Thèse n° 7218

Robust Distributed Learning

Présentée le 6 mars 2020

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

pour l'obtention du grade de Docteur ès Sciences

par

El Mahdi EL MHAMDI

Acceptée sur proposition du jury

Prof. B. Falsafi, président du jury
Prof. R. Guerraoui, directeur de thèse
Prof. M. Herlihy, rapporteur
Prof. F. Bach, rapporteur
Prof. M. Jaggi, rapporteur

École
polytechnique
fédérale
de Lausanne

2020

" لبحَر منين كاينبع و لارض اللّي كوّرها شكون "

المعطي بن يزّة و المُختار

"If a machine is expected to be infallible, it cannot also be intelligent."

Alan Mathison Turing

Acknowledgements

The many years that preceded and enabled this short 4-years journey would not have been what they were if it was not for my parents, my sister and my two brothers who shaped my personality and defined who I am. This work is dedicated to them.

This work would not have been possible without the collaborations I was allowed to start in the distributed computing laboratory with other members of the lab, some of which were true believers who abandoned their initial research plans to join the effort on robust learning. The use of *we* instead of *I* is meant to reflect that.

This thesis would not have happened if the randomness of life didn't put Rachid Guerraoui on my path when I started an online science tutoring project called *Wandida*. During the Wandida period, creating university-level computer science courses with Rachid made me discover how many foundational questions one could find in computing and, during the PhD period, Rachid gave me a level of freedom, autonomy and trust I could not have found elsewhere.

This thesis would also not have happened if that same randomness didn't put my fiancée Mariame on my path. I thank her for her support when I needed to decide whether I "go back to school" for a PhD, for her crucial presence when I was still on a blank page a few days away from my PhD deadline and a book to finish in the same time, and for every other key moment in the four years in between. My satisfaction in finishing this thesis is amplified by the joy of seeing her take the same leap of faith and "go back to school" after an even longer professional drift than mine. I am looking forward to see what will come out of her own PhD.

I thank the members of the jury, Professors Francis Bach and Martin Jaggi, who brought their internationally recognized machine learning expertise and Professor Maurice Herlihy, a true pioneer in distributed systems who established some foundational results that are prior work to this thesis and who also agreed to evaluate this thesis. It was intimidating but also intellectually stimulating and rewarding to defend the thesis in front of them. Many thanks go to Professor Babak Falsafi who presided the jury and provided valuable mentoring prior to the jury formation.

There are much more people who should be thanked for the past four years that I can list, from friends to sports partners to neighbours to colleagues. They will pardon listing only four of them by names: Kristine Verhamme and France Faille, they made the life in the lab easier and reminded us that self-care should always come before work, Zaina Ait Said and Imad Grandi, who left this world too soon for me to thank them.

Finally, I would like to thank the 8 million Swiss taxpayers who funded this research, the 83 million German and 67 million French taxpayers who funded a few years of my academic journey and

most importantly, the 36 million Moroccan taxpayers who funded 15 years of primary, middle school, high school and undergraduate university studies.

Preface

This thesis is part of the work performed from September 2015 to September 2019 in the Distributed Computing Laboratory at EPFL, directed by Prof. Rachid Guerraoui. It focuses on the robustness of distributed learning systems.
The main results presented in this dissertation appeared in the following publications (authors order is alphabetical, except for the one with a * written with biologists).

1. Peva Blanchard; El-Mahdi El-Mhamdi; Rachid Guerraoui; Julien Stainer (2017). Machine learning with adversaries: Byzantine tolerant gradient descent. In Advances in Neural Information Processing Systems **(NeurIPS)** (pp. 119-129).

2. El-Mahdi El-Mhamdi; Rachid Guerraoui and Sébastien Rouault (2018). The Hidden Vulnerability of Distributed Learning in Byzantium. In International Conference on Machine Learning **(ICML)** (pp. 3518-3527). Long talk.

3. Georgios Damaskinos, El-Mahdi El-Mhamdi; Rachid Guerraoui; Rhicheek Patra and Mahsa Taziki (2018). Asynchronous Byzantine Machine Learning (the case of SGD). In International Conference on Machine Learning **(ICML)** (pp. 1153-1162). Long talk.

4. El-Mahdi El-Mhamdi; Rachid Guerraoui (2017). When Neurons Fail. In 2017 IEEE International Parallel and Distributed Processing Symposium **(IPDPS)** (pp. 1028-1037).

5. El-Mahdi El-Mhamdi; Rachid Guerraoui and Sébastien Rouault (2017). On the robustness of a neural network. In 2017 IEEE 36th Symposium on Reliable Distributed Systems **(SRDS)** (pp. 84-93).

During these four years of PhD, my colleagues and I also worked on other aspects of robustness that are not included in this dissertation. Some of these aspects share the biological / fundamental research motivation of the later part of this thesis (6, 7, 8), some are related to the Byzantine resilient gradient aggregation problem introduced in this thesis (9, 10, 11, 12), and others are more general questions in artificial intelligence safety and reliability (13, 14, 15).

6. El-Mahdi El-Mhamdi*; Andrei Kucharavy*, Rachid Guerraoui and Rong Li (2018). Predicting complex genetic phenotypes using error propagation in weighted networks. bioRxiv, 487348.

7. El-Mahdi El-Mhamdi; Rachid Guerraoui; Alexandre Maurer and Vladislav Tempez (2019). Exploring the borderlands of the gathering problem. Bulletin of the European Association of Theoretical Computer Science **(EATCS)**. To appear in October 2019.

8. El-Mahdi El-Mhamdi; Rachid Guerraoui and Sergei Volodin (2019). Fatal Brain Damage. arXiv preprint arXiv:1902.01686.

9. Georgios Damaskinos, El-Mahdi El-Mhamdi; Rachid Guerraoui; Arsany Guirguis and Sébastien Rouault (2019). AGGREGATHOR: Byzantine Machine Learning via Robust Gradient Aggregation. In The Conference on Systems and Machine Learning **(SysML)**.

10. El-Mahdi El-Mhamdi and Rachid Guerraoui (2019). Fast and Secure Distributed Learning in High Dimension. arXiv preprint arXiv:1905.04374.

11. El-Mahdi El-Mhamdi; Rachid Guerraoui; Arsany Guirguis and Sébastien Rouault (2019). SGD: Decentralized Byzantine Resilience. arXiv preprint arXiv:1905.03853.

12. El-Mahdi El-Mhamdi; Rachid Guerraoui and Arsany Guirguis (2019). Fast Byzantine Machine Learning with Unreliable Servers (under submission).

13. El-Mahdi El-Mhamdi; Rachid Guerraoui; Hadrien Hendrikx and Alexandre Maurer (2017). Dynamic safe interruptibility for decentralized multi-agent reinforcement learning. In Advances in Neural Information Processing Systems **(NeurIPS)** (pp. 130-140). Spotlight Talk (given by H.H)

14. Henrik Aslund; El-Mahdi El-Mhamdi; Rachid Guerraoui and Alexandre Maurer (2018). Virtuously Safe Reinforcement Learning. arXiv preprint arXiv:1805.11447.

