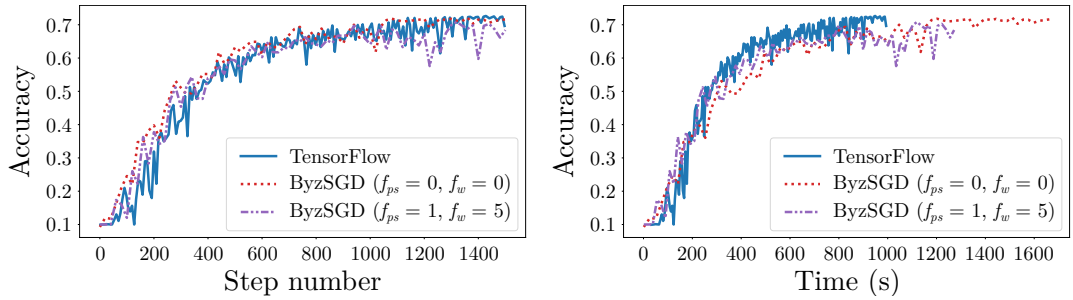


Figure 4.4: Top-1 cross-accuracy function of either the step number t or time, for *ByzSGD* vs the baseline (TF) in a non-Byzantine environment. The batch-size is 250.

of accuracy over time, the baseline reached the final cross-accuracy several times faster than *ByzSGD*. This result can largely be attributed to the cost of the additional processing and communication rounds carried out by our algorithm.

Figure 4.4 shows the same settings as Figure 4.3, except the batch-size had been set from $b = 100$ to $b = 250$, multiplying the gradient computation time by $\times 2.5$. As noted above, higher batch-sizes lead to higher top-1 cross-accuracies with statistically-robust GARs. The longer time it takes to compute gradients naturally impacts the speedup of *ByzSGD* over the baseline: the fraction of time spent computing gradients become more prevalent with $b = 250$ than with $b = 100$, mechanically increasing the speedup.

Both sets of experiments in figures 4.3 and 4.4 exhibit the same behavior regarding the variation of f_w and f_{ps}. Even if higher f_w and f_{ps} respectively decrease the number of gradients and parameter vectors awaited by each (non-Byzantine) node, the impact on the performance is barely visible. Two opposing effects are in play here. Higher f_w and f_{ps} both lead to less non-Byzantine gradients aggregated per step (since there is no actual attack). This filtering of valid gradients/parameter vectors increases the variance of the aggregate, in turn slightly increasing the time it takes to reach the final top-1 cross-accuracy. Opposing to this effect is the (slight) throughput gain *ByzSGD* benefits when less gradients and parameter vectors are awaited per step.

4.4 Concluding Remarks

In this chapter, we tackled the problem of *general* Byzantine SGD, where no single node needs to be trusted. Having no single-point-of-failure, the general Byzantine SGD model enables further more resilient distributed deployments. We propose *ByzSGD*, historically the first algorithm to solve the general Byzantine SGD problem. We formally prove the soundness of our algorithm, and we assess its costs against a non-resilient baseline (a distributed deployment of the same model and dataset with TensorFlow).

One of the important features of *ByzSGD* is that it is able to work in asynchrony: as long as messages can be exchanged between nodes, the algorithm will make progress. This is a performance feature synchronous and partially synchronous algorithms lack in the presence of Byzantine failures. This is arguably an important feature for correctness as well. Breaking the timing assumptions of a synchronous algorithm also breaks its guarantees. In practice, unexpectedly long delays can be triggered by e.g. a software update on a non-Byzantine node, temporary congestion on the network, or even by a distributed denial-of-service attack for powerful-enough adversaries.

The theoretical analysis (Section 4.2.3) shows that two opposing forces are at play. The variance of the non-Byzantine gradient estimations tends to push the non-Byzantine parameter vectors apart from each other. The DMC procedure works to pull these vectors back together. Intuitively, as Lemma 5 and Lemma 6 formalize, these forces induce three different training phases. The first phase corresponds to the *expansion of the envelope* of the non-Byzantine parameter vectors. This expansion peaks at step $\arg\max_{t\in\mathbb{N}} u_t$. Then comes the *contraction* phase. And finally for $t \geq t_c$, the *convergence* phase of the non-Byzantine parameter vectors. One important critic is that, unless $t_c = 0$[9], the adversary may be free to move the non-Byzantine parameter vectors (almost) wherever it likes during the first two phases, starting the convergence phase from a potentially sub-optimal (El-Mhamdi et al., 2018) region of the loss function.

Another set of critics[10] regards some theoretical assumptions, that may be (or are) difficult to satisfy in practice, e.g.: requiring i.i.d. gradients, or the Lipschitz continuity of the loss. In particular, two assumptions (Point 7. and Point 9. of Section 4.2.2) are not inherited from the literature. Point 7. is even stronger than mere Lipschitz continuity. Point 9. brings a subtle requirement, strictly stronger than the standard definition of asynchrony (Cachin et al., 2011), but also strictly more lenient than partial synchrony. One particularly demanding assumption in practice is the upper-bound on the *variance-norm* ratio (Point 8). This assumption is instrumental to converge despite Byzantine workers, yet often unsatisfied in practice (El-Mhamdi et al., 2020), enabling successful attacks (Baruch et al., 2019; Xie et al., 2019a). In the next chapter, we propose an inexpensive, easy-to-implement mitigation to this ubiquitous problem.

[9] This is theoretically possible, if η_t remains small enough or the loss has a very low Lipschitz constant.
[10] Please note that some of the listed critics here have been addressed in (El-Mhamdi et al., 2020a).

5 Distributed Momentum

Byzantine resilient Stochastic Gradient Descent (SGD) aims at shielding model training from *Byzantine faults*, be they ill-labeled training datapoints, exploited software/hardware vulnerabilities, or malicious worker nodes in a distributed setting. Two recent attacks have been challenging state-of-the-art statistically-robust defenses though, often successfully precluding the model from even fitting the training set.

The main identified weakness in this family of GARs is their requirement for a sufficiently low *variance-norm* ratio. Intuitively, the *variance-norm* ratio measure how informative about the real gradient $\nabla Q(\theta_t)$ the non-Byzantine stochastic gradient is. And if the stochastic gradients are too noisy, statistically-robust GARs are not able to correctly infer the half-space in which the real gradient is, opening the door for attacks.

In this chapter, we propose a practical method which, despite increasing the variance, reduces the *variance-norm* ratio, mitigating the identified weakness. We assess the effectiveness of our method over 736 different training configurations, comprising the 2 state-of-the-art attacks and 6 defenses. For confidence and reproducibility purposes, each configuration is run 5 times with specified seeds (1 to 5), totalling 3680 runs.

The work presented in this chapter follows the distributed setting presented in Section 2.1.2, simply tweaking the presented training algorithms (Section 5.3.1). The attack model is also the one presented in Section 2.2.

5.1 Motivation

In Byzantine SGD, three families of defense techniques can be distinguished.

The first employs *redundancy* schemes (Chen et al., 2018), inspired by coding theory. This approach has strong resilience guarantees, but its requirements to share data and synchronize between non-Byzantine workers which datapoints are sampled make

this approach unsuitable for several classes of applications, e.g. when data cannot be shared for privacy, scalability or legal reasons.

The second family, said *suspicion-based*, relies on estimating at the server how much each gradient decreases the loss (Xie et al., 2019b; Xie, 2019). There is a trade-off between aggregation time and precision of the scoring (using larger batch-size).

The third family uses *statistically-robust* aggregation schemes, and is the focus of this paper. The underlying idea is simple. At each training step, the server aggregates the stochastic gradients computed by the workers into one gradient, using a function called a Byzantine resilient GAR (Section 2.2). These statistically-robust GARs are designed to produce at each step a gradient that is expected to decrease the loss.

Intuitively, one can think of this third family as different formulations of the multivariate median. In particular, if the non-Byzantine gradients were all equal at each step, any different (adversarial) gradient would be rejected by each of these medians, and no attack would succeed. But due to their stochastic nature, the non-Byzantine gradients are different: their *variance* is strictly positive. Formal guarantees on any given statistically-robust GAR typically require that the *variance-norm* ratio, the ratio between the *variance* of the non-Byzantine gradients and the *norm* of the *expected* non-Byzantine gradient, remains below a certain constant (constant which depends on the GAR itself and fixed hyperparameters). Intuitively, this notion of *variance-norm* ratio can be comprehended quite analogously to the inverse of the *signal-to-noise* ratio (i.e. the *"noise-to-signal"* ratio) in signal processing.

However, Baruch et al. (2019) noted that an attack could send gradients that are close to non-Byzantine *outlier* gradients, building an apparent majority of gradients that could be sufficiently far from the real gradient to *increase* the loss. This can happen against most statistically-robust GARs, as the *variance-norm* ratio is often *too* large in practice. Two recent attacks (Baruch et al., 2019; Xie et al., 2019a) were able to exploit this fact to substantially hamper the training process (which our experiments confirm).

The work presented in this chapter aims at (substantially) improving the resilience of statistically-robust GARs "also in practice", by reducing the *variance-norm* ratio of the gradients received by the server. We do that by taking advantage of an old technique normally used for acceleration: momentum. This technique is regularly applied at the server, but instead we propose to confer it upon each distributed worker, effectively making the Byzantine resilient GAR aggregate *accumulated* gradients. Crucially, there is no computational complexity attached to our reformulation: it only *reorders* operations in existing (distributed) algorithms.

5.2 Studied Algorithms

5.2.1 Byzantine resilient GARs

We briefly present below the 6 studied *Gradient Aggregation Rules* (GARs). These GARs all are (α, f)-Byzantine resilient (Definition 1). Within its operating assumptions, a Byzantine resilient GAR guarantees *convergence* even in an adversarial setting.

We reuse the notations from Section 1.2.2, Table 1.1. Since there is only one trusted parameter server ($n_{ps} = 1$ and $f_{ps} = 0$), for concision in this chapter, let $n \triangleq n_w$ be the number of gradients the parameter server received from the n workers, and let $f \triangleq f_w$ be the maximum number of Byzantine gradients the GAR must tolerate.

Definition 2 (*Variance-norm* ratio)**.**

Let X a random variable following some distribution $\mathcal{X}$, for which both the first moment and the second centered moment exist.

The variance-norm *ratio is then defined as:* $\frac{\sqrt{\mathbb{E}\left[\|X-\mathbb{E}\,X\|^2\right]}}{\|\mathbb{E}\,X\|}$.

The *variance-norm* ratios respectively required by the 6 studied GARs are laid below. We will note $\mathbb{E}\,\|\mathcal{G}_t - \mathbb{E}\,\mathcal{G}_t\|^2$ the variance of the honest gradient (at step t), and note $\|\mathbb{E}\,\mathcal{G}_t\|$ the norm of the expected, honest gradient (at step t).

Krum. (Blanchard et al., 2017)

Each received gradient is assigned a score. The score of gradient x is the sum of the squared ℓ_2-distances between x and the $n-f-2$ closest gradients to x. The aggregated gradient is then the arithmetic mean of the $n-f-2$ gradients with the smallest scores. This variant is called *Multi-Krum* in the original paper.

To be (α, f)-Byzantine resilient, *Krum* requires the variance of the honest gradient $\mathbb{E}\,\|\mathcal{G}_t - \mathbb{E}\,\mathcal{G}_t\|^2$ to be bounded below the norm of the honest gradient $\|\mathbb{E}\,\mathcal{G}_t\|$ as follows:

$$2 \cdot \left(n-f+\frac{f\,(n-f-2)+f^2\,(n-f-1)}{n-2f-2}\right) \cdot \mathbb{E}\,\|\mathcal{G}_t - \mathbb{E}\,\mathcal{G}_t\|^2 < \|\mathbb{E}\,\mathcal{G}_t\|^2 \tag{5.1}$$

Median. (Yin et al., 2018)

The coordinate-wise median of the n received gradients. *Median* is proven (α, f)-Byzantine resilience with the following condition on the *variance-norm* ratio:

$$(n-f) \cdot \mathbb{E}\,\|\mathcal{G}_t - \mathbb{E}\,\mathcal{G}_t\|^2 < \|\mathbb{E}\,\mathcal{G}_t\|^2 \tag{5.2}$$

Trimmed Mean. (Yin et al., 2018)

The coordinate-wise trimmed-mean of the n received gradients. The *trimmed-mean* of n values is the arithmetic mean, after the f smallest and the f largest values have been discarded, of the remaining values. From Theorem 1 of Xie et al. (2018c), we can derive the following condition on the *variance-norm* ratio:

$$\frac{2\,(f+1)\,(n-f)}{(n-2f)^2}\,\mathbb{E}\,\|\mathcal{G}_t - \mathbb{E}\,\mathcal{G}_t\|^2 < \|\mathbb{E}\,\mathcal{G}_t\|^2 \tag{5.3}$$

Phocas. (Xie et al., 2018c)

The coordinate-wise arithmetic mean of the $n-f$ closest values to the coordinate-wise trimmed-mean. From Theorem 2 of Xie et al. (2018c):

$$\left(4 + \frac{12\,(f+1)\,(n-f)}{(n-2f)^2}\right)\mathbb{E}\,\|\mathcal{G}_t - \mathbb{E}\,\mathcal{G}_t\|^2 < \|\mathbb{E}\,\mathcal{G}_t\|^2 \tag{5.4}$$

MeaMed. (Xie et al., 2018a)

Same as *Phocas*, but with median replacing trimmed-mean. Theorem 5 of Xie et al. (2018a) provides the following condition:

$$10\,(n-f)\cdot\mathbb{E}\,\|\mathcal{G}_t - \mathbb{E}\,\mathcal{G}_t\|^2 < \|\mathbb{E}\,\mathcal{G}_t\|^2 \tag{5.5}$$

Bulyan. (El-Mhamdi et al., 2018)

This is a *composite* GAR, iterating on another GAR in a first selection phase. In the remaining of this paper, *Bulyan* will use *Krum*, so the first phase selects $n-2\,f-2$ gradients, at each iteration removing the highest scoring gradient. The aggregated gradient is the coordinate-wise arithmetic mean of the $n-4\,f-2$ closest values to the (coordinate-wise) median of the selected gradients.

The theoretical requirement on the *variance-norm* ratio are the same as the ones of the underlying GAR. That is, in this paper, they are the same as *Krum* (Equation (5.1)).

5.2.2 State-of-the-art Attacks

The two state-of-the-art attacks follow the same core principle. Interestingly, the attack presented in Section 3.2.2 also follows the same construction, except that the *attack vector* (see below) is a constant, unadapted to the non-Byzantine gradient submissions.

Let $\varepsilon \in \mathbb{R}_{\geq 0}$ be a non-negative factor, and $a_t \in \mathbb{R}^d$ an *attack vector* which value depends on the actual attack used (see below for possible values of a_t). At each step t, each of the f Byzantine workers submits the same Byzantine gradient: $\overline{g_t} + \varepsilon\, a_t$, where $\overline{g_t}$ is an approximation of the real gradient $\nabla Q(\theta_t)$ at step t. The value of ε is fixed (see below).

A Little is Enough. (Baruch et al., 2019)

In this attack, each Byzantine worker submits $\overline{g_t} + \varepsilon\, a_t$, with $a_t \triangleq -\sigma_t$ the opposite of the coordinate-wise standard deviation of the honest gradient distribution $\mathcal{G}_t$. Our experiments use $\varepsilon = 1.5$, as proposed by the original paper.

Fall of Empires. (Xie et al., 2019a)

Each Byzantine worker submits $(1-\varepsilon)\,\overline{g_t}$, i.e., $a_t \triangleq -\overline{g_t}$. The original paper tested $\epsilon \in \{-10, -1, 0, 0.1, 0.2, 0.5, 1, 10, 100\}$, our experiments use[1] $\varepsilon = 1.1$, corresponding in the notation of the original paper to $\epsilon \triangleq -(1-\varepsilon) = -(1-1.1) = 0.1$.

5.3 Momentum at the Workers

Intuitively, the Byzantine resilient GARs (Section 5.2.1) rely on the honest gradients being sufficiently *clumped*: the *variance-norm* ratio of the honest gradient should be sufficiently small. *Sufficiently small* is formalized for each of the studied GARs, in Equation (5.1) to Equation (5.5). For the purpose of filtering attacks, the practitioner wants the *variance-norm* ratio (of the honest gradients) to be as small as possible. In the edge case of a null *variance-norm* ratio, the honest gradients are almost-surely equal. Any GARs approximating a multidimensional median(oid), e.g. all the GARs presented in Section 5.2.1, would then always select the (identical) honest gradient, no matter the attack. But when the honest gradients are sufficiently "spread", namely when their *variance-norm* ratio is large enough, the attack vectors can intuitively "form an *apparent* majority" by relying on a few outlier (but honest) gradients (Baruch et al., 2019), and potentially substantially influence the aggregated gradient.

In this section, we present a simple technique that aims at reducing the *variance-norm* ratio: either increasing the norm without increasing (much) the variance, or decreasing the variance without decreasing (much) the norm of the honest gradients. The idea is to use *momentum*, which makes the parameters θ_t travel down the loss function with *inertia*, accumulating both the *real* gradient $\nabla Q(\theta_t)$ and the *error* (i.e. here, the *stochastic noise*) $g_t - \nabla Q(\theta_t)$. Intuitively, the accumulation of *errors* grows at a moderate rate, as past errors can be partially compensated by future ones. But when consecutive $\nabla Q(\theta_t)$ have sufficiently low solid angles, past real gradients do not

[1] This factor made this attack consistently successful in the original paper.

compensate future real gradients: the *norm* of G_t can grow "faster" (for each new step) than its *variance*, mitigating the potential impact of an attack.

5.3.1 Formulation

From the formulation of *momentum SGD* in a distributed setting (Equation (2.3)):

$$G_t \triangleq \sum_{u=0}^{t} \mu^{t-u} F\left(g_u^{(1)}, \ldots, g_u^{(n)}\right)$$

we instead confer the momentum computation on the workers:

$$G_t \triangleq F\Bigg(\underbrace{\sum_{u=0}^{t} \mu^{t-u} g_u^{(1)}}_{G_t^{(1)}}, \ldots, \underbrace{\sum_{u=0}^{t} \mu^{t-u} g_u^{(n)}}_{G_t^{(n)}}\Bigg) \tag{5.6}$$

Notations.

In the remaining of this paper, we call the original formulation:

(momentum) at the server,

and the proposed, revised formulation:

(momentum) at the worker(s).

The *variance-norm* ratio may[2] exist and can be estimated for any random variable. Regarding Byzantine resilience, only the *variance-norm* ratio of the gradients that are aggregated by the GAR is relevant. So in the experiments, when *momentum at the server* is employed, the *variance-norm* ratio of the honest gradient $G_t^{(i)} \triangleq g_t^{(i)}$, for any i the index of an honest worker. When *momentum at the workers* is used instead, $G_t^{(i)} \neq g_t^{(i)}$ (c.f. Equation (5.6)), and the *variance-norm* ratio will likely be different from the *variance-norm* ratio of $g_t^{(i)}$. Our goal is now to discover whether/when *momentum at the workers* induces a lower *variance-norm* ratio than with *momentum at the servers*.

5.3.2 Formal analysis

Without loss of generality, the indexes of the honest workers will be in $[1\,..\,n-f]$. We compare the *variance-norm* ratio of the honest, sampled gradients $g_t^{(1)} \ldots g_t^{(n-f)}$ against the *variance-norm* ratio of the honest, submitted gradients $G_t^{(1)} \ldots G_t^{(n-f)}$ when *classical* momentum is computed *at the workers*. We want the *variance-norm* ratio of the submitted gradients to be lower than the one of the sampled gradients.

[2] As long as the first two (centered) moments of a random variable exist, the *variance-norm* ratio exists.

We denote by $\mathbb{E}\,\mathcal{G}_t \triangleq \nabla Q(\theta_t)$ the "real" gradient at step t. Please note that the honest gradients are unbiased, and thus it holds that: $\forall i \in [1\,..\,n-f]\,,\ \mathbb{E}\,g_t^{(i)} = \mathbb{E}\,\mathcal{G}_t$.