15. El-Mahdi El-Mhamdi; Rachid Guerraoui; Lê Nguyên Hoang and Alexandre Maurer (2018). Removing Algorithmic Discrimination (With Minimal Individual Error). arXiv preprint arXiv:1806.02510.

Note on experiments: During this PhD, I was fortunate to benefit from the collaborations prof. Rachid Guerraoui allowed us to freely start in his lab, in particular, the work presented in this thesis could not happen without the crucial experimental contribution of my co-authors, Sébastien Rouault and Georgios Damaskinos in particular. While this thesis focuses on the theoretical problem statements, algorithmic solutions and proofs (my contributions), I have included experimental results for didactic illustration. These experiments are mostly from my co-authors, except the ones in Chapter 3 and the corresponding appendix (Krum, Multi-Krum, (1-p)-Krum and the Medoid) which were mine.

Abstract

Whether it occurs in artificial or biological substrates, *learning* is a distributed phenomenon in at least two aspects. First, meaningful data and experiences are rarely found in one location, hence *learners* have a strong incentive to work together. Second, a learner is itself a distributed system, made of more basic processes; the change in the connections between these basic processes is what allows learning. This generic view encompasses a large set of learning situations, from brains, to metabolic networks in the organism to the data centers where several machines are collaborating to recommend personalized content for a billion-users social media.

In both aforementioned aspects, a system's ability to cope with the failure of some of its components is crucial. This thesis explores the robustness of learning systems from these two aspects. The first is *coarse-grained*, as the unit of failure is a whole learner. The second is *fine-grained*, as the unit of failure is the basic component of the learner (e.g. a neuron or a synapse).

The first and larger part of this thesis focuses on the coarse-grained aspect. Specifically, we study the robustness of distributed Stochastic Gradient Descent (SGD is the work-horse algorithm behind most of today's machine learning success). We begin by proving that the standard deployment of SGD today is brittle, as this deployment typically consists of *averaging* the inputs from each learner. This leads to harmful consequences as the data that is used in machine learning comes from different and potentially unreliable sources. To account for the various types of failures (data poisoning, malicious users, software bugs, communication delays, hacked machines etc.), we adopt the general abstraction of arbitrary failures in distributed systems, namely, the *Byzantine failures* abstraction. We provide a sufficient condition for SGD to be Byzantine resilient and present three algorithms that satisfy our condition under different configurations.

The key algorithms that are introduced by this thesis are (1) Krum, a gradient aggregation rule (GAR) that we prove to be a robust alternative to averaging in synchronous settings; (2) Bulyan, a meta-algorithm that we prove to strengthen any given GAR in very high dimensional situations and (3) Kardam, a gradient filtering scheme that we prove to be Byzantine resilient in the more challenging asynchronous setting. For each of our algorithms, we also provide a few variants as well as a discussion of their practical limitations.

The second part of this thesis goes down to the fine-grained aspect. We focus on the special case of (artificial) neural networks. We view these networks as a weighted directed graph and prove upper bounds on the *forward propagated error* when the basic components (neurons and synapses) are failing. We also discuss the limitation of these bounds, how they could apply to future neuromorphic hardware and how they could inform on other systems such as biological (metabolic) networks.

Keywords: robustness, distributed systems, Byzantine fault tolerance, machine learning, aggregation, poisoning, neural networks.

Résumé

Qu'il se produise dans un substrat biologique ou artificiel, l'*apprentissage* est un phénomène distribué selon au moins deux aspects. Premièrement parce que les données et les expériences pertinentes sont rarement obtenues à la même location, ce qui incite fortement les *apprenants* à travailler ensemble. Deuxièmement parce qu'un apprenant est lui-même un système distribué, composé de processus plus basiques ; c'est le changement des connections entres ces processus basiques qui permet l'apprentissage. Ce point de vue générique englobe un large spectre de situations d'apprentissage, des cervaux aux réseaux métaboliques dans l'organisme aux centres de données où plusieurs machines collaborent afin de recommander du contenu personalisé à un milliard d'utilisateurs d'un réseau social.

Dans les deux aspects susmentionnés, la capacité d'un système à faire face aux dysfonctionnements de certains de ses composants est crucial. Cette thèse explore la robustesse des systèmes d'apprentissage à partir de deux aspects. Le premier aspect est macroscopique, l'unité de dysfonctionnement est un apprenant tout entier. Le second est microscopique, l'unité de dysfonctionnement est un composant basique de l'apprenant (ex. un neurone ou une synapse). La première et majeure partie de cette thèse se focalise sur l'aspect macroscopique. Plus précisément, nous y étudions la robustesse de la descente du gradient stochastique (l'algorithme de DGS est au coeur de la plupart des succès en apprentissage machine aujourd'hui). Nous commençons par prouver que le déploiement standard de DGS aujourd'hui est fragile, vu qu'il consiste principalement à moyenner les entrées de chaque apprenant. Ceci conduit à des conséquences néfastes vu que les données utilisées en apprentissage machine proviennent de différentes sources, potentiellement corrompues. Pour tenir compte des différents types de dysfonctionnements (empoisonnement des données, utilisateurs malicieux, bugs de logiciel, délais de communication, machines piratées etc.), nous adoptons l'abstraction générale des fautes arbitraires dans les systèmes distribués, à savoir celle des *fautes Byzantines*. Nous fournissons une condition nécessaire à DGS pour qu'il soit tolérant aux fautes Byzantines et nous présentons trois algorithmes qui satisfont cette condition sous différentes configurations.

Les algorithmes clés qui sont introduits par cette thèse sont (1) Krum, une règle d'agrégation de gradients (RAG) que nous prouvons être une alternative robuste pour l'agrégation de gradients en configuration synchrone ; (2) Bulyan, un méta-algorithme que nous prouvons être capable de renforcer toute RAG en très haute dimension et (3) Kardam, un schèma de filtrage de gradients que nous prouvons être tolérant aux Byzantins dans les configurations asynchrones, qui sont plus délicates. Pour chacun de ces algorithmes, nous fournissons aussi quelques variantes ainsi qu'une discussion de leurs limitations pratiques.

La seconde partie de cette thèse se planche sur l'aspect microscopique. Nous nous focalisons sur le cas spécial des réseaux de neurones (artificiels). Nous modélisons ces réseaux comme des graphes dirigés pondérés et nous prouvons des bornes supérieures sur la *propagation d'erreur vers l'avant (PEVA)* quand des composantes basiques (neurones ou synapses) dysfonctionnent. Nous discutons aussi les limitations de ces bornes, comment elles peuvent s'appliquer au futur matériel neuromorphique et comment elles peuvent nous informer sur d'autres systèmes comme les réseaux biologiques (métaboliques).

Contents

List of Figures

1 Introduction

A *learning system* is one that uses data to refine its workings. Learning systems are ubiquitous in biology, where adaptation is the key to survival. In computer science, learning systems are a promising complement to traditional, hand-programmed systems and are the driving force behind the current growth in artificial intelligence (AI). The flexibility that is offered by learning systems seems, however, to pose serious challenges to their robustness.

Before we delve into the technical motivations of this thesis, let us first insist on the urgency of the research on AI safety. We illustrate this urgency using one of the many aspects of AI safety, the robustness to *data poisoning*, which motivates a significant portion of our work.

1.1 Learning Systems Among Us

If you ask the general public about AI safety today, you will probably hear about killer robots or rogue self-driving cars. AI safety is thought of as a concern that can wait for the far **future**. In fact, AI safety is not just important as a long-term issue, the lack of safety in **already deployed** learning systems is an actual threat. Technical AI safety research is a pressing concern in the **present**.

Take the example of one of the most influential learning systems in our societies today: the recommender systems of social networks. Consider YouTube. Every hour, 30 000 hours worth of videos are uploaded to the platform[1]. YouTube then has to sort and rank the content in order to make recommendations to users. Reportedly, there are today about 5 million views per minute on YouTube, more than there are searches on Google or logins on Facebook. This makes YouTube one of the most influential media today. Most importantly, 70 % of these views are recommended by the platform and not directly searched by the viewer[2]. Manually programming a recommender system such as the one used by YouTube is a hopeless task given its complexity. Instead, a learning system is leveraging data to offer meaningful recommendations.

[1] YouTube by the Numbers: https://www.youtube.com/about/press/

[2] According to Neal Mohan, YouTube's chief product officer. https://www.cnet.com/news/youtube-ces-2018-neal-mohan/

One particular vulnerability of learning systems is that, since they learn from data, they are prone to *data poisoning*. Poisoning describes the act of degrading the quality of a learning system by feeding it maliciously mislabelled data before training.

Very few poisoned data is often enough. In February 2018, after the tragic Parkland High School shooting in Florida, YouTube's front page featured a video defaming teenager survivors Emma Gonzalez and David Hogg among others as "paid crisis actors pretending to be survivors" as they rose to prominence while campaigning for gun control. YouTube had to publicly apologize but the harm was done and the debate on gun control in the U.S. successfully poisoned by false claims. Some survivors of the shooting even received death threats[3]. These threats resulted from a *single* uploaded video.

An even larger scale of poisoning is the spread of anti-vaccine propaganda, put on steroids by recommender systems of social media. Consider a video that convinces young parents that vaccines are dangerous for their babies and is labeled as 'health advice'; this is arguably data poisoning, given the medical consensus on vaccines. For instance, the World Health Organization has declared vaccine hesitancy among the top 10 global health threats in 2019, reporting a surge of 30% in measles infections and resurgence in countries that were close to eliminating the disease [136]. Some estimate that about 1500 U.S deaths per year are linked to come-back diseases, due to anti-vaccine. The French medical research agency INSERM is noting a similar growth of anti-vaccine surge in western Europe[4]. Recent research suggests a dangerous role played by social medias' recommender systems[5] in the surge of the anti-vaccine resentment [150, 124, 149, 48].

From fuelling an ongoing ethnic cleansing[6] in Myanmar[7] to enabling interference in the elections of the most powerful country in the world[8], the past three years gave us numerous cases where the amplification effect of social media calls for urgent research on securing learning systems and making them less prone to poisoning. Poisoning is only one among many other pressing questions in AI safety [8], but it is probably the most urgent motivation for the type of research that the larger part (Part 1) of this thesis tackles.

In the following two sections, we present the main technical motivations of our work.

[3] https://goo.gl/DaK5X9

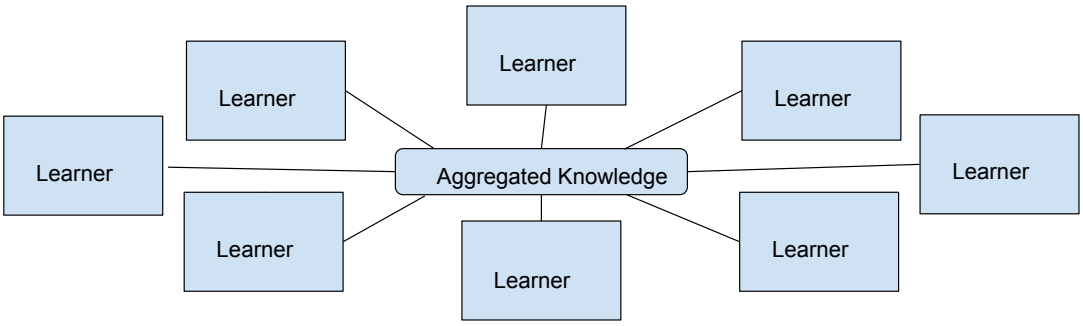
[4] From the INSERM website: https://goo.gl/J24Zud

[5] 2019 was a good year in terms of awareness. In a positive development, most platforms acknowledged their vulnerability to poisoning and role in the aforementioned social issues while initiating research [86] to combat these vulnerabilities. Platforms such as YouTube, Facebook, Twitter or Pinterest explicitly mentioned phony medical advice or anti-vaccine resentment in their different statements.

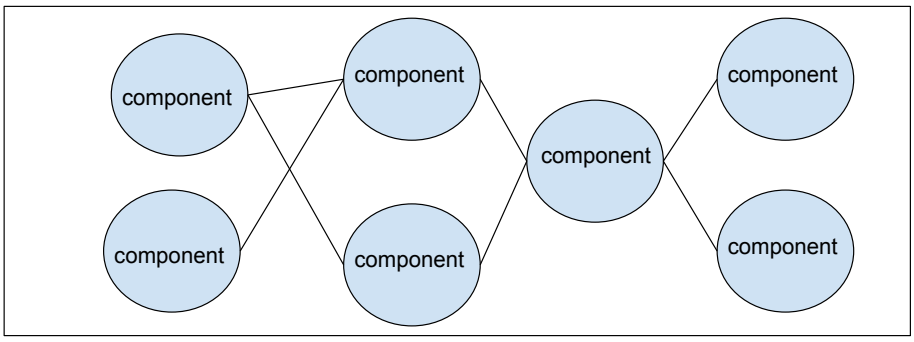
[6] The United Nations Human Rights Council: Report of Independent International Fact-Finding Mission on Myanmar (27 August 2018).