Let $\lambda_t \triangleq \|\mathbb{E}\,\mathcal{G}_t\| > 0$ be the real gradient's norm at step t.
Let $\sigma_t \triangleq \sqrt{\mathbb{E}\,\|\mathcal{G}_t - \mathbb{E}\,\mathcal{G}_t\|^2}$ be the standard deviation of the real gradient at step t. The *variance-norm* ratio of the non-Byzantine subset of the sampled gradients at step t is:

$$r_t^{(s)} \triangleq \frac{{\sigma_t}^2}{{\lambda_t}^2}$$

We will now compute the *variance-norm* ratio of the non-Byzantine subset of the submitted gradients. Let $G_t^{(i)}$, with $G_{-1}^{(i)} \triangleq 0$, be the gradient sent by any honest worker i at step t, i.e.:

$$G_t^{(i)} \triangleq \sum_{u=0}^{t} \mu^{t-u} g_u^{(i)}$$

The numerator of the *variance-norm* ratio is, for any two honest workers $i \neq j$:

$$\begin{aligned}
&\mathbb{E}\left\|G_t^{(i)} - G_t^{(j)}\right\|^2 \\
&= \mathbb{E}\left\|g_t^{(i)} + \mu\, G_{t-1}^{(i)} - g_t^{(j)} - \mu\, G_{t-1}^{(j)}\right\|^2 \\
&= \mathbb{E}\left\|g_t^{(i)} - g_t^{(j)}\right\|^2 + \mu^2\ \mathbb{E}\left\|G_{t-1}^{(i)} - G_{t-1}^{(j)}\right\|^2 + \underbrace{2\,\mu\left(\underbrace{\mathbb{E}\,g_t^{(i)} - \mathbb{E}\,g_t^{(j)}}_{=\mathbb{E}\,\mathcal{G}_t - \mathbb{E}\,\mathcal{G}_t}\right)\cdot\left(\mathbb{E}\,G_{t-1}^{(i)} - \mathbb{E}\,G_{t-1}^{(j)}\right)}_{=0} \\
&= \mathbb{E}\left\|g_t^{(i)} - g_t^{(j)}\right\|^2 + \mu^2\ \mathbb{E}\left\|G_{t-1}^{(i)} - G_{t-1}^{(j)}\right\|^2 \\
&= 2\,{\sigma_t}^2 + \mu^2\left(2\,{\sigma_{t-1}}^2 + \mu^2\left(2\,{\sigma_{t-2}}^2 + \mu^2\,(...)\right)\right) \\
&= 2\sum_{u=0}^{t} \mu^{2(t-u)} {\sigma_u}^2 \qquad (5.7) \\
&= 2\ \mathbb{E}\left\|G_t^{(i)} - \mathbb{E}\,G_t^{(i)}\right\|^2
\end{aligned}$$

And the denominator of the *variance-norm* ratio is:

$$\begin{aligned}
&\left\|\mathbb{E}\,G_t^{(i)}\right\|^2 = \left\|\mathbb{E}\,g_t^{(i)} + \mu\ \mathbb{E}\,G_{t-1}^{(i)}\right\|^2 \\
&= \left\|\mathbb{E}\,g_t^{(i)}\right\|^2 + 2\,\mu\ \mathbb{E}\,g_t^{(i)}\cdot\mathbb{E}\,G_{t-1}^{(i)} + \mu^2\ \left\|\mathbb{E}\,G_{t-1}^{(i)}\right\|^2 \\
&= {\lambda_t}^2 + 2\,\mu\ \mathbb{E}\,g_t^{(i)}\cdot\left(\mathbb{E}\,g_{t-1}^{(i)} + \mu\ \left(\mathbb{E}\,g_{t-2}^{(i)} + \mu\ (...)\right)\right) \\
&\quad + \mu^2\ \left({\lambda_{t-1}}^2 + 2\,\mu\ \mathbb{E}\,g_{t-1}^{(i)}\cdot\left(\mathbb{E}\,g_{t-2}^{(i)} + \mu\ (...)\right)\right. \\
&\qquad \left. + \mu^2\ \mathbb{E}\left\|G_{t-2}^{(i)}\right\|^2\right)
\end{aligned}$$

$$= \sum_{u=0}^{t} \mu^{2(t-u)} \left({\lambda_u}^2 + 2 \sum_{v=0}^{u-1} \mu^{u-v} \underbrace{\mathbb{E}\, g_u^{(i)} \cdot \mathbb{E}\, g_v^{(i)}}_{=\mathbb{E}\,\mathcal{G}_u \cdot \mathbb{E}\,\mathcal{G}_v} \right)$$

Thus, assuming honest gradients $\mathbb{E}\, G_t^{(i)}$ do not become null:

$$r_t^{(w)} \triangleq \frac{{\Omega_t}^2}{{\Lambda_t}^2} = \frac{\sum_{u=0}^{t} \mu^{2(t-u)} {\sigma_u}^2}{\sum_{u=0}^{t} \mu^{2(t-u)} \left({\lambda_u}^2 + s_u \right)}$$

where the expected "straightness" of the gradient trajectory at step u is defined by:

$$s_u \triangleq 2 \sum_{v=0}^{u-1} \mu^{u-v}\, \mathbb{E}\,\mathcal{G}_u \cdot \mathbb{E}\,\mathcal{G}_v$$

s_u quantifies what is intuitively the *curvature* of the gradient trajectory. Straight trajectories can make s_u grow up to $(1-\mu)^{-1} > 1$ times the expected squared-norm of the honest gradients, while highly "curved" trajectories (e.g. close to a local minimum) can make s_u negative.

This observation stresses that this formulation of momentum can sometimes be harmful for the purpose of Byzantine resilience. We measured s_u for every step $u > 0$ in our experiments, and we always observed that this quantity is positive and increases for a short window of (dozen) steps (depending on η_t), and then oscillates between positive and negative values. While the empirical impact is concrete, in the form of a decreased or even canceled loss in accuracy, we also believe there is room for further improvements, as discussed in Section 8.3.

The purpose of using momentum at the workers is to reduce the *variance-norm* ratio $r_t^{(w)}$, compared to $r_t^{(s)}$. Since $g_0^{(i)} = G_0^{(i)}$, we verify that $r_0^{(u)} = r_0^{(w)}$. Then $\forall t > 0$, assuming $\Omega_{t-1} > 0$ and $\sigma_t > 0$, we have:

$$\begin{aligned} r_t^{(w)} \leq r_t^{(s)} &\Leftrightarrow \frac{{\sigma_t}^2 + \mu^2\, {\Omega_{t-1}}^2}{{\lambda_t}^2 + s_t + \mu^2\, {\Lambda_{t-1}}^2} \leq \frac{{\sigma_t}^2}{{\lambda_t}^2} \\ &\Leftrightarrow \mu^2\, {\Omega_{t-1}}^2\, {\lambda_t}^2 \leq \left(s_t + \mu^2\, {\Lambda_{t-1}}^2 \right) {\sigma_t}^2 \\ &\Leftrightarrow s_t \geq \mu^2\, {\Lambda_{t-1}}^2 \left(\frac{r_{t-1}^{(w)}}{r_t^{(s)}} - 1 \right) \end{aligned} \tag{5.8}$$

The condition for decreasing $r_t^{(w)}$ can be obtained similarly, assuming $\Omega_{t-1} > 0$ and $\sigma_t > 0$:

$$r_t^{(w)} \leq r_{t-1}^{(w)} \Leftrightarrow s_t \geq {\lambda_t}^2 \left(\frac{r_t^{(s)}}{r_{t-1}^{(w)}} - 1 \right)$$

To study the impact of a lower learning rate η_t on s_t, we will assume that the real gradient ∇Q is l-Lipschitz. Namely:

$$\forall\,(t,u)\in\mathbb{N}^2, u<t, \|\mathbb{E}\,\mathcal{G}_t-\mathbb{E}\,\mathcal{G}_u\|^2\leq l^2\,\|\theta_t-\theta_u\|^2\leq l^2\left\|\sum_{v=u}^{t-1}\eta_v\,G_v\right\|^2$$

Then, $\forall\,(t,u)\in\mathbb{N}^2, u<t$, we can rewrite:

$$\|\mathbb{E}\,\mathcal{G}_t-\mathbb{E}\,\mathcal{G}_u\|^2=\underbrace{\|\mathbb{E}\,\mathcal{G}_t\|^2}_{{\lambda_t}^2}+\underbrace{\|\mathbb{E}\,\mathcal{G}_u\|^2}_{{\lambda_u}^2}-2\,\mathbb{E}\,\mathcal{G}_t\cdot\mathbb{E}\,\mathcal{G}_u$$

And finally, we can lower-bound s_t as:

$$\begin{aligned}
&\sum_{u=0}^{t-1}\mu^{t-u}\,\|\mathbb{E}\,\mathcal{G}_t-\mathbb{E}\,\mathcal{G}_u\|^2\\
&=\sum_{u=0}^{t-1}\mu^{t-u}\left({\lambda_t}^2+{\lambda_u}^2\right)-\underbrace{2\sum_{u=0}^{t-1}\mu^{t-u}\,\mathbb{E}\,\mathcal{G}_t\cdot\mathbb{E}\,\mathcal{G}_u}_{s_t}\\
&\leq\sum_{u=0}^{t-1}\mu^{t-u}\,l^2\left\|\sum_{v=u}^{t-1}\eta_v\,G_v\right\|^2\\
\Leftrightarrow\ s_t&\geq\sum_{u=0}^{t-1}\mu^{t-u}\left({\lambda_t}^2+{\lambda_u}^2-l^2\left\|\sum_{v=u}^{t-1}\eta_v\,G_v\right\|^2\right)\\
&\geq\frac{1-\mu^t}{1-\mu}{\lambda_t}^2+\sum_{u=0}^{t-1}\mu^{t-u}\left({\lambda_u}^2-l^2\left\|\sum_{v=u}^{t-1}\eta_v\,G_v\right\|^2\right)
\end{aligned}\tag{5.9}$$

When the real gradient ∇Q is (locally) Lipschitz continuous, reducing the learning rate η_t can suffice to ensure s_t satisfies the conditions laid above for decreasing the *variance-norm* ratio $r_t^{(w)}$ (Figure 5.4 indeed shows a pronounced "dip" when the learning rate is decreased); the purpose of momentum at the workers. Importantly this last lower bound, Equation (5.9), sets how the practitioner should choose two hyperparameters, μ and η_t, for the purpose of Byzantine resilience. Basically, as long as it does not harm the training without adversary, μ should be set *as high* and η_t *as low* as possible.

5.4 Experiments

Our experiments cover 2 models, 4 datasets, the 6 studied defenses under each of the 2 state-of-the-art attacks[3], different fractions of Byzantine workers (either *half* or *a quarter*), using Nestorov instead of classical momentum, plus unattacked settings

[3]To the best of our knowledge, putting aside simple attacks (e.g. sending attack gradients sampled from a Gaussian distribution) tested in each defense papers, no other attack has been published.

where each worker is honest and the GAR is mere *averaging*. Since our theoretical results (Section 5.3.2) suggest that smaller learning rates may reduce the *variance-norm* ratio, two learning rate schedules (an *optimal* and a *smaller* one) are also tested. For reproducibility and confidence in the empirical benefits of our reformulation, we test every combination of the hyperparameters mentioned above, and each combination is repeated 5 times with specified *seeds* (1 to 5, totally 3680 runs).

The tools we developed to implement our reformulation captures $\sim$20 metrics, including the evolution of the average loss, top-1 cross-accuracy and *variance-norm* ratio of the submitted gradients. In this section, we specifically report on these 3 metrics.

5.4.1 Experimental Setup

We use a compact notation to define the models: L(#outputs) for a fully-connected linear layer, R for ReLU activation, S for log-softmax, C(#channels) for a fully-connected 2D-convolutional layer (kernel size 3, padding 1, stride 1), M for 2D-maxpool (kernel size 2), B for batch-normalization, and D for dropout (with fixed probability 0.25).

We use the models from respectively Baruch et al. (2019) and Xie et al. (2019a):

	Fully connected	***Convolutional***
Model	(784)-L(100)-R-L(10)-R-S	(3, 32$\times$32)-C(64)-R-B-C(64)-R-B-M-D- -C(128)-R-B-C(128)-R-B-M-D- -L(128)-R-D-L(10)-S
Datasets	MNIST, Fashion MNIST (83 samples/gradient)	CIFAR-10, CIFAR-100 (50 samples/gradient)
#workers	$n = 51 \quad f \in \{24, 12\}$	$n = 25 \quad f \in \{11, 5\}$

For model training, we use the *negative log likelihood* loss and respectively 10^{-4} and 10^{-2} ℓ_2-regularization for the *fully connected* and *convolutional* models. We also clip gradients, ensuring their norms remain respectively below 2 and 5 for the *fully connected* and *convolutional* models. For performance evaluation, we measure both the *top-1 cross-accuracy* over the whole test set, and the average loss at each step.

Datasets are pre-processed before training. MNIST receives the same pre-processing as in Baruch et al. (2019): an input image normalization with mean 0.1307 and standard deviation 0.3081. Fashion MNIST, CIFAR-10 and CIFAR-100 are all expanded with horizontally flipped images. For both CIFAR-10 and CIFAR-100, a per-channel normalization with means 0.4914, 0.4822, 0.4465 and standard deviations 0.2023, 0.1994, 0.2010 (Liu, 2019) has been applied.

We denote by f the number of Byzantine workers either to the maximum for which

Krum can be used (roughly an half: $f = \left\lfloor \frac{n-3}{2} \right\rfloor$), or the maximum for *Bulyan* (roughly a quarter, $f = \left\lfloor \frac{n-3}{4} \right\rfloor$). The attack factors ε_t (Section 5.2.2) are set to constants proposed in the literature, namely $\varepsilon_t = 1.5$ for Baruch et al. (2019) and $\varepsilon_t = 1.1$ for Xie et al. (2019a). We also experiment two different learning rates. The first and largest is selected so as to maximize the *performance* (highest final cross-accuracy and accuracy gain per step) of the model trained without Byzantine workers. The second and smallest is chosen so as to minimize the *performance loss* under attack, without substantially impacting the final accuracy when trained without Byzantine workers. The *fully connected* and *convolutional* models are trained respectively with $\mu = 0.9$ and $\mu = 0.99$. These values were obtained by trial and error, to maximize the accuracy gain per step when there is no attack.

Reproducibility. Particular care has been taken to make our results reproducible. Each of the 5 runs per experiment are respectively seeded with seed 1 to 5. For instance, this implies that two experiments with same seed and same model also starts with the same parameters θ_0. To further reduce the sources of non-determinism, the CuDNN backend is configured in deterministic mode (our experiments ran on two *GeForce GTX 1080 Ti*) with benchmark mode turned off. We also used *log-softmax + nll loss*, which is equal to *softmax + cross-entropy loss*, but with improved numerical stability on PyTorch. We provide our code along with a script reproducing *all* of our results, both the experiments and the graphs, in one command. Details, including software and hardware dependencies, are available in Section A.1.1.

5.4.2 Experimental Results

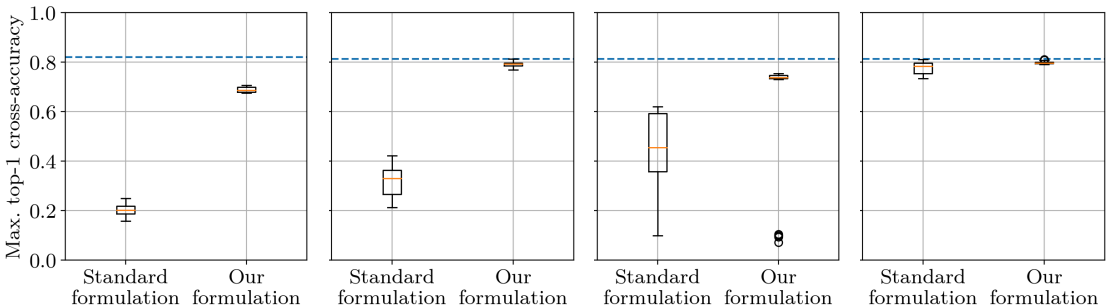


Figure 5.1: Highest measured top-1 cross-accuracy while training under attack. [a, b]: the *convolutional* model for CIFAR-10 under the attack from Baruch et al. (2019), and [c, d]: the *fully connected* model for Fashion-MNIST under the attack from Xie et al. (2019a). Roughly half the workers implements the attack in [a, c], and a quarter does in [b, d]. The dotted blue line is the median of the maximum top-1 cross-accuracy of the 5 runs without attack, and the boxes aggregate the maximum top-1 cross-accuracy obtained under attack with each 5 runs of the 6 studied defenses.

This section reports on the evolution of the average loss, top-1 cross-accuracy and

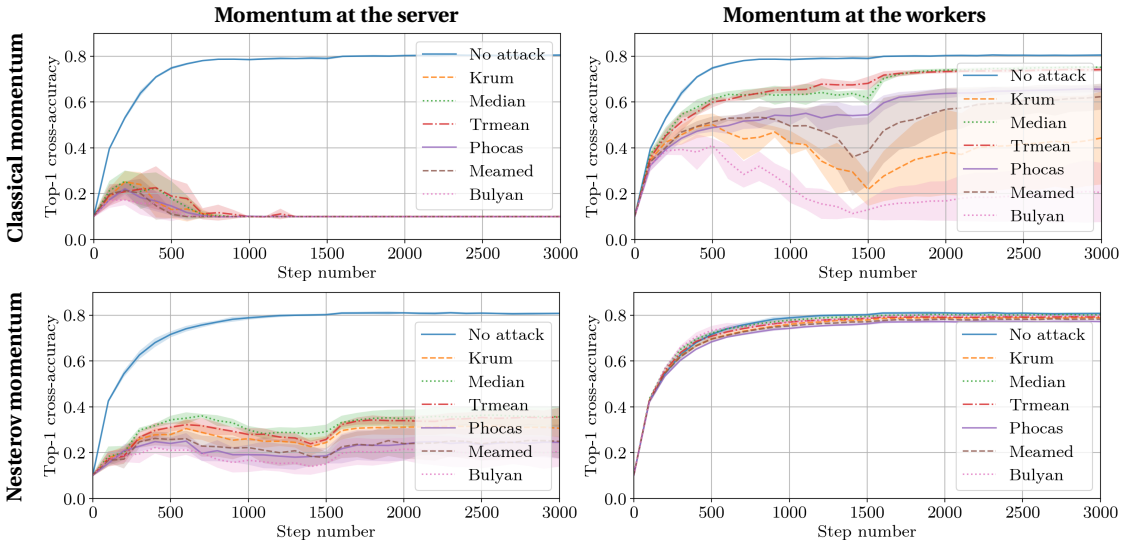


Figure 5.2: Top-1 cross-accuracy and average loss (over the $n - f$ honest workers). CIFAR-10 and the *convolutional* model (Section 5.4.1), with $n = 25$, $f = 5$ and $\eta_t = 0.01$ if $t < 1500$ else $\eta_t = 0.001$, under attack from (Baruch et al., 2019). Each line and colored surface correspond to respectively the average and standard deviation of the top-1 cross-accuracy over 5 seeded runs. Only two parameters change between these four graphs: where momentum is computed (*at the server* or *at the workers*), and which flavor of momentum is employed (*classical* or *Nesterov*).

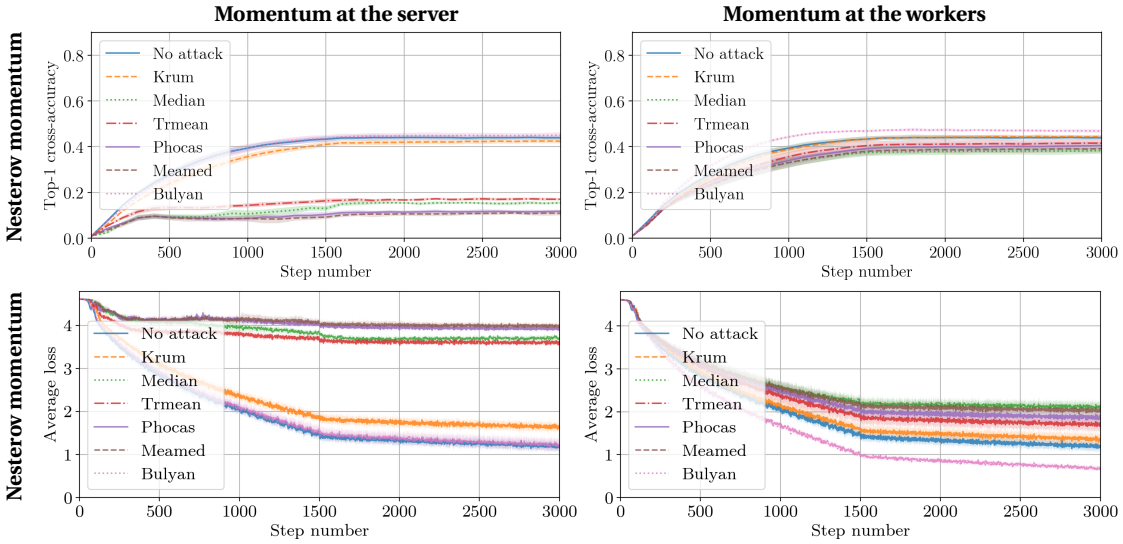


Figure 5.3: Top-1 cross-accuracy and average loss (over the $n - f$ honest workers). CIFAR-100 and the *convolutional* model, with $n = 25$, $f = 5$ and $\eta_t = 0.01$ if $t < 1500$ else $\eta_t = 0.001$, under attack from (Xie et al., 2019a). Each line and colored surface (barely visible due to a low variance across runs) correspond to respectively the average and standard deviation of the top-1 cross-accuracy or average loss over 5 seeded runs.