[7] *"We were too slow to respond to the concerns raised by civil society, academics and other groups in Myanmar"*. Mia Garlick, Facebook's director of Asia Pacific policy told Reuters. https://www.reuters.com/investigates/special-report/myanmar-facebook-hate/

[8] U.S. House of Representatives Permanent Select Committee on Intelligence: The Internet Research Agency and Advertisements. https://intelligence.house.gov/social-media-content/



A distributed system, aggregating knowledge from many learners (**Part 1 of the thesis**)



Zoom on one learner, itself a distributed system (**Part 2 of the thesis**)

Figure 1.1 – The coarse-grained aspect (Part 1) and the fine-grained aspect (Part 2) of this thesis.

1.2 Robust Distributed Learning

At a very high level, (machine) learning could be seen as an attempt to *aggregate* knowledge from data-points and update decision-making algorithms accordingly. A robust learning mechanism should learn from data without being vulnerable to malicious input.

The increasing amount of data available [44], together with the growing complexity of machine learning models [153], has led to learning schemes that require a lot of computational resources. As a consequence, most industry-grade machine-learning implementations are now distributed [1]. For example, as of 2012, Google reportedly used 16.000 processors to train an image classifier [116]. More recently, attention has been given to federated learning and federated optimization settings [95, 96, 118] with a focus on communication efficiency. However, distributing a computation over several machines (worker processes) induces a higher risk of failures. These include crashes and computation errors, stalled processes, biases in the way the data samples are distributed among the processes, but also, in the worst case, attackers trying to compromise the entire system. The most robust system is one that tolerates *Byzantine* failures [100], i.e., completely arbitrary behaviors of some of the processes.

A classical approach to mask failures in distributed systems is to use a state machine replication protocol [146], which requires however state transitions to be applied by all processes. In the case of distributed machine learning, this constraint can be translated in two ways: either *(a)* the processes agree on a sample of data based on which they update their local parameter vectors, or *(b)* they agree on how the parameter vector should be updated. In case *(a)*, the sample of data has to be transmitted to each process, which then has to perform a heavyweight computation to update its local parameter vector. This entails communication and computational costs that defeat the entire purpose of distributing the work. In case *(b)*, the processes have no way to check if the chosen update for the parameter vector has indeed been computed correctly on real data: a Byzantine process could have proposed the update and may easily prevent the convergence of the learning algorithm. Neither of these solutions is satisfactory in a realistic distributed machine learning setting.

In fact, most learning algorithms today rely on a core component, namely *stochastic gradient descent* (SGD) [24, 77], whether for training neural networks [77], regression [182], matrix factorization [69] or support vector machines [182]. In all those cases, a cost function – depending on the parameter vector – is minimized based on stochastic estimates of its gradient. (SGD is a stochastic approximation of gradient descent (GD) optimization, it replaces the actual gradient (calculated from the entire data set) by an estimate calculated from a randomly selected subset of the data). Distributed implementations of SGD [181] typically take the following form: a single parameter server is in charge of updating the parameter vector, while worker processes perform the actual update estimation. More specifically, the parameter server executes learning rounds, during each of which, the parameter vector is broadcast to the workers. In turn, each worker computes an estimate of the update to apply (an estimate of the *gradient*), and the parameter server aggregates their results to finally update the parameter vector. Today, this aggregation is typically implemented through averaging [140], or variants of it [181, 108, 164].

Stochastic Gradient Descent (SGD), is arguably the backbone of the most successful machine learning methods [103, 1, 44, 23]. Gradient Descent (GD), the underlying principle of SGD, is so straightforward that we find traces of it back in the first days of calculus in the works of Cauchy and Newton. In particular, GD relies on a simple observation: given a function Q, depending on a parameter x, if one keeps updating x in the opposite direction of the gradient of Q, with reasonably small, but not too small [23] steps, x eventually reaches the global minimum if Q is convex or, if Q is not convex[9], reaches a region where Q is either flat or in some *local minima*. SGD is the lightweight version of GD, where a sample is drawn at random from the entire data-set to estimate the gradient of Q on that sample and use as an approximation for the gradient on the total data -set. But for SGD to work, the aforementioned estimation of the gradient should however be done by correct and non-malicious workers.

Beyond image recognition or video labeling for social networks, SGD–based machine learning is venturing into safety–critical applications, like health–care [80] and transportation [22]. Meanwhile, a growing body of work, coined *Adversarial Machine Learning* [20, 72, 70, 99] is unveiling

[9] In practically interesting cases such as neural networks, Q is far from being convex [34, 35].

serious vulnerabilities in some of the most performing algorithms. Essentially, the general effort towards robust ML is conducted against three kinds of attacks: *poisoning* ones [19, 94] where an adversary injects poisoned data during the training phase, *exploratory* attacks, where a curious attacker attempts to infer privacy–sensitive information, and *evasion* attacks, where attackers try to fool an already trained model with adversarial inputs. The three fronts are complementary and each kind of attack poses a challenge on its own. Most of this thesis falls under the *poisoning* case.

1.3 Robust Learning Machines

We introduced the previous section by saying that (machine) learning could be seen as an attempt to *aggregate* knowledge from data-points and update decision-making algorithms accordingly, and we discussed the robustness of the *learning mechanism*. What about the robustness of the *decision making algorithms* themselves, once learning has happened?

The most powerful of these decision making algorithms today turns out to be the ones that contain many simple components and parameters. Among these, (artificial) Neural Networks (NNs) are responsible for the current success and regain of interest in machine learning [103]. These networks are very loosely inspired by the mammalian brain. Comprised of simple computing units, *neurons*, connected with simple links, *synapses*, NNs learn by modifying the *weights* of these links.

Biological plausibility, together with scalability, call for going one step further and considering each neuron as a *single* physical entity (that can fail *independently*), i.e., to consider genuinely distributed neural networks [123]. This approach is considered for example in the Human Brain project [117], trying to emulate the mammalian brain, or the various works on hardware-based neural networks, coined *neuromorphic* chips [82]. More recently, teams from IBM reported [61, 59] a successful neuromorphic implementation of neural networks that require a running power as low as 25 mW to 275 mW. In those settings, the unit of failure is one single neuron or synapse, and not a whole machine as in the previous section (c.f. Figure 1.1).

In the later part of our work, we explore this granularity and view a neural network as a distributed system where neurons can fail independently. We ask what is the maximum number of such failures that can be masked by the neural network, i.e., without having any impact on the overall computation. Addressing this question goes, however, first through precising it.

It is actually well known that the failure of neurons can be tolerated through additional learning phases [138, 159]. However, stopping a neural network and recovering its failures through a new learning phase is not an option for critical applications [36, 46, 63]. One can also consider specific a priori learning schemes that make it possible to tolerate failures a posteriori, e.g., shutting down parts of the network while learning, in order to cope with failures at run-time (dropout) [151, 91].

Our question can then be posed as follows: if (a) we do not make any assumption on the learning scheme and (b) we preclude the possibility of adding learning phases to recover from failures

when the neural network is in the deployment phase, what is the maximum number of faulty neurons that can be tolerated?