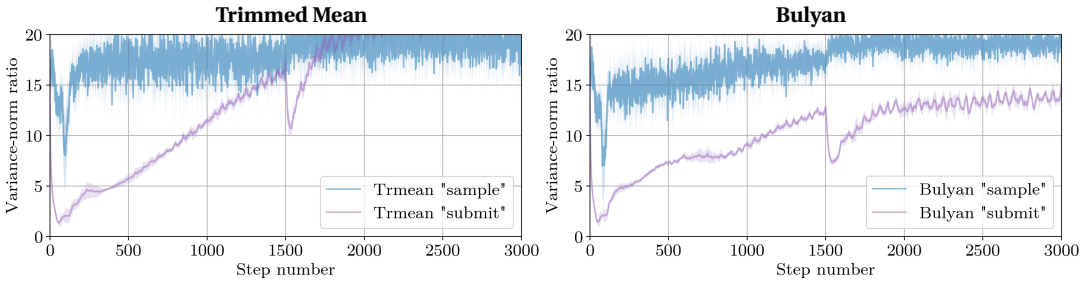


Figure 5.4: Same settings as in Figure 5.3, measuring at each step the empirical *variance-norm* ratios obtained by training with *Trimmed Mean* and *Bulyan* and *momentum at the workers*. "sample" corresponds to the *variance-norm* ratio of the sampled gradients, and "submit" to the *variance-norm* ratio of the *submitted gradients*.

variance-norm ratio of the submitted gradients. Section A.1.3 in the appendix reports on the entirety of our experimental results, and Section A.1.2 additionally experiments with a much larger model. One first remark is that our new formulation either obtains similar, or (substantially) increased maximum top-1 cross-accuracy compared to the standard formulation in the exact same settings (Figure 5.1 highlights the impact of our method). Namely, in only 4 runs (0.23% of all the runs) did our formulation lead to a decreased maximum top-1 cross-accuracy. Also, these decreases were only observed with the *fully connected* model, using *Krum* against Xie et al. (2019a), and for each of these 4 runs using any of the 4 other seeds made the decrease disappear.

In all of our experiments, we observe a strong correlation between higher top-1 cross-accuracies and lower average losses; e.g. see Figure 5.3. The two state-of-the-art attacks decreased the accuracy by at least 20%, compared to the unattacked case (see "No attack" in Figure 5.2), in 25.80% and 70.80% of the runs with respectively the *fully connected* and *convolutional* models.

Focusing on the *convolutional* model, when roughly *an half* of the workers are Byzantine, both attacks actually succeed in decreasing the accuracy by at least 20% in 100% of our runs. Our technique manages to recover at least 10% and 20% in respectively 79.75% and 49.25% of these runs. When roughly *a quarter* of the workers are Byzantine, the attacks decrease the accuracy by at least 20% in 46.46% of our runs. Our technique then manages to recover at least 20% in 95.07% of these runs. Figure 5.2 shows a fraction of these runs. El-Mhamdi et al. (2020b) reports on all of our experiments.

Technically, our reformulation aims at reducing the *variance-norm* ratio of the aggregated gradients. Intuitively, this ratio is expected to increase as the loss decreases; more correctly as the norm of the gradient decreases. For instance, Figure 5.4 displays the *variance-norm* ratios of *Trimmed Mean* and *Bulyan* using the same settings as in Figure 5.3. At least before the final cross-accuracy is reached, our technique consistently decreases the *variance-norm* ratio of the aggregated gradients. Also, we

consistently observed in the experiments that reducing the learning rate indeed reduces the *variance-norm* ratio (e.g. Figure 5.4, $t \geq 1500$).

5.5 Concluding Remarks

In this chapter, we have studied probably one of the most difficult assumptions to satisfy in practice, to safely use most *statistically-robust*, (α, f)-Byzantine resilient Gradient Aggregation Rules. The *variance-norm* ratio assumption is instrumental for this family of GARs, as it basically quantifies how *informative* the non-Byzantine gradients are, informative about the direction and length of the real gradient $\nabla Q(\theta_t)$.

The *variance-norm* ratio assumption is unfortunately rarely satisfied in practice. Ratios observed with academic models and datasets (e.g. Figure 5.4) are theoretically satisfying only to a few GARs, like *MDA*, and with f small compared to n.

The lack of validity of this theoretical requirement also has a practical impact. Both of the studied attacks rely on the frequent sampling of non-Byzantine, *outlier* gradients to pass harmful gradients through statistically-robust Byzantine resilient defenses Baruch et al. (2019), some specifically designed to increase the loss Xie et al. (2019a). In theory, the *variance-norm* ratio can be lowered to arbitrarily small values by increasing the batch-size, but this comes at higher, possibly unsustainable gradient estimation costs.

We propose a fairly simple tweak, that can substantially improve the effectiveness of existing statistically-robust defenses in practice. This tweak aims at reducing the *variance-norm* ratio, relying on the empirical observation that the real gradient does not change much between steps. To quantify this "rate of change", the relevant metric is the so-called *curvature* s_t of the parameter trajectory. If s_t is small enough, our tweak will reduce the *variance-norm* ratio compared to the *variance-norm* ratio of the sampled gradients; but if s_t becomes too large, our changes may become harmful.

In our 3680 runs[4], less than $1/400$ of them saw their top-1 cross-accuracies decrease due to the tweak we propose. Importantly, a substantial fraction of them saw their cross-accuracies leap, sometimes completely canceling the effects of the attack. And unlike increasing the batch-size, our tweak carries no computational complexity: it merely reorders existing computations associated with momentum.

This chapter may also highlight the relevance of the *variance-norm* ratio as a practical, predictive metric for the resilience of statistically-robust defenses. Interestingly, Karimireddy et al. (2020) defend the idea that "any optimization procedure which does not use history cannot converge", and as they note, "momentum [as done in this chapter] incorporates history". Karimireddy et al. (2020) also provide a proof of convergence.

[4]Without even counting the additional runs in Section A.1.2, which responded very well to our changes.

Part III
Optimized Implementations

6 Robust Aggregation in Practice

We present *AggregaThor*, the first framework to implement and assess Byzantine-resilient GAR in actual, datacenter-scale distributed settings. Byzantine resilience is certainly not *free*. The purpose of this chapter is to explore to what extend existing defense can affect the training performances, especially when there is no attack.

AggregaThor is built around TensorFlow[1], one of the major machine learning frameworks available today. Unlike previous implementations of (α, f)-Byzantine resilient GARs (e.g. in Section 3), we strive to write optimized, parallelized and *specialized* implementations. Our native implementations systematically beat implementations of the same GARs with TensorFlow and PyTorch[2]. We describe our design for *AggregaThor*, including a modification toward making TensorFlow viable for Byzantine deployments, and compare its performances against "vanilla" usages of TensorFlow.

The theoretical framework is the same as the one presented in Section 2.2.

6.1 Design of AggregaThor

We can identify three different, high-level components that must be adapted to work together in *AggregaThor*. First, the "vanilla" training procedure of TensorFlow, with its way of defining model, loading datasets, and running SGD. Second, the gradient aggregation rules, their implementations in native, specialized code, the support for automated, incremental compilation, and a convenient, user-friendly interface to use these GARs inside TensorFlow. Third, the communication layer, the distribution of a workload across a cluster, and other specificities of TensorFlow: blocking communications, the remote *graph* execution feature and utter absence of access control.

[1] This system chapter has been designed against TensorFlow 1.x (Abadi et al., 2016). Some remarks may not be up-to-date with TensorFlow 2.x, which underwent significant design changes in several areas.

[2] One of the other major machine learning frameworks freely available today; see Chapter 7.

6.1.1 Architecture and Byzantine resilience

AggregaThor is a framework that handles the distribution of the training of a TensorFlow neural network *graph* over a cluster of machines. The training procedure includes the use of an arbitrary GAR, which require accesses to the *flat* gradients computed by each worker. This introduces non-negligible changes over "vanilla" training pipelines.

Cluster definition

The first step to begin a distributed training session is to define the cluster, i.e.: which nodes will partake in the distributed training session, and under which roles. There are three possible roles[3]: *parameter server* (only one node can assume this role), *worker*, and *evaluator*. Evaluator nodes do not appear in the theoretical model (Section 2.1.2): their responsibility is simply to carry out periodic cross-accuracy measurements.

The cluster specification is a JSON string representing a dictionary, mapping each of the three job names to a list of "hostname:port" strings. For instance:

```
{
    "ps": ["ps.example.com:7000"],
    "workers": ["worker0.local:1234", "1.2.3.4:5555", "localhost:5555"],
    "eval": ["[2001:db8::8a2e:370:7334]:7000"]
}
```

The semantic is that the resources accessible from the designed hosts (e.g. GPUs) in each job (e.g. *workers*) will be reserved and used by this job (e.g. spawn one (virtual) worker per available GPU in nodes registered as *workers*).

A single node can assume several roles. In particular in our experiments, the parameter server node is also the (single) evaluator node. This is achieved by (1) not defining the *eval* job in the cluster specification, and (2) later instructing *AggregaThor* to use the resources on the *ps* job to deploy the part of the graph carrying out the evaluation of the model. As a side note, *AggregaThor* supports one and only one *replica*[4] per *job*.

Specifying the cluster may be tedious and error-prone. *AggregaThor* is able to automatically infer a cluster specification when running on a supported PaaS provider. For instance, we use Grid5000 in our evaluation of *AggregaThor* (c.f. Section 6.2), and Grid5000 provides a mean to query which nodes belong to the same "reservation" as the node doing the query. Upon request, *AggregaThor* can automatically parse the information provided by Grid5000 to build the corresponding cluster specification.

[3] Each of these roles appear in Figure 6.1, under `/job:ps`, `/job:workers` and `/job:eval`. The notion of "job", which is a way to state the intended use for each resource of a cluster, is inherited from TensorFlow.

[4] This is another notion proper to TensorFlow, notion we do not use and thus ignore in this chapter.

Cluster analysis

Now that all the nodes and their respective roles are specified, the next step is to connect to each of these node, and instantiate a TensorFlow `tf.train.Server` on each of them. A Python script handles this deployment from a cluster specification.

The *deployment* Python script requires that (1) each node in the cluster is accessible from the local node via SSH, and (2) both Python 3 and TensorFlow are available on each node. This deployment script has no other requirement, and in particular it does not require access to persistent storage onto each node (e.g. to copy itself onto each node). The deployment script works by maintaining an interactive SSH tunnel with each node in the cluster, running `python3`, and uploading the code that instantiates the `tf.train.Server` directly within the remote Python interpreter. By maintaining interactive SSH tunnels, this procedures offers two other advantages: (1) remote errors are reported centrally (in the standard output of the deployment script), and (2) centralized termination of the whole cluster (when the deployment script terminates, this closes the tunnels, terminating all the `tf.train.Server` instances).

The `tf.train.Server` instances enable TensorFlow to connect to and control each node in the cluster. The remaining step before deploying and running the whole training graph is then to discover which resources are available, and plan their usage.

At this point, thanks to the cluster specification, *AggregaThor* already knows which nodes will host each part of the whole training graph (represented in Figure 6.1). The only remaining question is which processing units (CPU, first GPU, second GPU, etc) on each node will actually carry the operations. *AggregaThor* lists every available processing unit per job. The allocation of the available processing units to each worker follows a few rules. First, *AggregaThor* tries to maximize *spreading*. For instance if there are 5 nodes in the cluster, each carrying 2 GPUs, and *AggregaThor* is requested to spawn $n = 4$ workers, then 1 GPU on 4 different nodes will be selected instead of 2 GPUs on 2 different nodes. The second (optional) rule is that GPUs are preferred over CPUs (*AggregaThor* also supports TPUs, which are then preferred over both GPUs and CPUs). The third (optional) rule is that each GPU can be allocated to only one worker, and if n is larger than the number of available GPUs in the *workers* job, the remaining workers will use CPUs (allocated so to maximize *spreading*, satisfying the first rule).

Cluster deployment

Each operation constituting the whole training graph is then deployed onto the cluster as illustrated in Figure 6.1, using the devices allocated in the cluster analysis phase.

In TensorFlow, Byzantine resilience cannot be achieved solely through the use of a Byzantine-resilient GAR. Indeed, TensorFlow allows any node in the cluster to execute

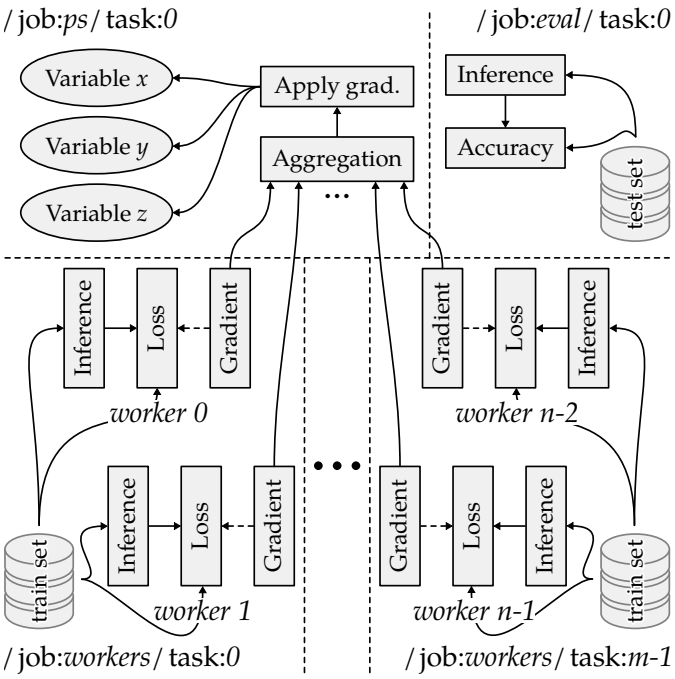


Figure 6.1: High-level components and execution graph. Each gray rectangle represents a group of several `tf.Operation` and `tf.Tensor`, and each plain arrow represents a single `tf.Tensor`. For readability purpose, the tensors from the variables to each "Inference" and "Gradient" groups of operations have not been represented.

arbitrary operations anywhere in the cluster. A single Byzantine worker could then continually overwrite the shared parameters with arbitrary values. This is actually how TensorFlow is supposed to work: each worker node defines his own parts of the graph, wherever needed in the cluster, to pull/update the shared parameters at each step.

One can overcome this issue in two steps. The first step is to modify the behavior of `tf.train.Server`, to only accept connections coming from a *trusted authority*. This involves setting up a proper *Public Key Intrastructure* (PKI), where for instance the single parameter server acts as the trusted authority. Once the connection between two `tf.train.Server` is established, messages (i.e. graph definitions and tensors) coming from the authority must at least be authenticated. Since these modifications involving cryptography are fairly complex and do not impact the training performances (the execution graph is created only once, and authenticated encryption is unlikely to induce a noticeable throughput hit[5]), they have not been implemented in *AggregaThor*. The second step is for each node to accept remote graph definitions only if they come from the trusted authority (e.g. the trusted parameter server). This second step changes the way we deploy graphs in *AggregaThor* compared to "vanilla" TensorFlow: the whole graph, as summarized in Figure 6.1, is defined by one entity.

[5]For a basic benchmark of ChaCha20-Poly1305: https://www.wireguard.com/performance/#results

There is one last subtlety during the deployment phase. Some graph operations cannot be sent onto a remote `tf.train.Server`. This is for instance the case with the `tf.py_func` operation, which calls a local Python closure when executed. This is also the case for our native GAR implementations (Section 6.1.2), as they are implemented in shared libraries that are not available in remote nodes. So when a custom GAR is used (i.e. for any GAR except *Average*), the trusted authority deploying the graph must always be the parameter server. This also changes the way we initially deploy the cluster with the *deployment script*: when using a custom GAR, this script does not deploy the `tf.train.Server` of the parameter server. The `tf.train.Server` of the parameter server is instead started from within the *AggregaThor* process, bypassing the need to transfer the GAR operations altogether.

6.1.2 Optimized GAR implementations

TensorFlow already offers many carefully-written tensor primitive operations. Naturally thought, no single primitive carries out a Byzantine resilient aggregation: most GARs are necessarily composed of many existing primitives. Such a composition might not be as performance- or memory-efficient as a custom operation, as custom operations might exploit shortcuts (as our implementation of *Bulyan* does) or specificities in the shape of the inputs (as our implementation of *Median* does, c.f. Section 7.1.2).

TensorFlow offers the ability to write custom operations[6]. Custom operations are fundamentally identical to the primitive operations of TensorFlow: they use the same facility to make themselves available to the Python runtime, and abide by the same interface. The Byzantine resilient GAR custom operations will be written in C++ (for the CPU implementations) and CUDA (for the Nvidia-compatible GPU implementations), and will be compiled as dynamically-loadable shared libraries.

Incremental compilation.

For convenience to both the developer of these GARs and the practitioner using *AggregaThor*, *AggregaThor* automatically handles the incremental compilation of each GAR. *AggregaThor* uses the preferred C++ compiler and linker of the platform[7]. The behavior for incremental compilation very closely follows the one of *GNU Make*[8]: basically, an object (e.g. `*.o`, `*.so`) is recompiled if and only if any of its dependencies (e.g. `*.cpp`, `*.hpp`) have a modification time *posterior* to the modification time of the object.

AggregaThor tries to automatically detect whether the CUDA software development

[6] https://www.tensorflow.org/guide/create_op

[7] `/usr/bin/c++`. *AggregaThor* only supports GNU/Linux-like platforms, and requires that a compilation toolchain is locally available if the GARs are not all already compiled.

[8] https://www.gnu.org/software/make/

toolkit and runtime are available, and whether a GAR exports GPU implementations. If either the CUDA toolkit or runtime is unavailable, *AggregaThor* will only compile and link the non-CUDA source code. This permits a graceful degradation in case the target platform does not have (Nvidia-compatible) GPUs, enabling the use of the CPU implementations only instead of merely failing the compilation process.

Automated loading and interfacing.

Whether or not incremental compilation was needed, to use the custom operations, they must be loaded into TensorFlow. *AggregaThor* automatically loads into the Python process every available (i.e. successfully compiled) GAR. Each custom operation GAR is made available to Python code as a Python function, under the same name as declared inside the custom operation interface. This allows for very convenient use of these custom operations, almost as if they were native TensorFlow operations, e.g.: instead of calling in Python `tf.my_gar`, one will call `native.instantiate_op("my_gar", ...)`.

In the literature the GAR expect to take *flat* gradients: one single vector in $\mathbb{R}^d$. In TensorFlow (and other frameworks), the parameter vector and gradients are actually split into several tensors. *AggregaThor* transparently merges the splits of each gradient into one *flat* gradient before forwarding them to the GAR, and splits the aggregated gradient back into pieces prior to updating each parameter vector.

Finally, *AggregaThor* does not only support TensorFlow custom operations. Besides custom operations, two other kinds of libraries are recognized: (1) "Python" libraries and (2) "dependency" libraries. These three supported kinds of libraries are built the same way; they only differ in how they are loaded into *AggregaThor*. In a nutshell, "Python" libraries are loaded with `ctypes`, allowing *raw* access to the exported symbols in Python, and the "dependency" libraries are simply not automatically loaded.

Dependency support.

AggregaThor supports dependencies between the native implementations. This is for instance useful when several GARs want to share common pieces of code. Arbitrary *acyclic* graphs of dependency can exist between native code. Compilation cycles are monitored, and an error will be raised if a cycle is detected during compilation.

Both the compilation and loading behaviors are changed by dependencies. If a library B depends on library A, library A will be built before library B. If library A fails to compile, the compilation of library B will be skipped. Library A will also be loaded before library B, and if the loading fails library B will not be loaded either. This is also where the "dependency" kind of library is useful: it is loaded only if at least one other library depends on it, and by construction compiled and loaded at most once.

6.1.3 Modularity by Design

From the standpoint of the practitioner, *AggregaThor* is mostly configured through the command line. Command line arguments configure which model to use, which dataset to train against, how many workers to launch, on which cluster, using which (Byzantine resilient) GAR, optimizer, learning rate, etc. The full list of command line arguments is available in the original publication (Damaskinos et al., 2019).

From the standpoint of the developer, *AggregaThor* is the framework which defines the *interfaces* between machine learning components (GAR, model, dataset, optimizer, etc), and manages the combination of a selection of these different *modules* into a training session. This modular approach, illustrated in Figure 6.2, trades versatility (modules are restricted by their respective interface, e.g. a GAR cannot directly access the parameter vector) for manageability (well-defined interfaces and responsibilities help deal with complexity) and "scriptability" (e.g. testing a wide range of GARs boils down to changing one parameter in a BASH script).

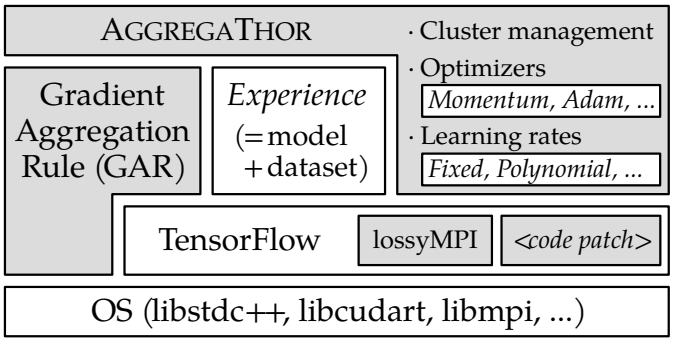


Figure 6.2: The components of *AggregaThor*, and their layered relations with existing components. New components have a gray background. *AggregaThor* acts as a light framework, managing the deployment and training of a model over a cluster. Please note that the *lossyMPI* component will not be discussed in this thesis[(a)].