At this point, the question might sound trivial and the answer could be simply: *none*. Indeed, how could a neural network tolerate failures if it was not specifically devised with that purpose in mind? More specifically, if the failure of a number of neurons do not impact the overall result, then these neurons could have been eliminated from the design of that network in the first place. In fact, the reason why the question is nontrivial is *over-provisioning* [101][10]. Indeed, neural networks are rarely built with the minimal number of neurons to perform a computation. To estimate exactly this minimal number, one needs to know the target function the network should approximate, which by definition is unknown since the sole mainspring for machine learning is that we only know a finite number of the values of the target function: the learning data set. In fact, it has been experimentally observed that over-provisioning leads to robustness to the loss of neurons [119, 32, 77, 91]. Yet, the exact relation between the over-provision and the actual number of failures to be tolerated has never been precisely established. To the best of our knowledge, our work establishes this relation for the first time.

1.4 Contributions

1.4.1 Byzantine Resilient SGD

We study the resilience to Byzantine failures of distributed implementations of Stochastic Gradient Descent (SGD). So far, distributed machine learning frameworks have largely ignored the possibility of failures, especially arbitrary (i.e., Byzantine) ones. Causes of failures include software bugs, network asynchrony, biases in local datasets, as well as attackers trying to compromise the entire system. Assuming a set of n workers, up to f being Byzantine, we ask how resilient can SGD be, without limiting the dimension, nor the size of the parameter space. We first show that no gradient aggregation rule based on a linear combination of the vectors proposed by the workers (i.e, current approaches) tolerates a single Byzantine failure. We then formulate a resilience property of the aggregation rule capturing the basic requirements to guarantee convergence despite f Byzantine workers. We propose *Krum*, an aggregation rule that satisfies our resilience property. We also report on experimental evaluations of Krum. To the best of our knowledge, Krum was the first provably Byzantine-resilient algorithm for distributed SGD.

These results are provided in Chapters 2 and 3 and were initially published in the following.

Peva Blanchard; El-Mahdi El-Mhamdi; Rachid Guerraoui; Julien Stainer (2017). Machine learning with adversaries: Byzantine tolerant gradient descent. In Advances in Neural Information Processing Systems (NeurIPS) (pp. 119-129).

[10] Another aspect of over-provisioning, which this thesis does not directly address is over-parametrization. In particular, recent efforts are being made to understand why neural networks with an excessive number of parameters do not over-fit [83, 33].

Peva Blanchard; El-Mahdi El-Mhamdi; Rachid Guerraoui; Julien Stainer (2017). Brief Announcement: Byzantine-Tolerant Machine Learning. In Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC) (pp. 455-457). ACM.

1.4.2 High Dimensional Vulnerabilities in Distributed Non-Convex Optimization

While machine learning is going through an era of celebrated success, concerns have been raised about the vulnerability of its backbone: SGD. Recent approaches have been proposed to ensure the robustness of distributed SGD against adversarial (Byzantine) workers sending *poisoned* gradients during the training phase. Some of these approaches have been proven *Byzantine–resilient*: they ensure the *convergence* of SGD despite the presence of a minority of adversarial workers. We show in this part of the work that *convergence is not enough*. In high dimension $d \gg 1$, an adversary can build on the loss function's non–convexity to make SGD converge to *ineffective* models. More precisely, we bring to light that existing Byzantine–resilient schemes leave a *margin of poisoning* of $\Omega\big(f(d)\big)$, where $f(d)$ increases at least like $\sqrt{d}$. Based on this *leeway*, we build a simple attack, and experimentally show its strong to utmost effectivity on CIFAR–10 and MNIST. We introduce *Bulyan*, and prove it significantly reduces the attacker's leeway to a narrow $\mathcal{O}(\frac{1}{\sqrt{d}})$ bound. We empirically show that Bulyan does not suffer the fragility of existing aggregation rules and, at a reasonable cost in terms of required batch size, achieves convergence *as if* only non–Byzantine gradients had been used to update the model.

These results are provided in Chapter 4 and appeared in the following.

El-Mahdi El-Mhamdi; Rachid Guerraoui and Sébastien Rouault (2018). The Hidden Vulnerability of Distributed Learning in Byzantium. In International Conference on Machine Learning (ICML) (pp. 3518-3527).

1.4.3 Asynchronous Byzantine Resilient SGD

Asynchronous distributed machine learning solutions have proven very effective so far, but always assuming perfectly functioning workers. In practice, some of the workers can however exhibit Byzantine behavior, caused by hardware failures, software bugs, corrupt data, or even malicious attacks. In this part of the work, we introduce *Kardam*, the first distributed asynchronous stochastic gradient descent (SGD) algorithm that copes with Byzantine workers. Kardam consists of two complementary components: a filtering and a dampening component. The first is scalar-based and ensures resilience against $\frac{1}{3}$ Byzantine workers. Essentially, this filter leverages the Lipschitzness of cost functions and acts as a self-stabilizer against Byzantine workers that would attempt to corrupt the progress of SGD. The dampening component bounds the convergence rate by adjusting to stale information through a generic gradient weighting scheme. We prove that Kardam guarantees almost sure convergence in the presence of asynchrony and Byzantine behavior, and we derive its convergence rate. We evaluate Kardam on the CIFAR-100 and EMNIST datasets and measure its overhead with respect to non Byzantine-resilient solutions. We empiri-

cally show that Kardam does not introduce additional noise to the learning procedure but does induce a slowdown (the cost of Byzantine resilience) that we both theoretically and empirically show to be less than f/n, where f is the number of Byzantine failures tolerated and n the total number of workers. Interestingly, we also empirically observe that the dampening component is interesting in its own right for it enables to build an SGD algorithm that outperforms alternative staleness-aware asynchronous competitors in environments with honest workers.

These results are provided in Chapter 5 and appeared in the following.

Georgios Damaskinos, El-Mahdi El-Mhamdi; Rachid Guerraoui; Rhicheek Patra and Mahsa Taziki (2018). Asynchronous Byzantine Machine Learning (the case of SGD). In International Conference on Machine Learning (ICML) (pp. 1153-1162).

1.4.4 Neural Networks as a Distributed System

In this part, we view a multilayer neural network as a distributed system of which neurons can fail independently, and we evaluate its robustness in the absence of any (recovery) learning phase. We give tight bounds on the number of neurons that can fail without harming the result of a computation. To determine our bounds, we leverage the fact that neural activation functions are Lipschitz-continuous. Our bound is on a quantity, we call the *Forward Error Propagation*, capturing how much error is propagated by a neural network when a given number of components is failing, computing this quantity only requires looking at the topology of the network, while experimentally assessing the robustness of a network requires the costly experiment of looking at all the possible inputs and testing all the possible configurations of the network corresponding to different failure situations, facing a discouraging combinatorial explosion.