[(a)] The designer and author of the implementation of this component is Arsany Guirguis.

Automated imports for convenience.

The code of *AggregaThor* needs not be modified when adding a new component, e.g. a GAR or a model. For instance, each GAR available in *AggregaThor* are actually written in different *Python modules* under directory `aggregators/`. Developing a new GAR consists in writing a new `.py` file in `aggregators/`. The GARs exported by this new Python module will then be imported by *AggregaThor*, and made available as new possible choices in the command line options.

Through several high-level libraries (e.g. https://github.com/google-research/tf-slim), TensorFlow also offers a large range of models and datasets. *AggregaThor* supports

automatically re-exporting models and datasets from some of these libraries, so that they can be selected in the command line by the practitioner. The advantage of such an automated interfacing is that any new contribution in one of the supported, external libraries will also become available in *AggregaThor*.

Custom arguments from the command-line.

The pair model-experiments, most optimizers and some GAR may take additional parameters. This is the case with *Krum* (Section 3.1.1) for instance, which take an additional, optional parameter m.

Such module-dependent arguments can be specified via the `--*-args` command line arguments. The expected format of these optional arguments is also technically module-dependent, although the usual format all modules currently follow is a list of one or more `property:value` strings, e.g. `batch-size:32` or `m:4`.

6.2 Evaluation of *AggregaThor*

We evaluate the performance of *AggregaThor* over an academic image classification task (CIFAR-10), due to its high-availability and wide adoption as a benchmark for distributed ML literature (Chilimbi et al., 2014; Abadi et al., 2016; Zhang et al., 2017).

6.2.1 Evaluation Setup

We present the details of the configuration, benchmarks, and methods we employ for our evaluation. For clarity and for the rest of this section, we will refer with *Krum* to the deployment of *AggregaThor* with the GAR being *Krum* and with *Bulyan* to the deployment of *AggregaThor* with the GAR being *Bulyan*.

Platform. Our experimental platform is Grid5000[9]. Unless stated otherwise, we employ 20 nodes, all within the same datacenter, each carrying 2 CPUs (Intel Xeon E5-2630) with 8 cores each, 128 GiB RAM and 10 Gbps Ethernet.

Dataset. We use the CIFAR-10 dataset (Krizhevsky et al., 2009), a widely used dataset in image classification (Srivastava et al., 2014; Zhang et al., 2017), which consists of $60\,000$ colour (RGB) 32×32 images in 10 classes. We perform min-max scaling as a pre-processing step for the input features of the dataset. We employ a convolutional neural network with a total of 1.75M parameters as shown in Table 6.1. We have implemented the same model with PyTorch to be compatible with *Draco* (Chen et al., 2018).

[9] https://www.grid5000.fr/

	Input	Conv1	Pool1	Conv2	Pool2	FC1	FC2	FC3
Kernel size	32×32×3	5×5×64	3×3	5×5×64	3×3	384	192	10
Strides		1×1	2×2	1×1	2×2			

Table 6.1: Architecture of the "CNN model".

Evaluation metrics.

We evaluate the performance of *AggregaThor* using the following standard metrics.

Throughput. This metric measures the total number of steps (i.e. gradient computation, aggregation and parameter update) executed per second. The factors that affect the throughput is the time to compute a gradient, the communication delays (worker receives the model and sends the gradient) and the idle time of each worker. The idle time is determined by the overhead of the aggregation at the server. While the server performs the aggregation and the descent, the workers wait (synchronous training).

Accuracy. This metric measures the top-1 cross-accuracy: the fraction of correct predictions among all the predictions, using the *testing* dataset (see below). We measure accuracy both with respect to the passage of time and increasing step numbers t.

Evaluation scheme.

To cross-validate the performance, we split the dataset into *training* and *test sets*. The dataset includes 50,000 training examples and 10,000 test examples. Note that, if not stated otherwise, we employ an RMSprop optimizer (Tieleman and Hinton, 2012) with a fixed initial learning rate of 10^{-3} and a mini-batch size of 100.

We split our 20 nodes into $n = 19$ workers and 1 parameter server. If not stated otherwise, we set $f = 4$ given that *Bulyan* requires $n \geq 4f + 3$.

We employ the fastest-to-convergence combination of other hyperparameters for the deployment of *Draco*. For example, we use the *repetition* method, because it gives better results than the *cyclic* one. We use the *reversed gradient* adversary model, with its parameters as recommended by the original authors, and a momentum of 0.9.

6.2.2 Non-Byzantine Environment

In this section, we report on the performance of our framework in a non-Byzantine distributed setup. Our baseline is *vanilla* TensorFlow (TF) deployed with the built-in averaging GAR: *tf.train.SyncReplicasOptimizer*. We compare TF against *AggregaThor* using (a) *Krum*, (b) *Bulyan*, (c) the coordinate-wise *Median* (Xie et al., 2018b), and (d)

the basic arithmetic mean (*Average*). We also report on the performance of (e) *Draco*.

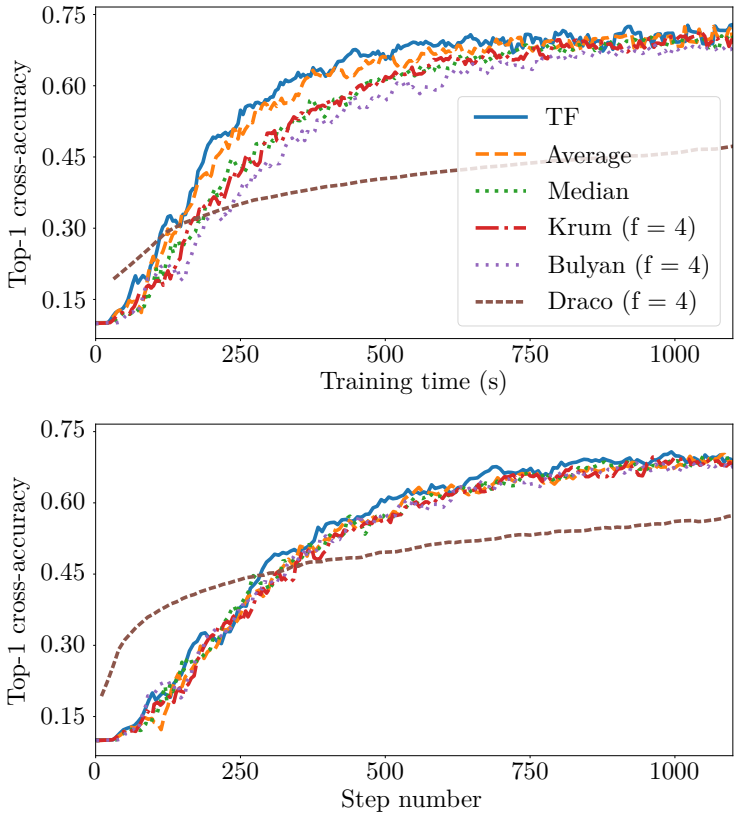


Figure 6.3: Overhead of *AggregaThor* in a non-Byzantine environment ($b = 250$).

Overhead in terms of convergence time.

In Figure 6.3, TensorFlow reaches 50% of its final accuracy in 3 minutes and 9 seconds, whereas *Krum* and *Bulyan* are respectively 19% and 43% slower for reaching the same accuracy. Our framework with *Average* leads to a 7% slowdown compared to the baseline. The *Median* GAR, with a mini-batch size of $b = 250$, converges as fast as the baseline (model update-wise), while with $b = 20$ (Figure 6.4), *Median* prevents convergence to a model achieving baseline accuracy.

We identify two separate causes for the overhead of *AggregaThor*. The first is the *computational overhead* of carrying out the Byzantine-resilient aggregation rules. The second cause is the inherent *variance increase* that Byzantine-resilient rules introduce compared to *Average* and the baseline. This is attributed to the fact that *Krum*, *Bulyan* and *Median* only use a fraction of the computed gradients; in particular *Median* uses only one gradient. Increasing the variance of the gradient estimation is a cause of convergence slowdown (Bottou, 1998). Since even *Median* converges as fast as the baseline with $b = 250$, the respective slowdowns of 19% and 43% for *Krum* and *Bulyan* corre-

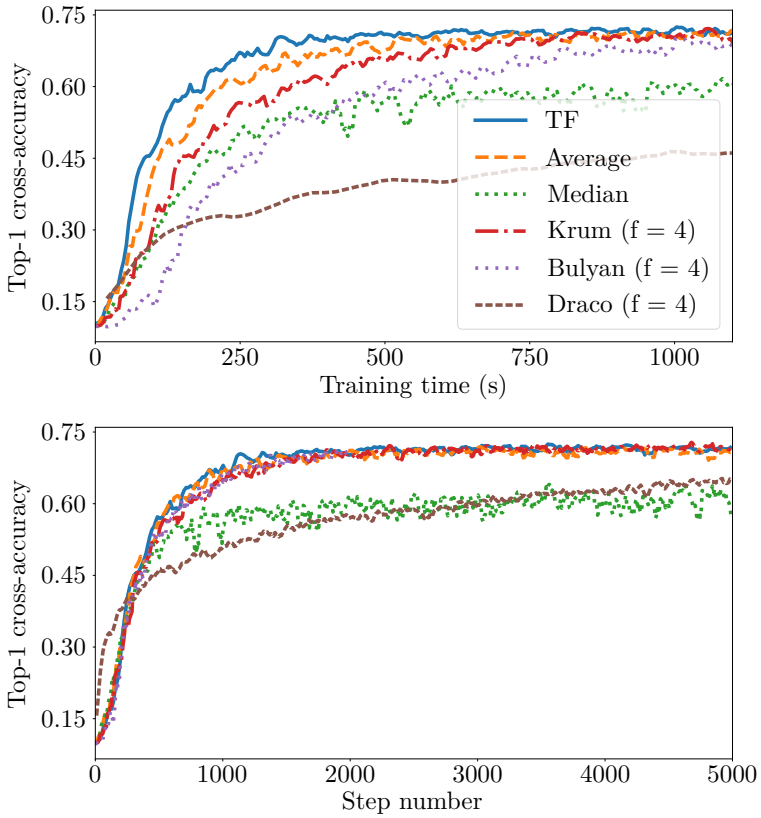


Figure 6.4: Overhead of *AggregaThor* in a non-Byzantine environment ($b = 20$).

spond *only* to the computational overhead. The practitioner using *AggregaThor* does not need to increase the mini-batch size to achieve baseline final accuracy (Figure 6.4).

Although *Draco* reaches the same final accuracy, the time to reach the model's maximal accuracy is slower than with our TensorFlow-based system. We attribute this mainly to the fact that *Draco* requires $2f + 1$ times more gradients to be computed than our system before performing a step.

We decompose the average latency per step to assess the effect of the aggregation time on the overhead of *AggregaThor* against TensorFlow. We employ the same setup as in Figure 6.3. Figure 6.5 shows that the aggregation time accounts for 35%, 27% and 52% of run times of *Median*, *Krum*, and *Bulyan* respectively. These ratios do not depend on the variance of the aggregated gradients, but solely on the gradient computation time: a larger/more complex model would naturally make these ratios decrease (i.e., the *relative cost* of Byzantine resilience would decrease). See Figure 6.6.

Impact of f on scalability.

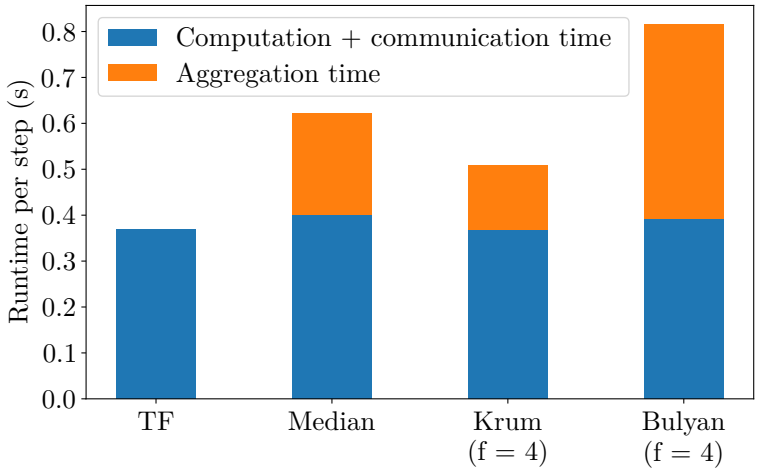


Figure 6.5: Latency breakdown ($b = 250$).

We measure the scalability of *AggregaThor* with respect to TensorFlow for two models: the CNN that we use throughout the evaluation and a significantly larger one, ResNet-50. Figure 6.6 shows that the throughput of all TensorFlow-based systems with up to 6 workers is the same. From this point on, the larger the number of workers, the larger the deviation between the Byzantine-resilient algorithms and TensorFlow. The reason behind this behavior is the fact that an increase in the number of workers forces the GARs (especially *Krum* and *Bulyan* which are quadratic in n) to do more and more operations. For example, *Bulyan* scales poorly for this setup. This is confirmed with ResNet-50 in Figure 6.6, where gradient computation is significantly more costly than gradient aggregation, mechanically enabling the GARs to display better scalability.

Figure 6.6 also indicates that the higher the declared f the higher the throughput (except for *Draco*). This is the direct outcome of the algorithmic design of the two statistically-robust GARs assessed in these experiments. Since $m = n - f - 2$, the higher f the fewer *iterations* for *Krum* (Blanchard et al., 2017) and *Bulyan* (El-Mhamdi et al., 2018). Of course, there actually is a trade-off between the update throughput and the variance of each aggregated gradient. Indeed: f controls how many gradients are aggregated. Higher f translates into fewer (non-Byzantine) gradients averaged by *Krum* (Blanchard et al., 2017) and *Bulyan* (El-Mhamdi et al., 2018), which might eventually hamper the training even without Byzantine workers.

Draco is always at least one order of magnitude slower than the TensorFlow-based systems. This low throughput limits its scalability. An interesting observation here is that changing the number of Byzantine workers does not have a remarkable effect on the throughput. This is attributed to the method Draco uses to handle Byzantine behaviors, linear in n (Chen et al., 2018), and thus remains unaffected when f changes.

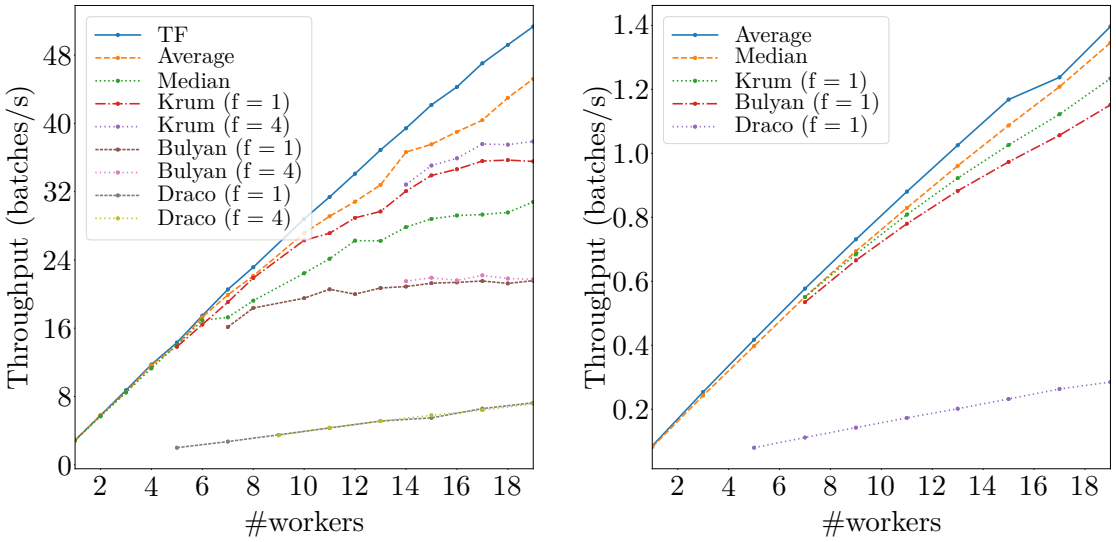


Figure 6.6: Throughput comparison, higher is better. Left: the model defined in Table 6.1, with 1.75M parameters. Right: the model is ResNet-50, with 23.5M parameters. The scalability to higher worker counts mechanically improves as the gradient estimation time increases with larger models.

Impact of f on convergence.

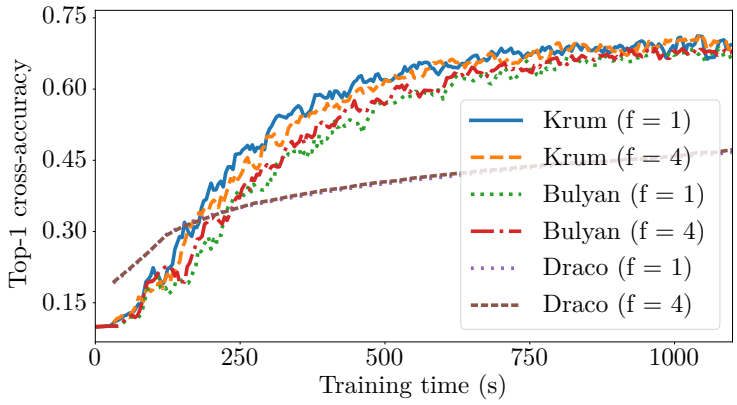


Figure 6.7: Impact of f on convergence ($b = 250$).

We show the effect of the choice of f in a non-Byzantine environment. Figure 6.7 shows that the larger value of f triggers a slightly slower convergence for *Krum* and slightly faster convergence for *Bulyan*. This is the direct consequence of the aforementioned trade-off. The throughput of *Krum* is boosted more than the throughput of *Bulyan* for the same increase on f (from 1 to 4). Therefore, in the case of *Bulyan*, the faster model updates compensate for the additional noise whereas in the case of *Krum* the throughput boost is not enough. For a smaller mini-batch size (Figure 6.8) the

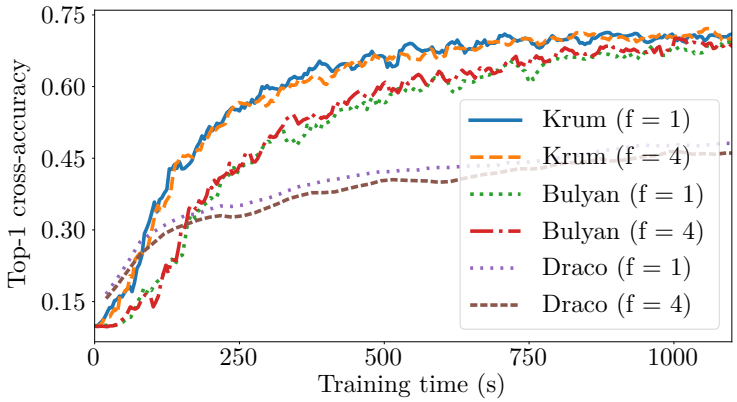


Figure 6.8: Impact of f on convergence ($b = 20$).

behaviour is similar but the impact of f is smaller. This is because the mini-batch size is an important parameter in practice, that can substantially affect the trade-off between the update throughput and the quality of each update (Akiba et al., 2017).

6.2.3 Adversarial Environment

We evaluate *AggregaThor* in a distributed setting under attack. We report on a *weak*, but arguably realistic family of adversarial behaviors: corrupted training set.

Corrupted data.

We highlight here the fact that TensorFlow cannot tolerate even one Byzantine worker (which employs corrupted data in this scenario) while *AggregaThor* can tolerate that.

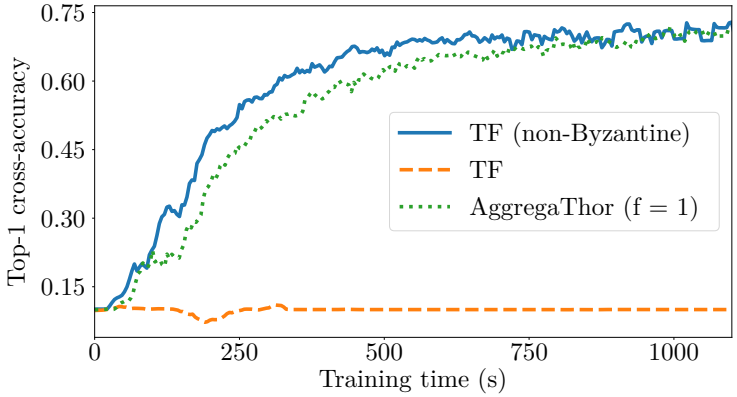


Figure 6.9: Impact of malformed input on convergence.

Figure 6.9 shows that for a mini-batch size of 250, the convergence behavior of *AggregaThor* is similar to the ideal one (TensorFlow in a non-Byzantine environment).

We thus highlight the importance of Byzantine resilience even for this "mild" form of Byzantine behavior (only one worker sends corrupted data) to which TensorFlow is intolerant (TensorFlow diverges). The use of Byzantine-resilient GARs, such as *Krum*, is naturally enough to thwart this kind of mild attack.

Experimenting with more complex attacks, like the ones presented by Baruch et al. (2019) and Xie et al. (2019a), would not bring anything new compared to e.g. the comprehensive study reported by Chapter 5. One of the main added value in this chapter is arguably the *time component* of the experiments: measuring the impact of Byzantine-resilient GARs over the total runtime of actual, distributed training sessions. The runtime of the studied GARs is fairly decorrelated from the presence of Byzantine vector, and to carry out complex attacks (which runtime is comparable to one aggregation) would likely bias our throughput and total runtime measurements.

6.3 Concluding Remarks

In this chapter we described the overall design of *AggregaThor*, historically the first framework to integrate Byzantine resilient gradient aggregations into TensorFlow. The design of *AggregaThor* significantly differs from the one of "vanilla" deployments with TensorFlow. We have seen that TensorFlow, and particularly through the way it handles distributed deployments, is not designed with Byzantine resilience in mind. We described in high-level terms what could be done to mitigate these issues, and other design choices made during the development of *AggregaThor*.

We have evaluated *AggregaThor* in datacenter-wide distributed deployments, using academic machine learning problems (CIFAR-10 with a simple convolutional network). Our baseline is "vanilla" TensorFlow, which as we show does not support adversarial behaviors during training. There are two main takeaways. The first takeway is that Byzantine resilient GARs, such as *Krum* with $m = n - f - 2$ and *Bulyan* of *Krum* (also with $m = n - f - 2$) do not induce any accuracy loss when there is no Byzantine worker. This is not the case for *Median*, at least with small batch sizes. The second takeway is that Byzantine resilient GARs may take a significant fraction of the total runtime (up to 50% for *Bulyan*) with small models ($d < 2$ M), although this runtime highly depends on which GAR is used (in our experiments, *Krum* takes half as long as *Bulyan* to run). Nevertheless with larger models ($d > 20$ M) and $n < 20$, the time spent estimating and transferring gradients becomes relatively larger than the time spent aggregating gradients, (subtantially) decreasing the throughput loss induced by the GARs.

The next chapter explores whether, like for gradient estimations, implementing GARs on GPUs can provide a significant speedup, potentially marginalizing the cost of statistically-robust Byzantine resilience in actual distributed deployments.

7 Faster Aggregation on GPUs

In Chapter 6, we have seen that existing Byzantine resilient GARs, such as *Krum* and *Bulyan*, carry a non-negligible runtime cost in practice; at least when running on CPUs. The question we explore in this chapter is whether implementing and running these GARs onto GPUs can provide a significant speedup over CPUs implementations.

For this chapter, we ported over PyTorch all the components of *AggregaThor*, except for the cluster and graph distribution parts. In this chapter, we will solely compare the *aggregation times* of each GAR. In addition to optimized GPU implementations, we also implement a version of *Krum* and *Bulyan* using only PyTorch's primitive operations.

7.1 Programming for GPUs

We briefly discuss the main differences to take into account to implement algorithms for CPUs vs. GPUs. We then dissect a practical implementation case.

7.1.1 Execution and Memory Considerations

CPUs offer a handful of pipelined, branch-predicted, (mostly[1]) independent *cores*. With the presence of a close (on-die), hierarchical cache, CPUs are basically made to execute instructions as fast as possible (especially when data and code fit in cache).

In comparison, GPUs feature hundreds more cores, offering thousands more hardware threads. But these cores are also much simpler: no branch-prediction and most importantly, hardware threads are grouped into blocks of e.g. 32 threads executing the same instruction[2]. GPUs also have high-throughput interconnects between main memory and cores, used optimally when the memory access patterns allow coalesced

[1] Different cores may fight for memory bandwidth, affecting each other's instruction throughput.
[2] Modern GPUs, e.g. since the Volta microarchitecture for Nvidia, may not have this constraint anymore.

data fetches (e.g. when all the threads from the same block simultaneously access adjacent memory). GPUs are basically made to process large chunks of data in parallel.

Since GARs are fundamentally concurrent workloads, we should highlight there are also differences regarding the memory model of CPUs and GPUs. CPUs, even with multiple sockets, allow any pair of hardware threads to synchronize on any (aligned) memory address. With CPUs, the memory is hierarchical[3] and remains coherent: L1d/i, L2(, L3), RAM, NVM/SSD/hard-drive/etc. With CUDA-compatible GPUs, while the silicon is not fundamentally different than with CPUs (there is both a RAM-like off-die memory, and a cache-like on-die memory), the model is different and several types of memories exist at what would be the same level for CPUs. There is a separation between "global" memory and "local" memory, which are both RAM-like off-die memories, but differ in how data is stored and how accesses can be coalesced. Some types of on-die memory (e.g. *texture* and *surface* memories) are not kept coherent with off-die memory. Regarding atomic operations, atomicity can be guaranteed either across all GPUs (like with CPUs), or instead only single GPU-wide or even only thread group-wide.

7.1.2 Practical case: SIMT median on GPUs

Median is not only a GAR, it is also a crucial component of *Bulyan*. Maximizing its performance in the context of gradient aggregation thus kills two birds with one stone.

Our implementation of the *Median* function on CPU is quite straightforward: each of the $m \geq 1$ available cores processes a continuous share of $\frac{n}{m}$ coordinates. Then each core applies, for each coordinate of its share, *introselect* (or equivalent) by calling the standard C++ `std::nth_element`.

Nevertheless, even embarrassingly parallel algorithms like *Median* would not necessarily get all the benefits from running on GPUs. That is because modern GPUs, to achieve parallel execution on many threads while limiting instruction fetch costs, batch threads into groups of e.g. 32 threads that execute the same instruction[4]. Algorithms like *introselect* (Musser, 1997) are branch-intensive, with possibly many instructions executed in each branch, and so, may fail to scale on GPUs.

Reminiscent of (Kachelrieß, 2009), our implementation of median is built around a primitive that orders 3 elements without branching. This is made possible through the use of the *selection instruction*, which converts a *predicate* into an integer value.

Let `v` be a table of 3 elements to reorder by increasing values. The selection instruction enables our implementation to compute:

[3] Here we are not considering latency, which may not only depend on the level of the accessed data.
[4] In case of branching, each branch will be executed one after the other, blocking the other threads.

```
int[] c = { v[0] > v[1], v[0] > v[2], v[1] > v[2] },
```

where `a > b` equals `1` if `a > b`, else `a > b` equals `0`.

Then using the intermediate results (found by solving the "reordering truth tables"):

```
int[] i = {
    (1+c[0]+2*c[1]+c[2]-(c[1]⊕c[2]))/2,
    (4-c[0]-2*c[1]-c[2]+(c[0]⊕c[1]))/2 },
```

we can finally reorder the elements of `v` into `w` with:

```
int[] w = { v[i[0]], v[3-i[0]-i[1]], v[i[1]] }.
```

Using this reordering primitive, we manage to implement an efficient version of *Median* with minimal branching, tailored for machine learning usages (Figure 7.1, "Median (PyTorch)" vs. "Median (custom)") where n is fairly small and d can be very large.

7.2 Experiments

We report on the performance of our implementations over two metrics:

1. the aggregation time of our implementations of *Median*, *Krum* and *Bulyan*, compared to the implementation of *Median* in PyTorch 1.6 (CUDA 10.1), and
2. the maximum top-1 cross-accuracy reached on a commonly used classification task in the ML litterature, compared to mere averaging and *Median*.

7.2.1 Setup

We run our experiments on the following hardware: (CPU) Intel® Core™ i7-8700K @ 3.70GHz, (GPU) Nvidia GeForce GTX 1080 Ti, and (RAM) 64 GB.

We report on the aggregation time, i.e. the time needed by a GAR to aggregate its input gradients and provide the output gradient. This metric is arguably the empirical counterpart of the asymptotic complexity, respectively $\mathcal{O}(n^2 d)$, $\mathcal{O}(n^2 d)$ and $\mathcal{O}(nd)$ for *Krum*, *Bulyan* and *Median*. To study the empirical behaviors of *Krum* and *Bulyan* compared to *Median*, we then vary both n and d over a realistic range of values. Namely we set $(n, d) \in \{7, 9, 11, \ldots, 35, 37, 39\} \times \{10^5, 10^6, 10^7\}$ and $f = \left\lfloor \frac{n-3}{4} \right\rfloor$.

The protocol for one run on the GPU is the following:

1. n gradients are independently sampled in $\mathcal{U}(0,1)^d$.
2. These gradients are moved over to the GPU main memory.
3. The command queue is then flushed on the GPU with `torch.cuda.synchronize()`, ensuring no kernel is pending on the CUDA stream.
4. The timer is then started.
5. The GAR is called on the GPU with the n input gradients.
6. The command queue is then flushed again, waiting for the GAR's execution to fully complete.
7. The timer is finally stopped, giving the execution time of one run.

When running the GAR on the CPU, the protocol is the same minus steps 2., 3. and 6. There are 7 runs per values of (n,d), from which we remove the first execution time (as it often looked like an outlier, especially on GPU), and we report on the average and standard deviation of the 6 other measurements in Table 7.1, Table 7.2, Figure 7.1.

We report on the maximum top-1 cross-accuracy reached by a distributed training process using either *Krum*, *Bulyan*, *Median* or mere averaging for aggregation. We set $n = 11$ workers and $f = 2$. There is no attack thought: this experiment highlights the benefits of averaging more gradients per aggregation step, as *Krum* and *Bulyan* do, over aggregation rules that keep (the equivalent of) only one gradient, e.g. *Median*.

The classification task we consider is Fashion-MNIST (Xiao et al., 2017), with $60\,000$ training points and $10\,000$ testing points. The model that we train is a convolutional network, composed of two 2D-convolutional layers followed by two fully-connected layers. The first convolutional layer has 20 channels (kernel-size 5, stride 1, no padding) and the second 50 channels (same kernel-size, stride and padding). Each convolutional layer uses the *ReLU* activation function followed by a 2D-maxpool of size 2×2. The first fully-connected layer has 500 hidden units, employing *ReLU*, and the second has 10 output units. We train the model using a cross-entropy loss (*log-softmax* normalization + *negative log likelihood* loss) over 3000 steps, with a fixed learning rate of 0.1 and momentum 0.9. To compute their gradients, each worker employs minibatches of size $b \in \{5, 10, 15, \ldots, 45, 50\}$. Every 100 steps we measure the top-1 cross-accuracy of the model over the whole testing set, and we keep the highest accuracy achieved over the whole training. For reproducibility purpose we seed each training, repeated 5 times with seeds 1 to 5. We report on the average and standard deviation of the highest accuracy achieved using each GAR and batch size in Figure 7.2.

d	*Average*	*Median* (PyTorch) *Median* (custom)	*Krum* (PyTorch) *Krum* (custom)	*Bulyan* (PyTorch) *Bulyan* (custom)
10^5	0.23 ± 0.049	69 ± 0.93 2.4 ± 0.48	4.5 ± 0.039 2.7 ± 0.030	125 ± 1.1 6.5 ± 0.071
10^6	5.03 ± 0.34	707 ± 9.2 21 ± 2.7	46 ± 0.44 29 ± 0.51	1255 ± 24 59 ± 1.1
10^7	82 ± 0.058	7000 ± 86 208 ± 4.7	786 ± 0.29 348 ± 1.0	13047 ± 93 630 ± 2.3

Table 7.1: GAR execution time (ms) on CPU (over 6 runs), $n = 15$.

d	*Average*	*Median* (PyTorch) *Median* (custom)	*Krum* (PyTorch) *Krum* (custom)	*Bulyan* (PyTorch) *Bulyan* (custom)
10^5	0.11 ± 0.0012	1.0 ± 0.028 0.26 ± 0.00066	3.9 ± 0.18 1.4 ± 0.41	5.9 ± 0.37 1.9 ± 0.26
10^6	0.55 ± 0.0011	7.0 ± 0.0015 0.92 ± 0.0020	8.9 ± 0.040 6.5 ± 0.0076	23 ± 0.012 8.3 ± 0.0085
10^7	5.1 ± 0.0014	70 ± 0.0081 8.1 ± 0.61	54 ± 0.039 53 ± 0.46	190 ± 0.31 71 ± 0.37

Table 7.2: GAR execution time (ms) on GPU (over 6 runs), $n = 15$.

7.2.2 Experimental Results

Table 7.1 and Table 7.2 compare the runtime of the GARs we implemented (except for *Average* and *Median* (PyTorch), which are both primitive operations in PyTorch), respectively on CPU and on GPU (hardware specifications are listed in Section 7.2.1). In particular, these tables compare our specialized, optimized, "custom" GAR implementations against the implementations made of one or several PyTorch primitives.

We can make two observations. First we did not implement, in our "custom" implementation of *Krum*, any computational/memory shortcut that cannot be implemented with PyTorch primitives; and we observe that our custom implementation is not much faster than the PyTorch one. More precisely, performances on GPU are almost identical[5]. Regarding the CPU, we noticed that PyTorch 1.6 did only use half of the available hardware threads, corresponding to the number of physical cores. This is sub-optimal for memory-bound workloads, such as medium to large gradient aggregation. On CPU, the PyTorch implementation[6] of *Krum* is approximately twice as slow as our custom implementation, which can be explained by PyTorch's sub-optimal use of CPU resources. This first observation may indicate that, if our other custom implementations are faster than their PyTorch counterparts, this would mostly be due to the shortcuts

[5] Especially with larger values of d, which hides the higher fixed costs of the PyTorch implementation.
[6] The *exact same* "PyTorch implementation" of a GAR is used to run on both CPU and GPU.

we found, and not the PyTorch primitives being intrinsically slower than necessary.

The second observation is that our other custom implementations are indeed faster, on both CPU and GPU, than their PyTorch counterparts. The custom implementation of *Median* employs the GPU-friendly trick detailed in Section 7.1.2. The custom implementation of *Bulyan* additionally fuses Equation (3.2) (Section 3.3.1) into one single, parallel operation. These optimizations can lead to drastic speedups, $\times 10$ and more.

The original question this chapter asked is whether running (Byzantine resilient) GARs on a GPU, instead of a CPU, can provide a significant speedup. Comparing Table 7.1 with Table 7.2, the harmonic mean of the aggregation speedups is above $\times 15$ when $d = 10^7$ (i.e. slightly less parameters than in ResNet-18). Since the communication time will not depend on the use of GPUs, such a speedup could substantially reduce the share of runtime spent in gradient aggregation (c.f. Figure 6.5 for the breakdown on CPU).

In Figure 7.1, the first observation that we can make is that the computational cost of both *Krum* and *Bulyan* indeed appears quadratic in n, the number of workers. The number of workers n is kept below 24 for *Bulyan* due to a limited amount of available on-die *shared memory* on the GPU we used. Regarding *Median*, for which we expect a linear increase with n, the tendency is not clear. The *Median* implementation provided by PyTorch shows a two-phase behavior, and our tailored implementation of *Median* is much faster[7] but exhibits a slightly superlinear behavior.

In Figure 7.1, and despite a higher asymptotic complexity, *Krum* and *Bulyan* achieve lower aggregation times than *Median* (PyTorch implementation) for respectively $n \leq 7, n \leq 9\,(d = 10^5)$, $n \leq 15, n \leq 13\,(d = 10^6)$ and $n \leq 17, n \leq 15\,(d = 10^7)$. Essentially, the higher the dimension of the model, the higher the number of workers up to which *Bulyan* is more competitive than the *Median* (PyTorch implementation). Our custom implementation of *Median* is faster than any other GAR in every tested setting.

For reference, ResNet-50 contains $d \approx 24M$ parameters. For such neural network sizes, major DNN frameworks already show scaling issues when employing only 8 workers (Luo et al., 2018). This inherent limitation the practitioner has to apply on the number of workers not to saturate the standard parameter server (even when using high-throughput 56 Gbps *IP-over-InfiniBand* networks (Luo et al., 2018)) would actually make *Krum* and *Bulyan* faster than *Median* (PyTorch implementation) in reasonable deployments ($n < 20$). The steady performance of *Krum* is mostly explained by the fact that its most computationally intensive part, the gradients' pairwise distances computation, is also naturally parallelizable on GPU: it consists in many additions and multiplications executed in parallel. The remaining computations for *Krum* merely consists in ordering *scalar* values. The same applies for *Bulyan*: our implementation

[7] NB: the implementation of PyTorch does unnecessary work by also returning the selected indexes.

does the costly pairwise distance computation only once, and since $f \approx \frac{n}{4}$ the median of *Bulyan* is computed over a substantially reduced set of pre-aggregated gradients.

The empirical "slowdown" effect of each GAR is captured in Figure 7.2. Each of the studied GAR *throw away* gradients that are, in these experiments, all correct. Compared to mere averaging the $n = 11$ gradients, aggregating less gradients per step has a tangible impact on the model performance: either more training steps, or higher batch sizes per worker, is needed to compensate. By averaging only (the equivalent of) one gradient per step, *Median* (no matter the implementation as both our implementation and PyTorch's implementation return the same values) shows in this Byzantine-free settings a tangible loss in top-1 cross-accuracy compared to *Bulyan* and *Krum*, which both achieve almost the same performance as averaging.

7.3 Concluding Remarks

In this chapter, we asked and answered whether Byzantine resilient GARs can benefit from running on GPUs. The answer is yes, and the aggregation speedup is sizeable: from $\times 6.6$ (*Krum*) to $\times 26$ (*Median*) when $d = 10^7$ (roughly as large as ResNet-18).

Our results also highlight to what extent fusing operations can improve aggregation throughput, compared to combining several of the primitive operations provided e.g. by PyTorch. The arguably substantial speedup obtained with our GPU implementation of *Median* is actually the product of two factors: (1) an efficient construction of the algorithm on GPUs, and (2) the fact that the PyTorch *Median* primitive, by also returning the selected indices, does unnecessary work for the purpose of Byzantine resilience.

Finally, when there is no (obvious) shortcut or opportunity to fuse primitive operations together, there may not be any tangible performance gain with a custom implementation. This is a rather important result for the practitioner, as developing a custom implementation can be fairly complex and time-consuming. This result does not appear obvious, in particular for PyTorch-based implementations, as combining primitive operations in PyTorch implies running parts of the aggregation in the Python interpreter, which could have induced a non-negligible overhead.

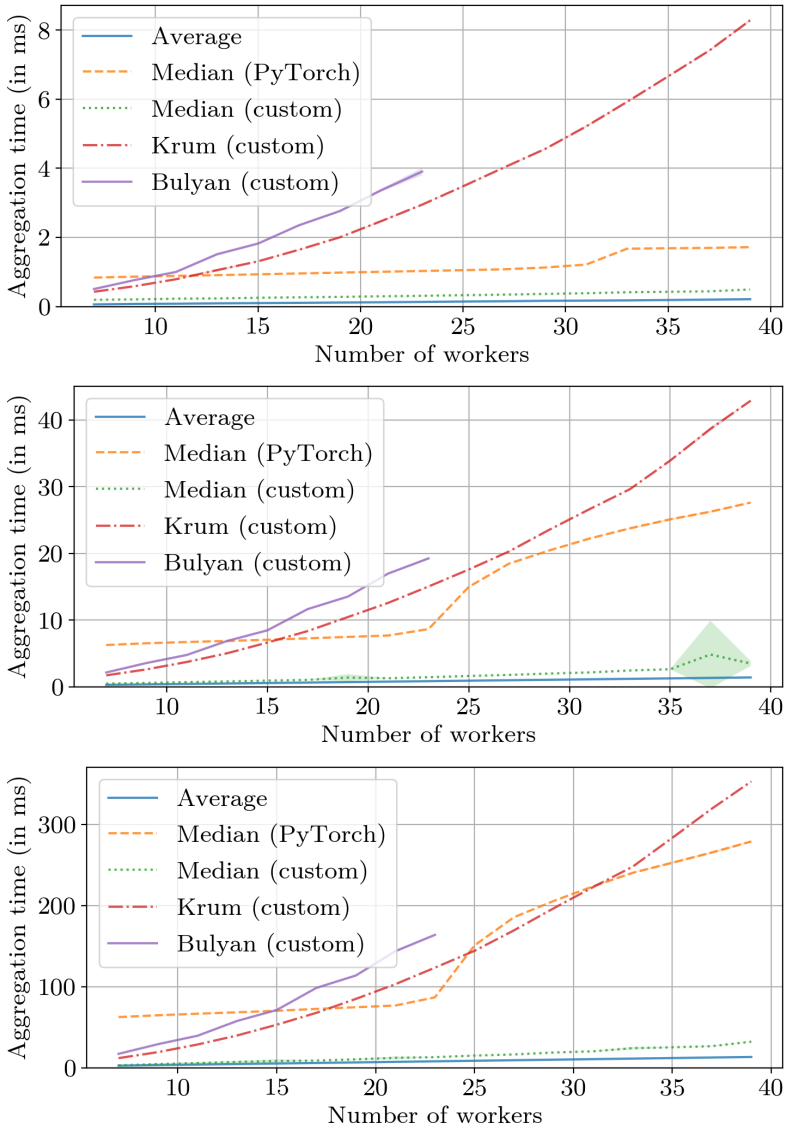


Figure 7.1: Average aggregation time and standard deviation (over 6 runs) function of the number n of aggregated gradients. From top to bottom: $d = 10^5, 10^6, 10^7$.