We distinguish the case of neurons that can fail and stop their activity (*crashed* neurons) from the case of neurons that can fail by transmitting arbitrary values (*Byzantine* neurons). In the crash case, our bound involves the number of neurons per layer, the Lipschitz constant of the neural activation function, the number of failing neurons, the synaptic weights and the depth of the layer where the failure occurred. In the case of Byzantine failures, our bound involves, in addition, the synaptic transmission capacity. Interestingly, as we prove, our bound can easily be extended to the case where synapses can fail.

More precisely, we present tight bounds on the number of faulty neurons a *feed-forward neural network*[11] can tolerate without harming the computation result, nor requiring any additional learning phase or specific prior learning algorithm. In fact, our bounds are not simply expressed in terms of *numbers of failures*, but in terms of *weight* and *failure* distribution. Indeed, unlike process failures in traditional distributed computing that all have the same effect, neuron failures do not: they are *weighted*. In the general case of *multilayer*, or so called *deep*, networks, we

[11] Feed-forward networks are the most common in the literature [77], and today's most popular topology: the convolutional neural network for example [102], is a particular case of the feed-forward topology. We will discuss some of these in Section 7.3.

formulate our results in the form of a *fault-per-layer distribution*.

Our results are obtained using analytic properties of the different mathematical components of a neural network, namely the activation function, the synaptic weights, and the neural computation model. By relying on the very fact that neural activation functions are bounded, and in practice Lipschitzian [147], we set precise bounds on the error propagation over the layers, and subsequently establish tight bounds on the number of failures a neural network can tolerate.

We present three applications of our results. The first is a quantification of the effect of memory cost reduction on the accuracy of a neural network. The second is a quantification of the amount of information any neuron needs from its preceding layer, enabling thereby a boosting scheme that prevents neurons from waiting for unnecessary signals. Our third application is a quantification of the trade-off between neural networks robustness and learning cost.

These results are provided in chapters 6 and 7 and appeared in the following.

El-Mahdi El-Mhamdi; Rachid Guerraoui (2017). When Neurons Fail. In 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS) (pp. 1028-1037).

El-Mahdi El-Mhamdi; Rachid Guerraoui and Sébastien Rouault (2017). On the robustness of a neural network. In 2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS) (pp. 84-93).

1.5 Roadmap

Part I is the major contribution of this thesis, it presents the coarse-grained view on robustness and the question of Byzantine resilient machine learning. Specifically, it contains the solutions we have developed in synchronous, high dimensional, and asynchronous contexts, namely Krum, Bulyan and Kardam. Chapter 2 contains the basic model and preliminary formalism that is shared by the following chapters on distributed learning. In turn, each later chapter contains definitions of concepts that are specific to that chapter and a comparison with related work that are most related to the results of the chapter. Specifically, Chapter 3 tackles synchronous gradient descent and introduces our solution, *Krum*. Chapter 4 addresses the vulnerabilities that arise in high dimensional models and introduces our solution, *Bulyan*. Finally, Chapter 5 addresses the asynchronous setting and introduces our solution, *Kardam*.

Part II is the other contribution of this thesis, the fine-grained view where we examine the robustness of learning machines themselves. Chapter 6 provides the model as well as the formal definitions that are needed for this part. Chapter 7 provides our set of bounds on error propagation and the condition on the number of failing components that can derived from these bounds.

In Part III, we discuss the results of this thesis in the context of the ongoing effort to understand the robustness of learning systems. Chapter 8 does so for our work on distributed learning, connecting it to the growing body of work on Byzantine resilient machine learning that developed

in the past three years. This chapter also presents our research agenda on this direction which is two-fold: (1) How to use our algorithmic toolbox to build reliable distributed learning systems and (2) How to improve our theory by discarding some of our limiting hypothesis. Chapter 9 looks at the less explored fine-grained question and discusses the limitations of our results. This chapter also introduces our research agenda on this front which has two facets. (1) One facet where we question our hypotheses to provide better bounds on error propagation in learning systems, and (2) Another facet where we collaborate with biologists in an effort to understand the robustness of another class of learning systems, biological (metabolic) networks.

Robust Distributed Learning Part I

2 Preliminaries

2.1 Distributed Machine Learning with SGD

Machine learning schemes involve two ingredients: a d-dimensional parameter, x, that defines an algorithm, and a cost function Q, such that $Q(x)$ estimates the error made by the algorithm when tuned with the value x. On a high level, *learning* consists in finding the best algorithm, namely, the one with value x^*, the minimizer of the cost, i.e. $x^* = \arg\min_{x \in \mathbb{R}^d} Q(x)$. This high level view applies whether the learning is *supervised*, i.e. guided by the labels in the data, *unsupervised*, i.e. only guided by the structure of unlabeled data, or *reinforced*, i.e. guided by rewards depending on what parameter x dictates the action to be taken when facing a given data (an environmental state). Optimizing cost functions is the backbone of machine learning.

The parameter is updated as follows:

$$x_{t+1} = x_t - \eta_t \nabla Q(x_t), \quad \text{(GD parameter update)} \tag{2.1}$$

where x_{t+1} and x_t refer to the next and current value of the parameter, $\nabla Q(x_t)$ the current gradient of the cost function and η_t is a step-size, called the *learning rate*. In practice, $\nabla Q(x_t)$ is not computed, instead, a random sample ξ^t is drawn at each step to perform an approximation $G(x_t, \xi^t)$ such that, under reasonable assumptions, replacing $\nabla Q(x_t)$ by $G(x_t, \xi^t)$ in Equation 2.1 leads to minimizing Q. The "S" in SGD refers to this stochastic approximation of the gradient.

In a distributed setting, the (heavy) task of computing $G(x_t, \xi^t)$ is outsourced by a *parameter server* (or simply "server") to several *workers*. Typically following the now standard parameter server scheme [105, 106]. The computation is divided into *epochs*, i.e., parameter updates. The server gathers gradient estimations from the workers and employs them to perform a single model update, then broadcasts the new model to every worker for computing new gradient estimations.

When both the server-worker communication latency and the workers computing capabilities are homogeneous and reliable, it is preferable to aggregate all the workers' gradient estimations (typically by averaging them) and perform a *synchronous* update using Equation 2.1. This prov-

ably [26] speeds up learning, in the sense that the server performs fewer updates to reach the optimum, x^*, compared to if it was learning alone without leveraging many workers.

If however the workers are heterogeneous in their computing or communication capacities, it can become preferable to perform *asynchronous* updates [109, 112, 79, 105]. Ones where the server does not wait for late workers, but instead updates the parameter as soon as a gradient estimation is delivered from a worker, the server then broadcasts the new value of the parameter.