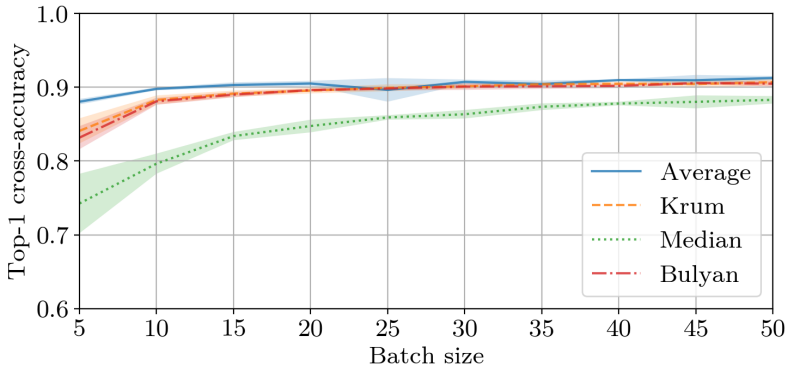


Figure 7.2: Maximum top-1 cross-accuracy reached by the model with a given GAR and gradient batch size. Each experiment is repeated 5 times, with seeds 1 to 5 for reproducibility purpose, and we report on the average and standard deviation of the measured maximum accuracies.

Part IV
Summary and Future Work

8 More Effective Defenses

Effective defenses has two distinct interpretations: either whether the defenses provide the *desired outcome*, or whether the defenses are *ready for use* in actual deployments.

In this thesis, we explored both interpretations.

8.1 The Curse of Dimensionality

First in Chapter 3, we have seen that stateless aggregation rules based on a *distance-minimization scheme* are vulnerable to the curse of dimensionality. The adversary can use what we called the *leeway of attack* to increase one chosen coordinate on the aggregated gradient, moving the parameter vector to a sub-optimal region of $\mathbb{R}^d$. We observed in our experiments that this attack can have devastating effects. Also, this attack does not (necessarily) violate (α, f)-Byzantine resilience, which theoretical guarantees may not match with intuitive definitions of Byzantine resilience.

As a pragmatic and computationally efficient response to the family of attack we exposed, we proposed *Bulyan*, a new, composite gradient aggregation rule. We discuss a potential improvement in Section 9.1.

8.2 Decentralized Resilience

In Chapter 4, we removed the last single point of failure in standard distributed SGD: the parameter server. We proposed (historically) the first distributed SGD algorithm that does not require a trusted, central parameter server to operate. Our algorithm works with multiple parameter servers, guarantees convergence despite colluding Byzantine workers and servers, and can make progress despite network asynchrony. We also experimentally assessed the slowdown induced by our algorithm, compared to the standard distributed SGD deployment using TensorFlow.

There are still several issues before our algorithm can be trusted in actual deployments. For instance, the initial *expansion phase* (c.f. Section 4.4) may allow the adversary to choose what would be equivalent to the initial parameter vector in standard distributed SGD. A perhaps more important consideration for actual deployments is that the datasets held by the non-Byzantine workers may not be *homogeneous*. Section 9.2 discusses decentralized, *heterogeneous* learning.

8.3 The Impact of the Variance-norm Ratio

In Chapter 5, we faced two powerful attacks targeting statistically-robust, Byzantine resilient GARs. The main identified weakness in this family of GARs is their requirement for a sufficiently low *variance-norm* ratio. Intuitively, the *variance-norm* ratio measures how informative about the real gradient $\nabla Q(\theta_t)$ the non-Byzantine stochastic gradient is. We proposed a pragmatic method which, despite increasing the variance, reduces the *variance-norm* ratio. We studied the impact of our method on the *variance-norm* ratio, and experimentally assessed its practical impact over more than 3680 runs.

We observed a sustained correlation between lower *variance-norm* ratio and higher maximum top-1 cross-accuracy (and lower training losses). This tends to indicate that the *variance-norm* ratio is one important predictive metric regarding statistically-robust Byzantine resilience in practice. Our theoretical analysis suggests that further reducing the *variance-norm* ratio is possible, by dynamically decreasing the momentum factor μ when the *curvature* s_t is high, and increasing it back when s_t is low. If this technique alone is not enough to get the *variance-norm* ratio below its theoretical requirement (e.g. Equation (5.1) and Equation (5.2)), our analysis also ultimately suggests to reduce the learning rate η_t.

While we focused on the synchronous setting (Section 2.1.2), which received in comparison a substantial attention in the Byzantine resilient literature, this work might also be applied in asynchronous settings. Specifically, combining our idea with a filtering scheme such as *Kardam* (Damaskinos et al., 2018) is in principle possible, as this filter and momentum commute. However, further analysis of the interplay between the dynamics of stale gradients and the dynamics of momentum remain necessary.

8.4 Practical, Optimized Byzantine resilience

Part III in this thesis has been devoted to system implementations and experiments.

While the asymptotic complexity of existing Byzantine resilient GARs can be studied on paper, we wanted to uncover their actual costs when deployed on real clusters. In Chapter 6, we developed *AggregaThor*, historically the first distributed, Byzantine

resilient implementation of SGD on TensorFlow. Our implementation substantially differ from "vanilla" deployments, as TensorFlow originally allows any node to *overwrite* data anywhere else in the same cluster. We developed several GARs in C++, parallelized on and optimized for CPUs, and we integrated them to TensorFlow.

With our implementation of *Krum*, 26% of the total runtime (when training academic models and datasets) is spent aggregating gradients. This fraction goes up to 50% with our implementation of *Bulyan*. Basically, running GARs on CPU can hamper throughput (Figure 6.5) and scalability (Figure 6.6, left) with small models; yet with larger models, our CPU implementations arguably scale much better (Figure 6.6, right).

We then wanted to know whether, like for gradient estimations, implementing GARs on GPUs can provide a significant speedup, potentially marginalizing the cost of statistically-robust Byzantine resilience in actual distributed deployments. To that end we ported most of the functionalities of *AggregaThor* to PyTorch, a major machine learning framework. As the programming model of CUDA-compatible GPUs is fairly different than the one of CPUs, GAR implementations for GPUs can substantially differ from the implementations of same GARs on CPUs. We describe a specialized construction block for the *Median*, designed to make the most out of GPUs when $d \gg 1$ and n is small. We benchmarked our GAR implementations against various model sizes d and number of workers n. Our optimized implementations are consistently at least as fast, and often much faster than (compositions of) primitive operations in PyTorch. In particular, our *Median* implementation is much faster than the one provided in PyTorch. Overall in our experiments, we observed that porting and running GARs on GPUs can reduce the aggregation time $\times 15$ over CPUs on average (harmonic mean). This speedup alone could be enough to make Byzantine gradient aggregation take a minor fraction of the total training runtime in actual distributed deployments.

9 Future Directions

9.1 Model-aware Aggregation Rules

Not all model parameters have an equal impact on the behavior of the model. For instance in a feed-forward, fully-connected neural network, even the neural weights at the same layer do not all equally impact the result of the forward pass; they have different *roles*. Weights can belong to neurons which output is scaled down in most subsequent layers, while other weights can be biases at the latest layer and substantially impact the model behavior. The *role* of a single parameter also is a dynamic process, that evolves as the parameter vector gets updated with each training step.

None of the existing Byzantine resilient gradient aggregation rules consider the impact each coordinate has on the model. The implicit assumption that all parameters are equivalent is instrumental in the attack we proposed in Chapter 3, which leaves the adversary utterly free to choose which coordinate to attack with its *leeway* (in particular in our experiments, we attacked a bias in the last layer).

Developing Byzantine resilient GARs which take into account parameter *roles* seems quite challenging though. How to even formalize each coordinate *role*? Also, which complexity would computing this *role* have (e.g. computing a Hessian in $\mathcal{O}(d^2)$ would not be practical)? Perhaps even more importantly, how do the *roles* of each coordinate change over time? This could allow the adversary to attack coordinates just before they become more influential, potentially bypassing defensive strategies.

9.2 Heterogeneous Learning

Our distributed SGD model (Section 2.1.2) assumes *homogeneous* dataset at each worker: each non-Byzantine worker independently samples its datapoints from the same distribution. This assumption can be challenged in practice. In the context

of Federated Learning for instance, data is routinely assumed to be generated at the workers by various persons/organizations (Kairouz et al., 2019), and never shared.

Focusing on the work presented in this thesis, Chapter 4 concluded on several remaining issues, among which the homogeneous dataset assumption. In (El-Mhamdi et al., 2020a), as a direct continuation of Chapter 4, my coauthors and I tackle the heterogeneous problem in the Byzantine, decentralized, asynchronous case. Notably, we study to which extent convergence can be achieved, stumbling upon an impossibility result intuitively resulting from the heterogeneity inducing a *non-nullable variance-norm* ratio among heterogeneously-sampled non-Byzantine gradients.

In the case of Byzantine heterogeneous learning, the challenge we highlight actually regards societal implications. Coupled with statistically-robust[1] gradient aggregation rules, Byzantine heterogeneous learning can restrict minorities with vastly diverging views (expressed by how they built datasets with fairly different distributions). This is due to the very nature of statistically-robust defenses, which work by filtering outliers in favor of the majority. Such an approach had not been an issue before, as the honest participants and their respective datasets had always been assumed homogeneous.

If data privacy is not an issue, resampling (He et al., 2020) could be a pragmatic approach to the problem of Byzantine heterogeneous learning. Data privacy issues could also be mitigated with additional, privacy-preserving techniques.

9.3 Privacy and Byzantine resilience

Another type of threat arises when the non-Byzantine workers are not willing to share with each other and the parameter server(s) their respective training datasets, or merely derived information about their datasets. When there is no Byzantine node, the literature may already provide several appealing techniques (Kairouz et al., 2019).

The case of privacy-preserving, Byzantine resilient SGD is more challenging though. Pillutla et al. (2019) proposes a construction to approximate *Geomed* with a *secure sum* (Bonawitz et al., 2017), which allows to compute and reveal the sum of the gradients without revealing any single one of them[2]. In a nutshell, the approach of Pillutla et al. (2019) consist in iteratively weighting each gradient in a way that would approximate *Geomed*. And for the applications that cannot even leak the parameter vector trajectory/the aggregated gradients, another approach exists: *Differential Privacy* (DP).

In distributed SGD (Section 2.1.2), Differential Privacy (Dwork et al., 2006) would for instance add *noise* to each submitted gradient. DP intuitively works by decorrelating gradients from training samples. Two key properties of DP are *composability* and

[1] It is unclear whether redundancy- and suspicion-based defenses can be applied under heterogeneity.
[2] For illustration, the classical *dining cryptographers* problem is one instance of a secure sum.

robustness to post-processing, which in the context of Byzantine SGD ensure that any GAR can by used and the whole training process will still satisfy differential privacy.

Statistically-robust defenses and differential privacy nevertheless remain hard to compose. Indeed: when $d \gg 1$, the privacy noise can be quite large, which may substantially increase the *variance-norm* ratio of the submitted gradients. My coauthors and I studied the intrinsic antagonism between DP and statistically-robust defenses, published in (Guerraoui et al., 2021b), highlighting how combining DP with such defenses becomes practically impossible with large models. In a nutshell, when d is too large, it may[3] become implausible to ever satisfy the *variance-norm* ratio condition, which is necessary to statistically-robust GARs. We also experimentally explore how difficult training even small ($d < 100$) models with both DP and Byzantine resilience can be.

Future work may try to mitigate these issues, or find different paths altogether.

[3]Depending on the *sensitivity* of the the non-Byzantine gradient estimations.

A Additional Experimental Results

A.1 Distributed Momentum

A.1.1 Reproducing the results

Our contributed code is available at `https://github.com/LPD-EPFL/ByzantineMomentum`, or as a ZIP archive from OpenReview (`https://openreview.net/forum?id=H8UHdhWG6A3`).

Software dependencies.

Python 3.7.3 has been used, over several GNU/Linux distributions (Debian 10, Ubuntu 18). Besides the standard libraries associated with Python, our scripts also depend on[1]:

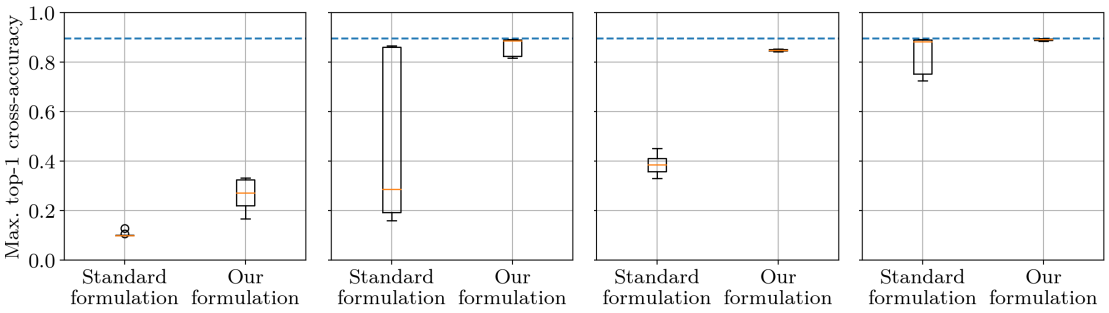
Library	Version	Library	Version	Library	Version
numpy	1.19.1	requests	2.21.0	pytz	2020.1
torch	1.6.0	urllib3	1.24.1	dateutil	2.8.1
torchvision	0.7.0	chardet	3.0.4	pyparsing	2.2.0
pandas	1.1.0	certifi	2018.08.24	cycler	0.10.0
matplotlib	3.0.2	idna	2.6	kiwisolver	1.0.1
PIL	7.2.0	six	1.15.0	cffi	1.13.2

Hardware dependencies.

We list below the hardware components used:

- 1 Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz
- 2 Nvidia GeForce GTX 1080 Ti
- 64 GB of RAM

[1] This list was automatically generated (see `get_loaded_dependencies()` in `tools/misc.py`).



[a] [b] [c] [d]

Figure A.1: CIFAR-10 and *wide-resnet* model. [a] Roughly an half ($f = 4$) Byzantine workers implementing (Baruch et al., 2019). [b] Roughly a quarter ($f = 2$) Byzantine workers implementing (Baruch et al., 2019). [c] Roughly an half Byzantine workers implementing (Xie et al., 2019a). [d] Roughly a quarter Byzantine workers implementing (Xie et al., 2019a).

Command.

Our results are reproducible in one command. In the root directory of the ZIP file:

```
$ python3 reproduce.py
```

On our hardware, reproducing the results takes a bit less than a week. Please be aware this script requires non-negligible disk space: 2.1 GB of run data, and 132 MB of graphs.

Instructing the script to launch several runs per available GPU may reduce the total runtime with some hardware. For instance, to push up to 4 concurrent runs per GPU:

```
$ python3 reproduce.py -supercharge 4
```

A.1.2 Larger models

To assess our method on even larger models, we consider the "wide-resnet" model family implemented by Kim (2020). We use the same model-specific parameters as the ones proposed by the original author, namely: 28 (depth), 10 (widen factor), 0.3 (dropout rate), and 10 output classes (for CIFAR-10). This model contains $36\,489\,290$ trainable parameters, almost 28 times more than the $1\,310\,922$ trainable parameters of the *convolutional* model.

We employ the same hyperparameters as in our main experiments with the *convolutional* model (Section 5.4.1), except for the number of workers (set to $n = 11$), the mini-batch size per worker (set to 20), and the learning rate schedule (0.02 for $t < 8000$, 0.004 for $8000 \leq t < 16000$, 0.0008 for $t \geq 16000$).

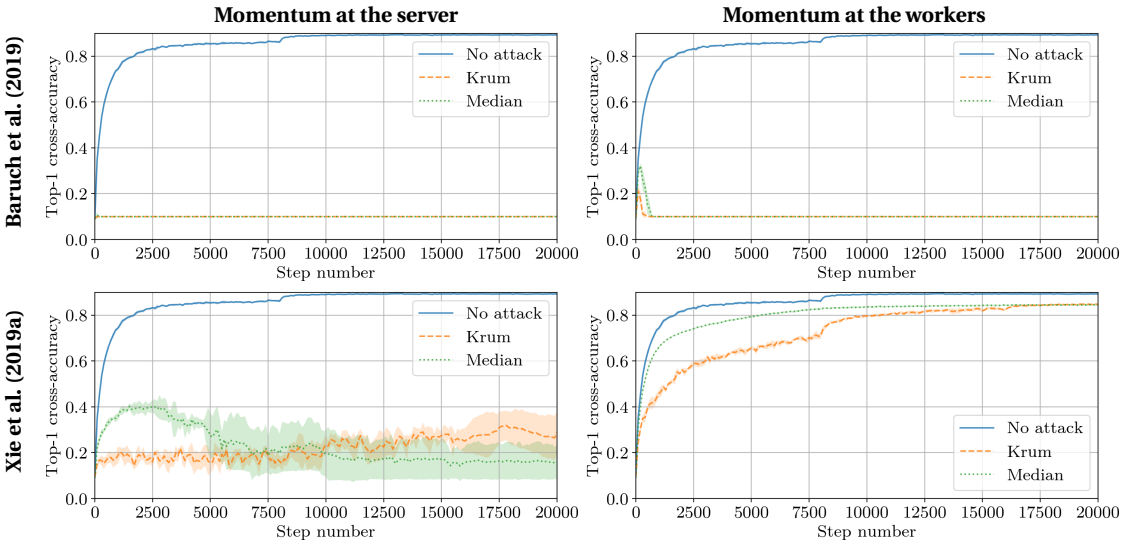


Figure A.2: CIFAR-10 and *wide-resnet* model, roughly an half of Byzantine workers.

The training procedure lasts for 20 000 steps and only employs Nesterov momentum, as proposed by the original author (Kim, 2020). We report on the maximum observed top-1 cross-accuracy in Figure A.1 and evolution of the top-1 cross-accuracy in figures A.2 and A.3.

These results are also reproducible in one command. In the root directory of the ZIP file, simply execute in a shell:

```
$ python3 reproduce-appendix.py
```

On our hardware, reproducing these results takes several weeks. Some of the 6 presented GARs could not be employed, as they repeatedly trigger *out-of-memory* errors on our GPUs. These GARs have been disabled in this specific script.

A.1.3 More experimental results

This section reports on the entirety of the main experiments, completing Section 5.4 of the main paper. For every pair model-dataset, the following parameters vary:

- Which attack: (Baruch et al., 2019) or (Xie et al., 2019a)
- Which defense: *Krum*, *Median*, *Trimmed Mean*, *Phocas*, *MeaMed*, or *Bulyan*
- How many Byzantine workers (*an half* or *a quarter*)
- Where momentum is computed (*server* or *workers*)
- Which flavor of momentum is used (*classical* or *Nesterov*)
- Which learning rate is used (*larger* or *smaller*)

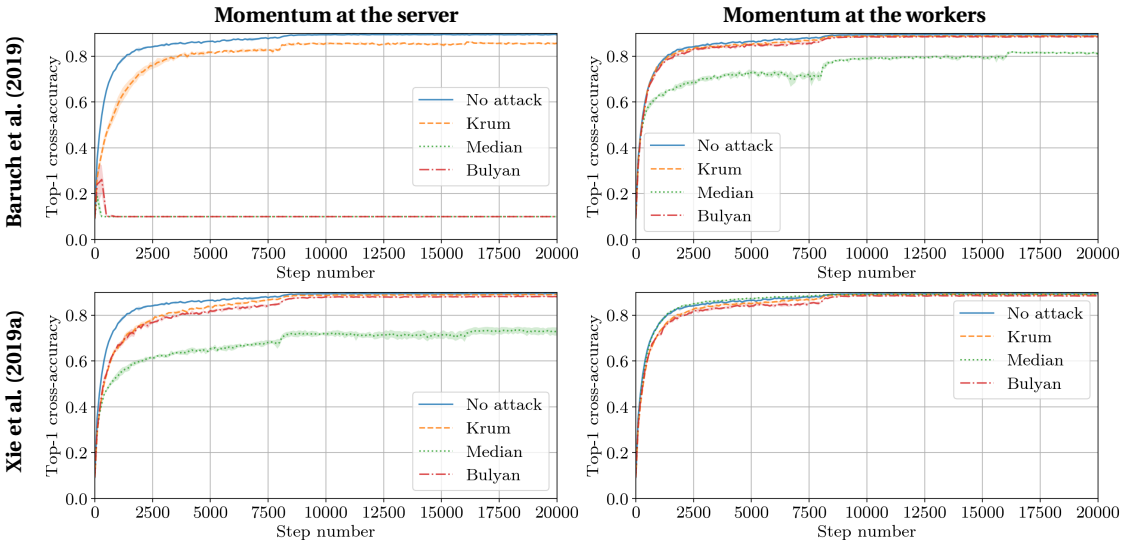


Figure A.3: CIFAR-10 and *wide-resnet* model, roughly a quarter of Byzantine workers.

Every possible combination is tested[2], leading to a total of 736 different experiment setups. Each setup is tested 5 times, each run with a fixed seed from 1 to 5, enabling verbatim reproduction of our results[3]. In this specific section, we report on:

- the maximum observed top-1 cross-accuracy with each of the 6 studied GARs,
- the evolution of the average and standard deviation of the *top-1 cross-accuracy* for every tested setup.

The results regarding the maximum observed top-1 cross-accuracy are layed out by "block" of 4 experiment setups, among which only the flavor of momentum and the attack used are different. Namely: [a] classical momentum under attack from (Baruch et al., 2019), [b] nesterov momentum under attack from (Baruch et al., 2019), [c] classical momentum under attack from (Xie et al., 2019a), [d] nesterov momentum under attack from (Xie et al., 2019a). These results are available from Figure A.4 to Figure A.19.

[2]Along with baselines using *averaging* without attack.

[3]Despite our best efforts, there may still exist minor sources of non-determinism, like race-conditions in the evaluation of certain functions (e.g., parallel additions) in a GPU. Nevertheless we believe these should not affect the results in any significant way.

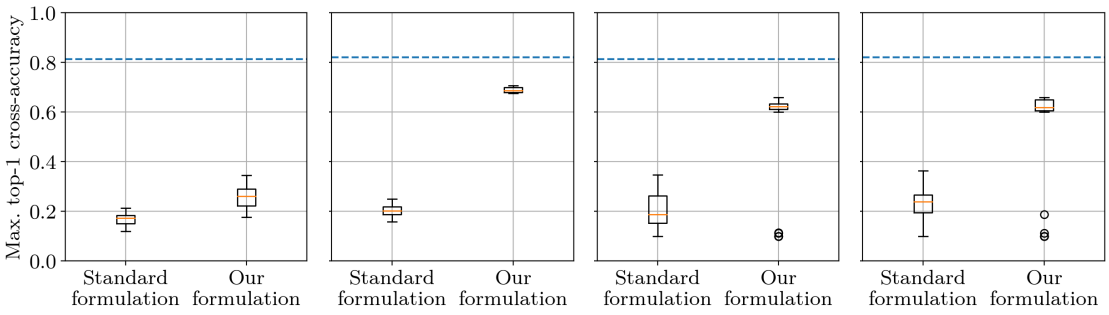


Figure A.4: CIFAR-10 and *convolutional* model, with $n = 25$, $f = 11$ and $\eta_t = 0.01$ if $t < 1500$ else $\eta_t = 0.001$.

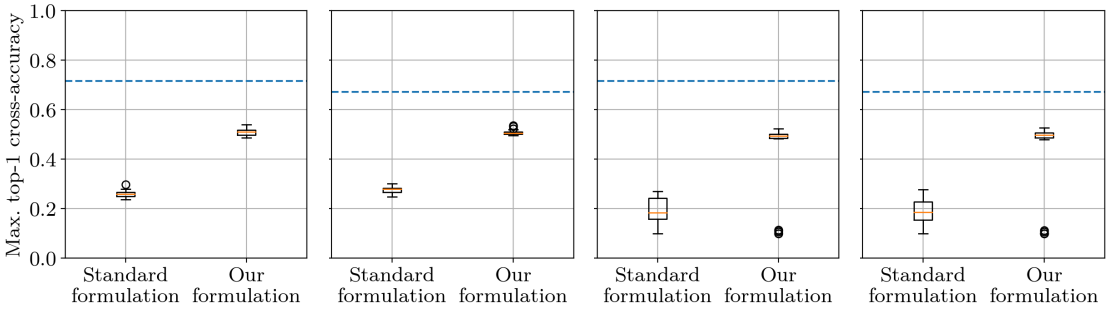


Figure A.5: CIFAR-10 and *convolutional* model, with $n = 25$, $f = 11$ and $\eta_t = 0.001$.

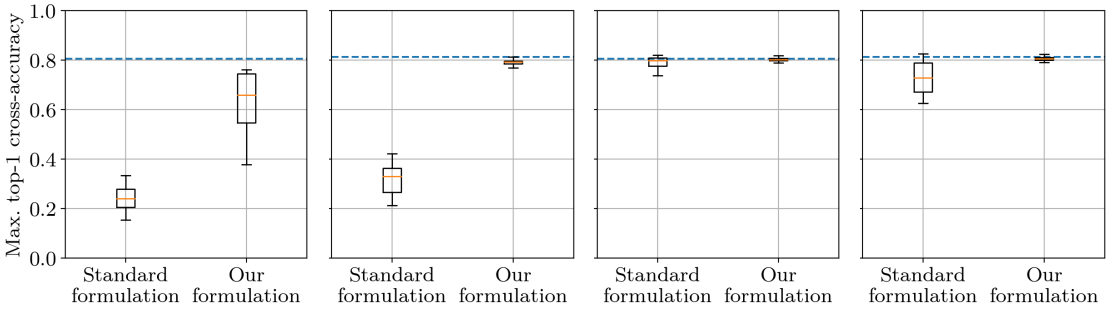


Figure A.6: CIFAR-10 and *convolutional* model, with $n = 25$, $f = 5$ and $\eta_t = 0.01$ if $t < 1500$ else $\eta_t = 0.001$.

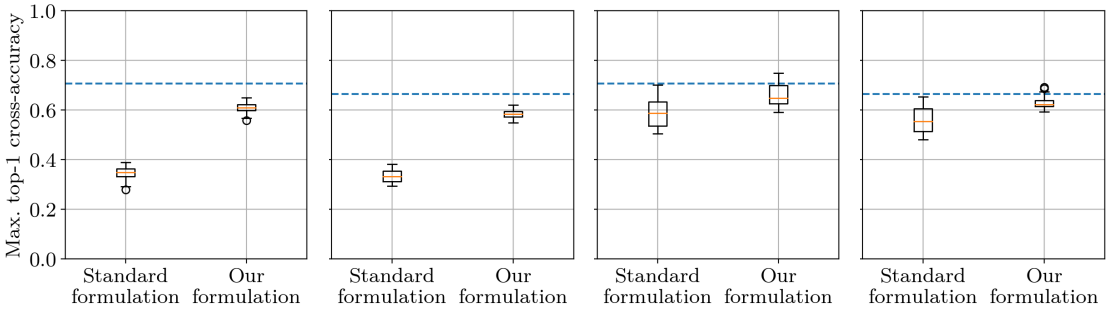


Figure A.7: CIFAR-10 and *convolutional* model, with $n = 25$, $f = 5$ and $\eta_t = 0.001$.

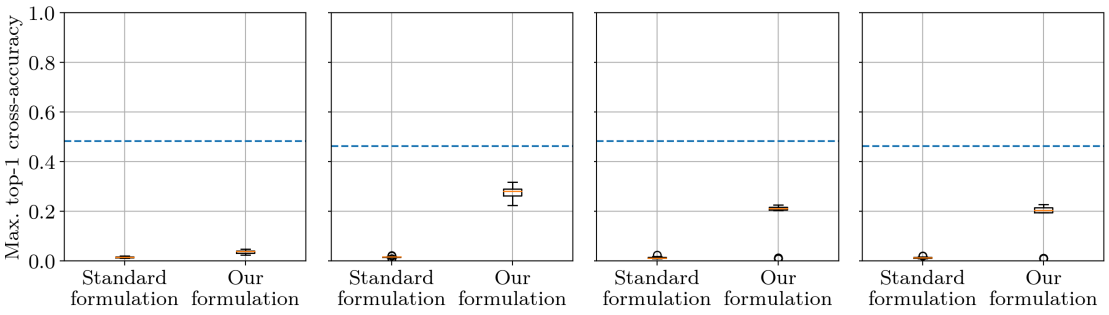


Figure A.8: CIFAR-100 and *convolutional* model, with $n = 25$, $f = 11$ and $\eta_t = 0.01$ if $t < 1500$ else $\eta_t = 0.001$.

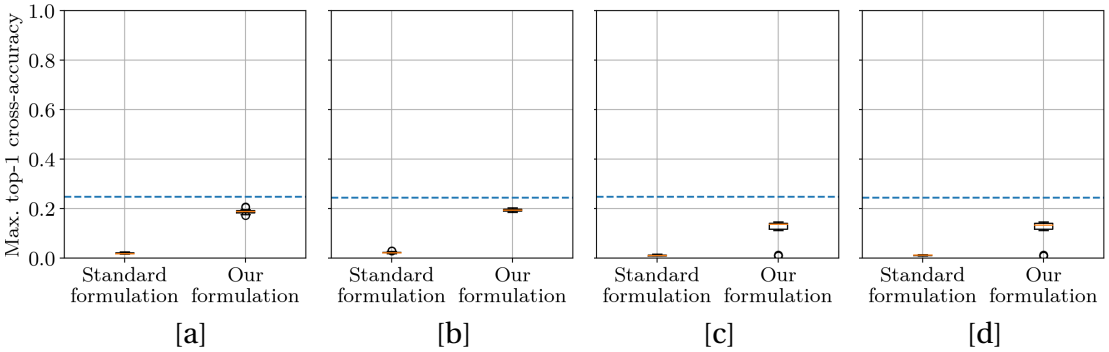


Figure A.9: CIFAR-100 and *convolutional* model, with $n = 25$, $f = 11$ and $\eta_t = 0.001$.

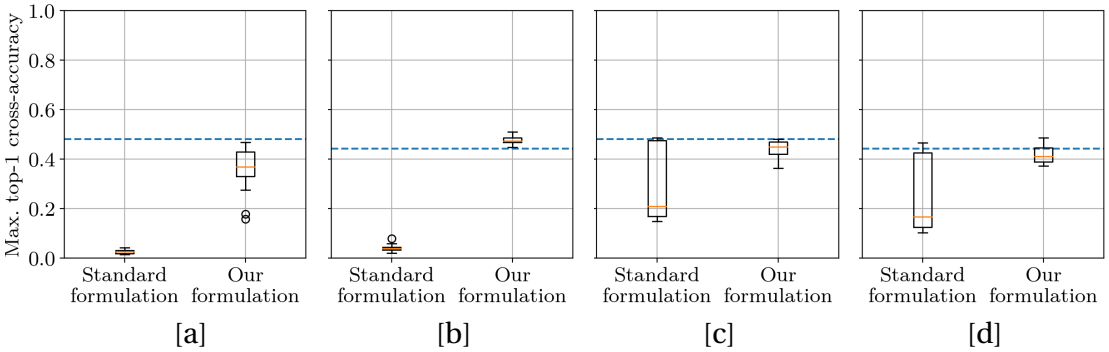


Figure A.10: CIFAR-100 and *convolutional* model, with $n = 25$, $f = 5$ and $\eta_t = 0.01$ if $t < 1500$ else $\eta_t = 0.001$.

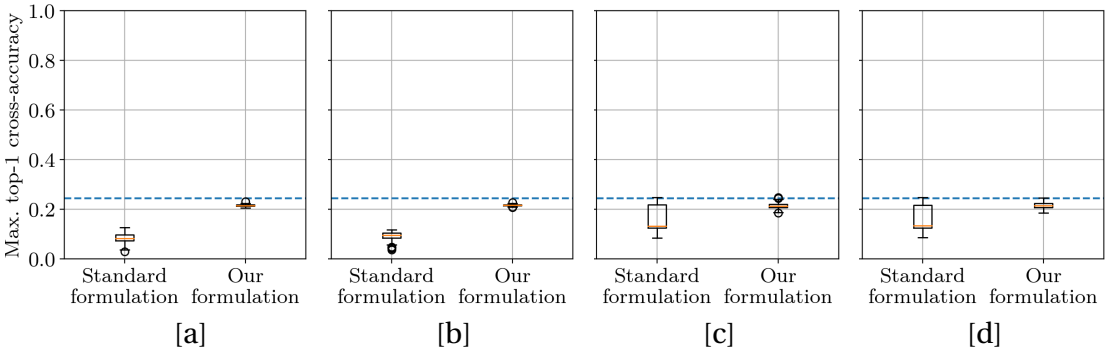


Figure A.11: CIFAR-100 and *convolutional* model, with $n = 25$, $f = 5$ and $\eta_t = 0.001$.

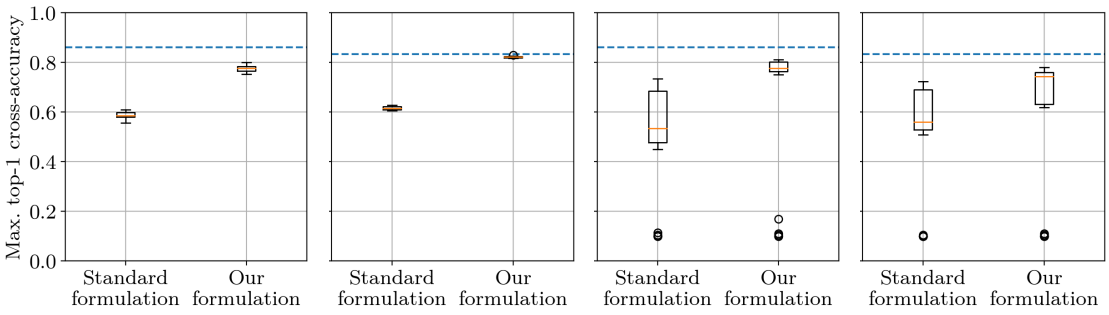


Figure A.12: Fashion MNIST and *fully connected*, with $n = 51$, $f = 24$ and $\eta_t = 0.5$.

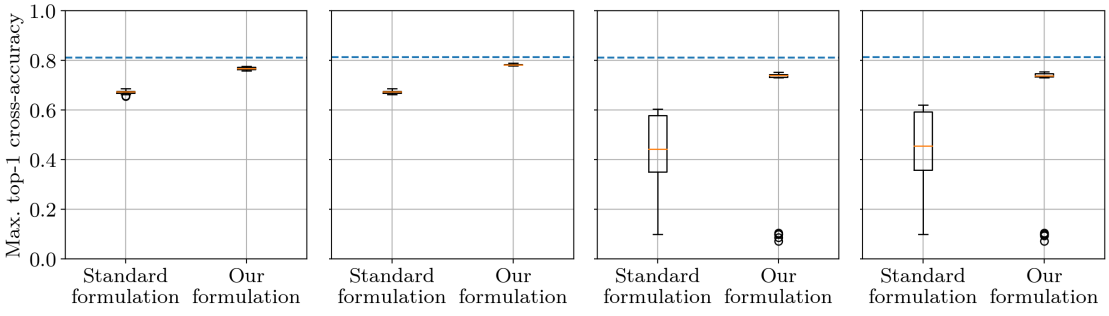


Figure A.13: Fashion MNIST and *fully connected*, with $n = 51$, $f = 24$ and $\eta_t = 0.02$.

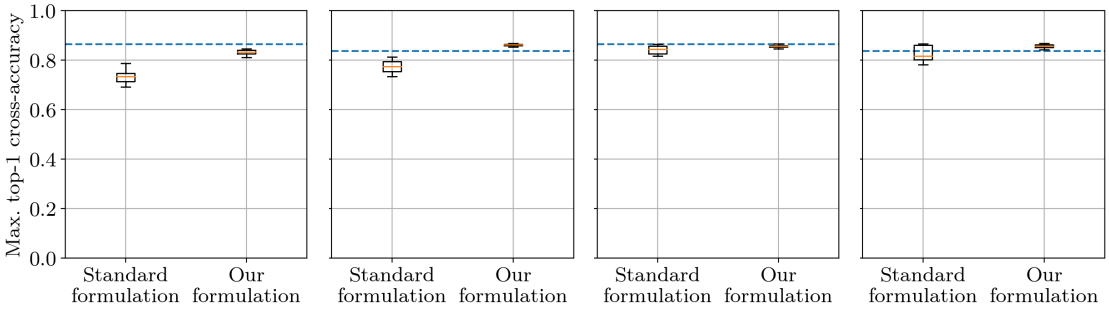


Figure A.14: Fashion MNIST and *fully connected*, with $n = 51$, $f = 12$ and $\eta_t = 0.5$.

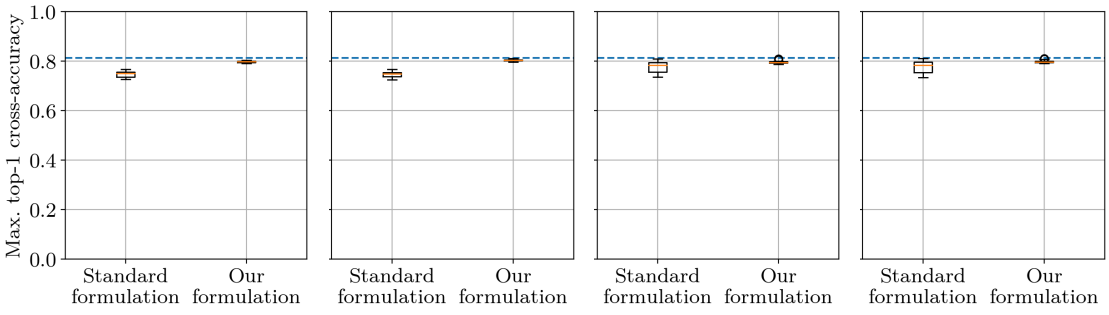


Figure A.15: Fashion MNIST and *fully connected*, with $n = 51$, $f = 12$ and $\eta_t = 0.02$.

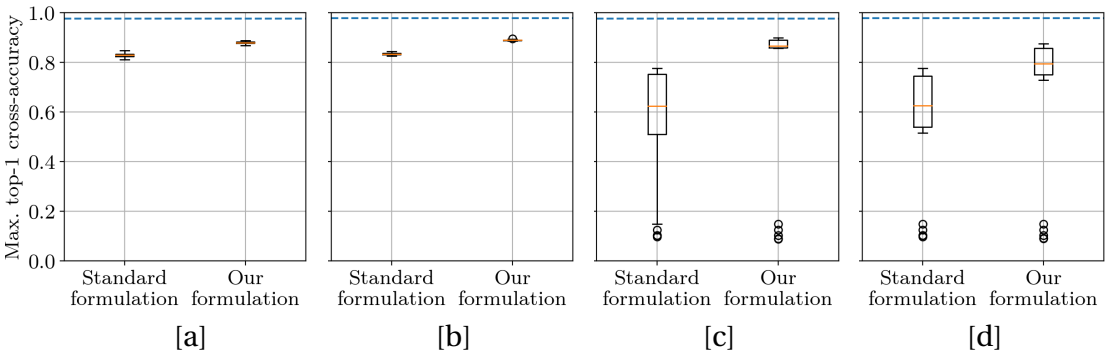


Figure A.16: MNIST and *fully connected* model, with $n = 51$, $f = 24$ and $\eta_t = 0.5$.

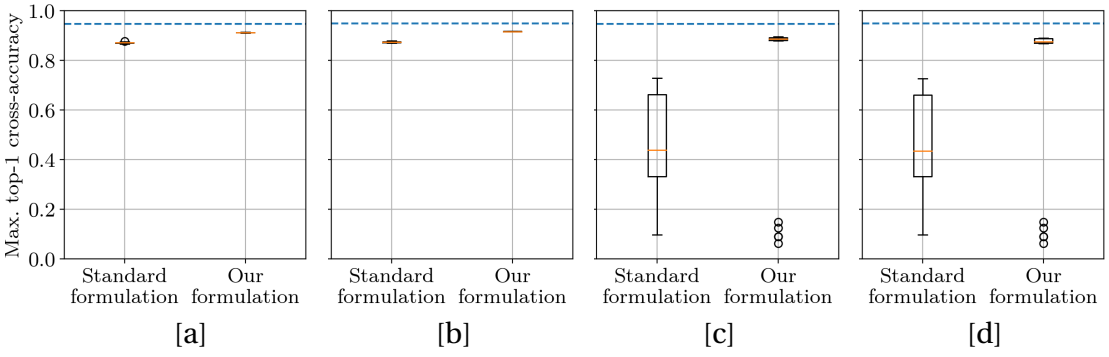


Figure A.17: MNIST and *fully connected* model, with $n = 51$, $f = 24$ and $\eta_t = 0.02$.

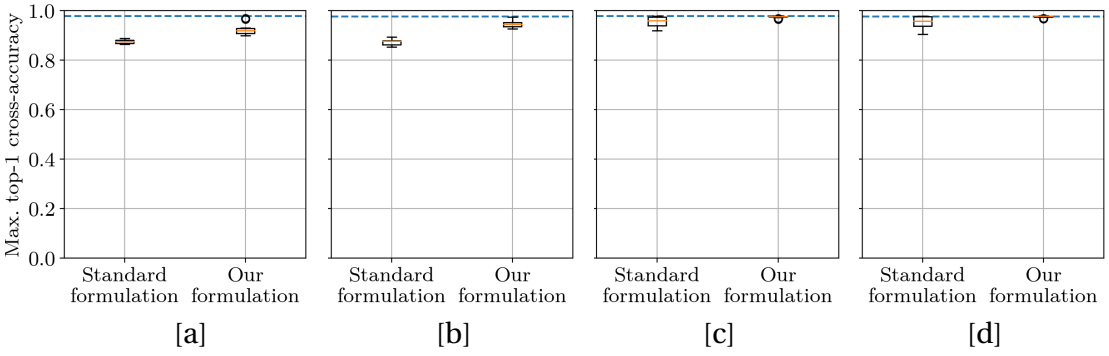


Figure A.18: MNIST and *fully connected* model, with $n = 51$, $f = 12$ and $\eta_t = 0.5$.