2.2 Common Model

We consider the general distributed system model of [1] commonly referred to as the *parameter server* and consisting of a parameter server[1], and n workers, f of them possibly Byzantine (behaving arbitrarily). Computation is divided into (infinitely many) synchronous rounds. During round t, the parameter server broadcasts its parameter vector $x_t \in \mathbb{R}^d$ to all the workers. Each correct worker p computes an estimate $V_p^t = G(x_t, \xi_p^t)$ of the gradient $\nabla Q(x_t)$ of the cost function Q, where ξ_p^t is a random variable representing, e.g., the sample (or a mini-batch of samples) drawn from the dataset. A Byzantine worker b proposes a vector V_b^t which can deviate arbitrarily from the vector it is supposed to send if it was correct, i.e., according to the algorithm assigned to it by the system developer (see Figure 2.1).

Since the communication is synchronous, if the parameter server does not receive a vector value V_b^t from a given Byzantine worker b, then the parameter server acts as if it had received the default value $V_b^t = 0$ instead.

The parameter server computes a vector $F(V_1^t, \ldots, V_n^t)$ by applying a deterministic function F (aggregation rule) to the vectors received. We refer to F as the *aggregation rule* of the parameter server. The parameter server updates the parameter vector using the following SGD equation

$$x_{t+1} = x_t - \gamma_t \cdot F(V_1^t, \ldots, V_n^t).$$

The correct (non-Byzantine) workers are assumed to compute unbiased estimates of the gradient $\nabla Q(x_t)$. More precisely, in every round t, the vectors V_i^t's proposed by the correct workers are independent identically distributed random vectors, $V_i^t \sim G(x_t, \xi_i^t)$ with $\mathbb{E}_{\xi_i^t} G(x_t, \xi_i^t) = \nabla Q(x_t)$. This can be achieved by ensuring that each sample of data used for computing the gradient is drawn uniformly and independently, as classically assumed in the literature of machine learning [23].

The Byzantine workers have full knowledge of the system, including the aggregation rule F as well as the vectors proposed by the workers. They can furthermore collaborate with each other [113].

Also, note that the parameter server, as is any server in distributed settings, can be an abstraction implemented by the workers, in a peer-to-peer version of distributed machine learning. In

[1] The parameter server is assumed to be reliable. Classical techniques of state-machine replication can be used to ensure this.

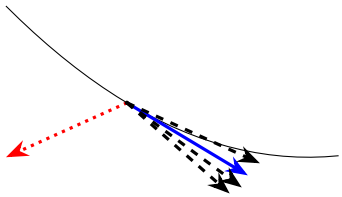


Figure 2.1 – The gradient estimates computed by correct workers (black dashed arrows) are distributed around the actual gradient (solid arrow) of the cost function (thin black curve). A Byzantine worker can propose an arbitrary vector (red dotted arrow).

practice, this can be dictated by privacy concerns, imagine for instance a group of medical institutions which, collectively, are trying to learn the optimal parameter vector x for a neural network that predicts cancer, we can imagine that each institution has access to a subgroup of patients on which it locally computes an estimate of the gradient, it only share this estimate, or differentially private versions of it [2] with the other workers, without sharing its local data; this way, the group of workers are collectively learning the best parameter x^* without harming the privacy of their patients.

2.3 Byzantine Resilience

We first show in this chapter that no linear combination (current approaches) of the updates proposed by the workers can tolerate a *single* Byzantine worker. Basically, a single Byzantine worker can force the parameter server to choose any arbitrary vector, even one that is too large in amplitude or too far in direction from the other vectors. Clearly, the Byzantine worker can prevent any classic averaging-based approach to converge. We then formulate a condition for an aggregation rule to be Byzantine resilient.

In most SGD-based learning algorithms used today [24, 77, 69], the aggregation rule consists in computing the average [2] of the input vectors. Lemma 1 below states that no linear combination of the vectors can tolerate a single Byzantine worker. In particular, averaging is not Byzantine resilient.

Lemma 1. *Consider an aggregation rule F_{lin} of the form $F_{lin}(V_1, \ldots, V_n) = \sum_{i=1}^{n} \lambda_i \cdot V_i$, where the λ_i's are non-zero scalars. Let U be any vector in $\mathbb{R}^d$. A single Byzantine worker can make F always select U. In particular, a single Byzantine worker can prevent convergence.*

Proof. Immediate: if the Byzantine worker proposes $V_n = \frac{1}{\lambda_n} \cdot U - \sum_{i=1}^{n-1} \frac{\lambda_i}{\lambda_n} V_i$, then $F = U$. Note that the parameter server could cancel the effects of the Byzantine behavior by setting, for example, λ_n to 0. This however requires means to detect which worker is Byzantine. □

In the following, we define basic requirements on an appropriate Byzantine-resilient aggregation rule. Intuitively, the aggregation rule should output a vector F that is not too far from the "real"

[2] Or a closely related rule.

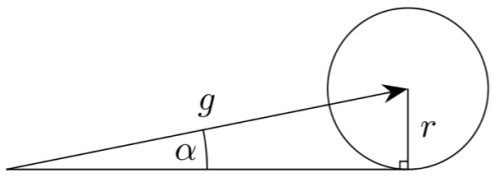


Figure 2.2 – If $\|\mathbb{E}F - g\| \leq r$ then $\langle \mathbb{E}F, g\rangle$ is bounded below by $(1-\sin\alpha)\|g\|^2$ where $\sin\alpha = r/\|g\|$.

gradient g, more precisely, the vector that points to the steepest direction of the cost function being optimized. This is expressed as a lower bound (condition *(i)*) on the scalar product of the (expected) vector F and g. Figure 2.2 illustrates the situation geometrically. If $\mathbb{E}F$ belongs to the ball centered at g with radius r, then the scalar product is bounded below by a term involving $\sin\alpha = r/\|g\|$.

Condition *(ii)* is more technical, and states that the moments of F should be controlled by the moments of the (correct) gradient estimator G. The bounds on the moments of G are classically used to control the effects of the discrete nature of the SGD dynamics [23]. Condition *(ii)* allows to transfer this control to the aggregation rule.

Definition 1 ((α, f)-Byzantine Resilience)**.** *Let $0 \leq \alpha < \pi/2$ be any angular value, and any integer $0 \leq f \leq n$. Let $V_1, \ldots, V_n$ be any independent identically distributed random vectors in $\mathbb{R}^d$, $V_i \sim G$, with $\mathbb{E}G = g$. Let $B_1, \ldots, B_f$ be any random vectors in $\mathbb{R}^d$, possibly dependent on the V_i's. aggregation rule F is said to be (α, f)-Byzantine resilient if, for any $1 \leq j_1 < \cdots < j_f \leq n$, vector*

$$F = F(V_1, \ldots, \underbrace{B_1}_{j_1}, \ldots, \underbrace{B_f}_{j_f}, \ldots, V_n)$$

satisfies (i) $\langle \mathbb{E}F, g\rangle \geq (1-\sin\alpha)\cdot\|g\|^2 > 0$ *and* (ii) *for $r = 2,3,4$, $\mathbb{E}\|F\|^r$ is bounded above by a linear combination of terms $\mathbb{E}\|G\|^{r_1}\ldots\mathbb{E}\|G\|^{r_{n-1}}$ with $r_1 + \cdots + r_{n-1} = r$.*

3 Krum: Synchronous Distributed Gradient Descent

3.1 Introduction

Choosing the appropriate aggregation of the vectors proposed by the workers turns out to be challenging. A non-linear, *squared-distance-based* aggregation rule, that selects, among the proposed vectors, the vector "closest to the barycenter" (for example by taking the vector that minimizes the sum of the squared distances to every other vector), might look appealing. Yet, such a squared-distance-based aggregation rule tolerates only a single Byzantine worker. Two Byzantine workers can collude, one helping the other to be selected, by moving the barycenter of all the vectors farther from the "correct area".