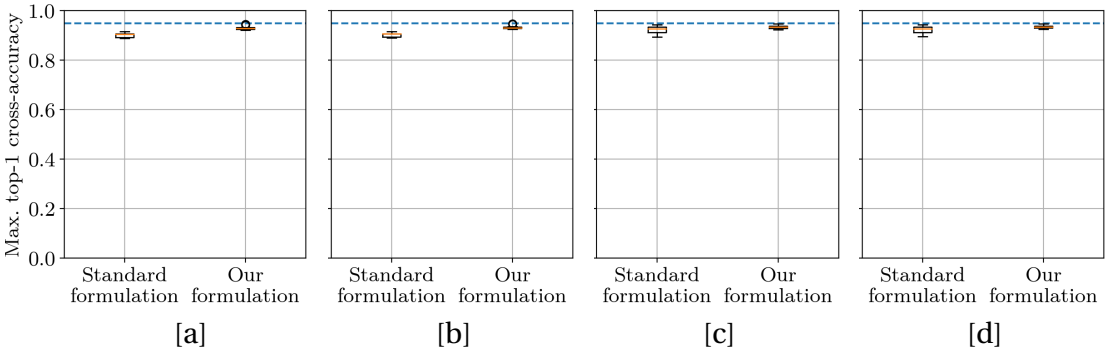


Figure A.19: MNIST and *fully connected* model, with $n = 51$, $f = 12$ and $\eta_t = 0.02$.

B Additional Proofs

B.1 The Hidden Vulnerability: Bulyan's resilience

In Section 3.2, we explained that the *curse of dimensionality* leaves the Byzantine workers, at a coordinate i, with a margin of $\mathcal{O}\left(\sqrt{d}\right)$ computed as the difference between the Byzantine proposed i-th coordinate and the honest proposed vectors' i-th coordinates. In what follows, we prove that any vector produced by *Bulyan* is constrained, in each coordinate, to remain within $\mathcal{O}\left(\frac{\bar{\sigma}}{\sqrt{d}}\right)$ of the honest workers, with: $\bar{\sigma} \triangleq \mathbb{E}\left\|G_t^{(i)} - G_t^{(j)}\right\|$.

Let F be a (α, f)-Byzantine resilient GAR.

Proposition 1. *Denote by Bu_t the vector chosen by Bulyan of F at round t. Then for any dimension $i \in [1\,..\,d]$ and any honest worker k proposing gradient g_k, we have $\mathbb{E}\,|Bu_t[i] - g_k[i]| = \mathcal{O}\left(\frac{\bar{\sigma}}{\sqrt{d}}\right)$.*

Proof. Let $\xi = (\xi_1, \ldots, \xi_{n-f})$ denote the random $(n-f)$-tuple of training samples used by the honest workers. By assumption, the ξ_k (for $k \in [1\,..\,n-f]$ are independent and identically distributed. Let $i \in [1\,..\,d]$ be any coordinate index. We denote by B any vector that is selected by *Bulyan* of F in the set $\mathcal{M}[i]$ (i.e, $B[i]$ scores among the β closest values to $Median[i]$). Let k be any honest worker proposing gradient g_k.

Since B was selected by Bulyan, $B[i]$ is among the closest β propositions to $Median[i]$. We know that $Median[i]$ is the the median coordinate of $\theta \geq 2f+3$ propositions, and we know that $\beta = \theta - 2f$ therefore, all the set $\mathcal{M}[i]$ is closer to $Median[i]$ than at least $2f$ other propositions, in particular, on each side of $Median[i]$ (we are in a single dimension) there are at least f workers who are farther from $Median[i]$ than is any $B[i]$. Therefore, there are at least two different honest workers, call them l and r whose i-th coordinates are respectively on the left and on the right of the $B[i]$, for every B in $\mathcal{M}[i]$, i.e, $g_l[i] \leq B[i] \leq g_r[i]$. There are three cases:

1. $g_k[i] \in]-\infty, g_l[i]]$, then $|B[i] - g_k[i]| < |g_l[i] - g_k[i]|$

2. $g_k[i] \in]g_l[i]\,, g_r[i][$, then $|B[i] - g_k[i]| < |g_l[i] - g_r[i]|$
3. $g_k[i] \in [g_r[i]\,, +\infty[$, then $|B[i] - g_k[i]| < |g_r[i] - g_k[i]|$

Denote by $\mathbb{I}_h$ the indicator function of each of the three cases $h \in \{1, 2, 3\}$ above, i.e. $\mathbb{I}_h = 1$ only if we are in case h, $\mathbb{I}_h = 0$ otherwise. Then we have the following bound:

$$|B[i] - g_k[i]| < \mathbb{I}_1 \ |g_l[i] - g_k[i]| + \mathbb{I}_2 \ |g_l[i] - g_r[i]| + \mathbb{I}_3 \ |g_r[i] - g_k[i]|$$

Let $B_1, \cdots, B_\beta$ be the β elements of $\mathcal{M}[i]$, the previous inequality holds for every B_h, denote by $\mathbb{I}_{r,h},\ r \in [1\,..\,3]$ the corresponding indicator functions for each h, we have:

$$\begin{aligned}
|Bu_t[i] - g_k[i]| &\leq \frac{1}{\beta} \sum_{h=1}^{\beta} |B_h[i] - g_k[i]\,| \\
&\leq \frac{1}{\beta} \sum_{h=1}^{\beta} (\mathbb{I}_{1,h} \ |g_l[i] - g_k[i]| + \mathbb{I}_{2,h} \ |g_l[i] - g_r[i]| + \mathbb{I}_{3,h} \ |g_r[i] - g_k[i]|)
\end{aligned}$$

Since g_l, g_r and g_r are all honest workers, which in addition are positioned w.r.t. to other honest workers, they are i.i.d random variables following the randomness of ξ and satisfy a vector-wise variance bound (norm 2) $\mathbb{E}\,|g_r - g_l| = \mathbb{E}\,|g_k - g_l| = \mathbb{E}\,|g_k - g_r| \leq \mathbb{E}\,|g_k - G| + \mathbb{E}\,|G - g_r| = \mathcal{O}(\bar{\sigma})$, where G is the unbiased estimator used by the honest workers with a bounded variance such that, component-wise (we divide by $\sqrt{d}$): $\mathbb{E}\,|Bu_t[i] - g_k[i]| = \mathcal{O}\Big(\frac{\bar{\sigma}}{\sqrt{d}}\Big)$. □

Proposition 1 proves that *Bulyan* of F reduces the component-wise margin of an attacker, i.e. how much the latter can deviate from honest workers component-wise, while still be influencing the aggregated gradient.

A last natural question to be posed is: will *Bulyan* of F introduce an additional bias in gradient estimations? The answer, provided by Corollary 1, is *no*. We show that *Bulyan* of F keeps the gradient estimation in the cone of angle α around the true gradient. In particular, *Bulyan* of F is also provably convergent.

Corollary 1. *As F is assumed to be (α, f)-Byzantine resilient, Bulyan of F is also (α, f)-Byzantine resilient.*

Proof. This is an immediate consequence of the (α, f)-Byzantine resilience of F (Definition 1) and of the fact that any vector used as an input to the last (averaging) step of Bulyan already comes from the cone of angle α, since it was selective by an iteration of F on a set of vectors of cardinal $\geq 2\,f + 2$. Let g be the true gradient, a triangle inequality applied between g, Bu and the β terms coming from the iterations of F, call them $F_k,\ k \in [1\,..\,\beta]$ gives: $|Bu - g| \leq \frac{1}{\beta}\,|F_k - g|$. Given how the iterations over F are

performed (without re-sampling ξ), the F_k are themselves i.i.d. and by taking the $\mathbb{E}$ on the inequality, every term in the sum of the right-hand side is bounded by $|g| \cdot \sin(\alpha)$ (since it lives in the cone of angle α around g). Therefore: $|\mathbb{E}\, Bu - g| \leq |g| \cdot \sin(\alpha)$ which means that $\mathbb{E}Bu$ is also a vector in the cone of angle α around g. The proof on the statistical moments is obtained with same steps above (except of bounding with $\mathbb{E}\,|G|^r$'s instead of $\sin(\alpha) \cdot |g|$ □

Finally, even if the focus of our work was rather on narrowing the leeway of Byzantine workers which we argue is a more powerful requirement than (α, f)-Byzantine resilient alone. It is worth mentioning that as a consequence of our results, convergence is ensured for Bulyan.

Corollary 2 (Convergence)**.** *With Bulyan of F, the sequence of models x_t adopted by the master almost surely converges to a region where $\nabla Q(x) = 0$*

Proof. As a consequence of Corollary 1, Bulyan is also (α, f)-Byzantine resilient, by Proposition 2 of (Blanchard et al., 2017) guarantees almost sure convergence. □

Bibliography

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *OSDI*.

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

Alistarh, D., Allen-Zhu, Z., and Li, J. (2018). Byzantine stochastic gradient descent. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pages 4618–4628.

Alistarh, D., Li, J., Tomioka, R., and Vojnovic, M. (2016). Qsgd: Randomized quantization for communication-optimal stochastic gradient descent. *arXiv preprint arXiv:1610.02132*.

Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485.

Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., and Shmatikov, V. (2020). How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics*, pages 2938–2948. PMLR.

Baruch, M., Baruch, G., and 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*.

Belkin, M., Hsu, D., Ma, S., and Mandal, S. (2019). Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854.

Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., and Roli, F. (2013). Evasion attacks against machine learning at test time. In *Joint*

European conference on machine learning and knowledge discovery in databases, pages 387–402. Springer.

Biggio, B., Nelson, B., and Laskov, P. (2012). Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389*.

Blanchard, P., El-Mhamdi, E.-M., Guerraoui, R., and 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, 4-9 December 2017, Long Beach, CA, USA*, pages 119–129.

Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., Ramage, D., Segal, A., and Seth, K. (2017). Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191.

Bottou, L. (1998). Online Learning and Stochastic Approximations. *On-line learning in neural networks*, 17(9):142.

Bottou, L. (2012). *Stochastic Gradient Descent Tricks*, pages 421–436. Springer Berlin Heidelberg, Berlin, Heidelberg.

Bousquet, O. and Bottou, L. (2008). The tradeoffs of large scale learning. In *Neural Information Processing Systems*, pages 161–168.

Boussetta, A., El-Mhamdi, E.-M., Guerraoui, R., Maurer, A., and Rouault, S. (2021). Aksel: Fast byzantine sgd. In *24th International Conference on Principles of Distributed Systems (OPODIS 2020)*.

Cachin, C., Guerraoui, R., and Rodrigues, L. (2011). *Introduction to reliable and secure distributed programming*. Springer Science & Business Media.

Carlini, N., Tramer, F., Wallace, E., Jagielski, M., Herbert-Voss, A., Lee, K., Roberts, A., Brown, T., Song, D., Erlingsson, U., et al. (2020). Extracting training data from large language models. *arXiv preprint arXiv:2012.07805*.

Charikar, M., Steinhardt, J., and Valiant, G. (2017). Learning from untrusted data. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 47–60.

Chen, L., Wang, H., Charles, Z. B., and Papailiopoulos, D. S. (2018). DRACO: byzantine-resilient distributed training via redundant gradients. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 902–911.

Chen, L., Ye, Y., and Bourlai, T. (2017a). Adversarial machine learning in malware detection: Arms race between evasion attack and defense. In *2017 European Intelligence and Security Informatics Conference (EISIC)*, pages 99–106. IEEE.

Chen, Y., Su, L., and Xu, J. (2017b). Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. *CoRR*, abs/1705.05491.

Chilimbi, T. M., Suzue, Y., Apacible, J., and Kalyanaraman, K. (2014). Project adam: Building an efficient and scalable deep learning training system. In *OSDI*, volume 14, pages 571–582.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

Damaskinos, G. (2020). *Private and Secure Distributed Learning*. PhD thesis, École Polytechnique Fédérale de Lausanne.

Damaskinos, G., El-Mhamdi, E.-M., Guerraoui, R., Guirguis, A. H. A., and Rouault, S. (2019). Aggregathor: Byzantine machine learning via robust gradient aggregation. In *The Conference on Systems and Machine Learning (SysML), 2019*.

Damaskinos, G., El-Mhamdi, E.-M., Guerraoui, R., Patra, R., and Taziki, M. (2018). Asynchronous byzantine machine learning (the case of SGD). In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 1153–1162.

Dwork, C., McSherry, F., Nissim, K., and Smith, A. (2006). Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer.

El-Mhamdi, E., Guerraoui, R., Guirguis, A., Hoang, L. N., and Rouault, S. (2020). Genuinely distributed byzantine machine learning. In Emek, Y. and Cachin, C., editors, *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*, pages 355–364. ACM.

El-Mhamdi, E.-M. (2020). *Robust Distributed Learning*. PhD thesis, École Polytechnique Fédérale de Lausanne.

El-Mhamdi, E.-M., Farhadkhani, S., Guerraoui, R., Guirguis, A., Hoang, L. N., and Rouault, S. (2020a). Collaborative learning as an agreement problem. *arXiv preprint arXiv:2008.00742*.

El-Mhamdi, E.-M., Guerraoui, R., and Rouault, S. (2017). On the robustness of a neural network. In *2017 International Symposium on Reliable Distributed Systems (SRDS)*.

El-Mhamdi, E.-M., Guerraoui, R., and Rouault, S. (2018). The hidden vulnerability of distributed learning in byzantium. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 3518–3527.

El-Mhamdi, E.-M., Guerraoui, R., and Rouault, S. (2020b). Distributed momentum for byzantine-resilient stochastic gradient descent. In *International Conference on Learning Representations*.

El-Mhamdi, E.-M., Guerraoui, R., and Rouault, S. (2020c). Fast and robust distributed learning in high dimension. In *2020 International Symposium on Reliable Distributed Systems (SRDS)*, pages 71–80. IEEE.

Fischer, M. J., Lynch, N. A., and Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *JACM*, 32(2):374–382.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

Guerraoui, R., Guirguis, A., Plassmann, J. M., Ragot, A. A., and Rouault, S. (2021a). Garfield: System support for byzantine machine learning. In *51st IEEE/IFIP International Conference on Dependable Systems and Networks*.

Guerraoui, R., Gupta, N., Pinot, R., Rouault, S., and Stephan, J. (2021b). Differential privacy and byzantine resilience in sgd: Do they add up? In *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*.

He, L., Karimireddy, S. P., and Jaggi, M. (2020). Byzantine-robust learning on heterogeneous datasets via resampling. *arXiv preprint arXiv:2006.09365*.

Hecht-Nielsen, R. (1992). Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier.

Hsieh, K., Harlap, A., Vijaykumar, N., et al. (2017). Gaia: Geo-distributed machine learning approaching lan speeds. In *NSDI*, pages 629–647.

Kachelrieß, M. (2009). Branchless vectorized median filtering. In *2009 IEEE Nuclear Science Symposium Conference Record (NSS/MIC)*, pages 4099–4105. IEEE.

Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., et al. (2019). Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*.

Karimireddy, S. P., He, L., and Jaggi, M. (2020). Learning from history for byzantine robust optimization. *arXiv preprint arXiv:2012.10333*.

Kim, B. (2020). Best cifar-10, cifar-100 results with wide-residual networks using pytorch. https://github.com/meliketoy/wide-resnet.pytorch. MIT license, using commit 292b3ede0651e349dd566f9c23408aa572f1bd92.

Kim, L. (2012). How many ads does google serve in a day?

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105.

Lamport, L., Shostak, R., and Pease, M. (1982). The Byzantine generals problem. *TOPLAS*, 4(3):382–401.

Li, L., Xu, W., Chen, T., Giannakis, G. B., and Ling, Q. (2019). Rsa: Byzantine-robust stochastic aggregation methods for distributed learning from heterogeneous datasets. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1544–1551.

Li, M., Zhou, L., Yang, Z., et al. (2013). Parameter server for distributed machine learning. In *Big Learning NIPS Workshop*, volume 6, page 2.

Liu, K. (2019). Train cifar-10 with pytorch.

Liu, S. (2021). A survey on fault-tolerance in distributed optimization and machine learning. *arXiv preprint arXiv:2106.08545*.

Liu, S., Gupta, N., and Vaidya, N. H. (2021). Approximate byzantine fault-tolerance in distributed optimization. *arXiv preprint arXiv:2101.09337*.

Luo, L., Nelson, J., Ceze, L., Phanishayee, A., and Krishnamurthy, A. (2018). Parameter box: High performance parameter servers for efficient distributed deep neural network training. *CoRR*, abs/1801.09805.

Moosavi-Dezfooli, S.-M., Fawzi, A., Fawzi, O., and Frossard, P. (2017). Universal adversarial perturbations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Musser, D. R. (1997). Introspective sorting and selection algorithms. *Software: Practice and Experience*, 27(8):983–993.

Nesterov, Y. (1983). A method for solving a convex programming problem with convergence rate o(1/k2). *Soviet Mathematics Doklady*, 27:372–367.

Pillutla, K., Kakade, S. M., and Harchaoui, Z. (2019). Robust aggregation for federated learning. *arXiv preprint arXiv:1912.13445*.

Polyak, B. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4:1–17.

PyTorch contributors (2016). Pytorch. https://pytorch.org/.

Rajput, S., Wang, H., Charles, Z., and Papailiopoulos, D. (2019). Detox: A redundancy-based framework for faster and more robust gradient aggregation. *Neural Information Processing Systems*.

Roelofs, R., Shankar, V., Recht, B., Fridovich-Keil, S., Hardt, M., Miller, J., and Schmidt, L. (2019). A meta-analysis of overfitting in machine learning. *Advances in Neural Information Processing Systems*, 32:9179–9189.

Rousseeuw, P. J. (1985). Multivariate estimation with high breakdown point. *Mathematical statistics and applications*, 8:283–297.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, 61:85–117.

Schneider, F. B. (1990). Implementing fault-tolerant services using the state machine approach: A tutorial. *CSUR*, 22(4):299–319.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 15(1):1929–1958.

Su, L. (2017). *Defending distributed systems against adversarial attacks: consensus, consensus-based learning, and statistical learning*. PhD thesis, University of Illinois at Urbana-Champaign.

Sun, Z., Kairouz, P., Suresh, A. T., and McMahan, H. B. (2019). Can you really backdoor federated learning? *arXiv preprint arXiv:1911.07963*.

TensorFlow contributors (2015). Tensorflow. https://www.tensorflow.org/.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5–rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning.

Wang, H., Sreenivasan, K., Rajput, S., Vishwakarma, H., Agarwal, S., Sohn, J.-y., Lee, K., and Papailiopoulos, D. (2020). Attack of the tails: Yes, you really can backdoor federated learning. *arXiv preprint arXiv:2007.05084*.

Xiao, H., Biggio, B., Brown, G., Fumera, G., Eckert, C., and Roli, F. (2015). Is feature selection secure against training data poisoning? In *ICML*, pages 1689–1698.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747.

Xie, C. (2019). Zeno++: robust asynchronous SGD with arbitrary number of byzantine workers. *CoRR*, abs/1903.07020.

Xie, C., Koyejo, O., and Gupta, I. (2018a). Generalized byzantine-tolerant SGD. *CoRR*, abs/1802.10116.

Xie, C., Koyejo, O., and Gupta, I. (2018b). Generalized Byzantine-tolerant sgd. *arXiv preprint arXiv:1802.10116*.

Xie, C., Koyejo, O., and Gupta, I. (2018c). Phocas: dimensional byzantine-resilient stochastic gradient descent. *CoRR*, abs/1805.09682.

Xie, C., Koyejo, O., and Gupta, I. (2019a). 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*, page 83.

Xie, C., Koyejo, S., and Gupta, I. (2019b). Zeno: Distributed stochastic gradient descent with suspicion-based fault-tolerance. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 6893–6901.

Yamada, Y., Iwamura, M., Akiba, T., and Kise, K. (2019). Shakedrop regularization for deep residual learning. *IEEE Access*, 7:186126–186136.

Yin, D., Chen, Y., Kannan, R., and Bartlett, P. (2018). Byzantine-robust distributed learning: Towards optimal statistical rates. In *International Conference on Machine Learning*, pages 5650–5659. PMLR.

Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. *arXiv preprint arXiv:1605.07146*.

Zhang, H., Zheng, Z., Xu, S., Dai, W., Ho, Q., Liang, X., Hu, Z., Wei, J., Xie, P., and Xing, E. P. (2017). Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters. In *USENIX ATC*, pages 181–193.

Zhang, S., Choromanska, A. E., and LeCun, Y. (2015). Deep learning with elastic averaging sgd. In *NIPS*, pages 685–693.