Figure 3.1 – Selecting the vector that minimizes the sum of the squared distances to other vectors does not prevent arbitrary vectors proposed by Byzantine workers from being selected if $f \geq 2$. If the gradients computed by the correct workers lie in area $\mathscr{C}$, the Byzantine workers can collude to propose up to $f-1$ vectors in an arbitrarily remote area $\mathscr{B}$, thus allowing another Byzantine vector b, close to the barycenter of proposed vectors, to be selected.

We formulate a Byzantine resilience property capturing sufficient conditions for the parameter server's aggregation rule to tolerate f Byzantine workers. Essentially, to guarantee that the cost will decrease despite Byzantine workers, we require the vector output chosen by the parameter server *(a)* to point, on average, to the same direction as the gradient and *(b)* to have statistical moments (up to the fourth moment) bounded above by a homogeneous polynomial in the moments of a correct estimator of the gradient. These two requirements enable us to re-use traditional convergence proofs in non-convex optimization, mostly the one in [23]. One way to ensure such a resilience property is to consider a *majority-based* approach, looking at every subset of $n-f$ vectors, and considering the subset with the smallest diameter. While this approach is more robust to Byzantine workers that propose vectors far from the correct area, its exponential

computational cost is prohibitive. Interestingly, combining the intuitions of the *majority-based* and *squared-distance*[1]*-based* methods, we can choose the vector that is somehow the closest to its $n-f$ neighbors. Namely, the one that minimizes the sum of squared distances to its $n-f$ closest vectors. This is the main idea behind our aggregation rule, we call *Krum*[2]. Assuming $2f+2<n$, we show that Krum satisfies the resilience property aforementioned and the corresponding machine learning scheme converges. An important advantage of Krum is its (local) time complexity ($O(n^2 \cdot d)$), linear in the dimension of the gradient, where d is the dimension of the parameter vector. (In modern machine learning, the dimension d of the parameter vector may take values in the hundreds of billions [163].) For simplicity of presentation, the version of Krum we first consider selects only one vector. We also discuss other variants.

We evaluate Krum experimentally, and compare it to classical averaging. We confirm the very fact that averaging does not stand Byzantine attacks, while Krum does. In particular, we report on attacks by omniscient adversaries – aware of a good estimate of the gradient – that send the opposite vector multiplied by a large factor, as well as attacks by adversaries that send random vectors drawn from a Gaussian distribution (the larger the variance of the distribution, the stronger the attack). We also evaluate the extent to which Krum might slow down learning (compared to averaging) when there are no Byzantine failures. Interestingly, as we show experimentally, this slow down occurs only when the mini-batch size is close to 1. In fact, the slowdown can be drastically reduced by choosing a reasonable mini-batch size. We also evaluate *Multi-Krum*, a variant of Krum, which, intuitively, interpolates between Krum and averaging, thereby allowing to mix the resilience properties of Krum with the convergence speed of averaging. Multi-Krum outperforms other aggregation rules like the medoid, inspired by the geometric median.

The code to reproduce experiments with Krum, Bulyan and other variants can be found in the following repository: https://github.com/LPD-EPFL/AggregaThor. This repository is part of an ongoing effort to build systems that are based on the algorithms we present in this thesis.

Chapter Organization.

Section 3.2 introduces our Krum function, computes its computational cost and proves its (α, f)-Byzantine resilience. Section 3.3 analyzes the convergence of a distributed SGD using Krum. Section 3.4 presents our experimental evaluation of Krum and Section3.5 introduces some variants of Krum. We discuss related work and open problems in Section 3.6. Some complementary experimental results are given in the Appendix.

[1] In all this chapter, distances are computed with the Euclidean norm.

[2] Krum, in Greek Κρούμος, was a Bulgarian Khan of the end of the eighth century, who undertook offensive attacks against the Byzantine empire. Bulgaria doubled in size during his reign.

3.2 The Krum Function

We now introduce *Krum*, our aggregation rule, which, we show, satisfies the (α, f)-Byzantine resilience condition. The barycentric aggregation rule $F_{bary} = \frac{1}{n}\sum_{i=1}^{n} V_i$ can be defined as the vector in $\mathbb{R}^d$ that minimizes the sum of squared distances [3] to the V_i's $\sum_{i=1}^{n} \left\| F_{bary} - V_i \right\|^2$. Lemma 1, however, states that this approach does not tolerate even a single Byzantine failure.

One could try to select the vector U *among* the V_i's which minimizes the sum $\sum_i \|U - V_i\|^2$, i.e., which is "closest to all vectors"[4]. However, because such a sum involves all the vectors, even those which are very far, this approach does not tolerate Byzantine workers: by proposing large enough vectors, a Byzantine worker can force the total barycenter to get closer to the vector proposed by another Byzantine worker. This situation is depicted in Figure 3.1. In other words, since this aggregation rule takes into account all the vectors, including the very remote ones, the Byzantine workers can collude to force the choice of the parameter server.

Our approach to circumvent this issue is to preclude the vectors that are too far away. More precisely, we define our *Krum* aggregation rule $\textsc{Kr}(V_1, \ldots, V_n)$ as follows. For any $i \neq j$, we denote by $i \to j$ the fact that V_j belongs to the $n - f - 2$ closest vectors to V_i. Then, we define for each worker i, the *score* $s(i) = \sum_{i \to j} \left\| V_i - V_j \right\|^2$ where the sum runs over the $n - f - 2$ closest vectors to V_i. Finally, $\textsc{Kr}(V_1, \ldots, V_n) = V_{i_*}$ where i_* refers to the worker minimizing the score, $s(i_*) \leq s(i)$ for all i.[5]

Lemma 2. *The expected time complexity of the Krum Function* $\textsc{Kr}(V_1, \ldots, V_n)$*, where* $V_1, \ldots, V_n$ *are* d*-dimensional vectors, is* $O(n^2 \cdot d)$

Proof. For each V_i, the parameter server computes the n squared distances $\left\| V_i - V_j \right\|^2$ (time $O(n \cdot d)$). Then the parameter server selects the first $n - f - 1$ of these distances (expected time $O(n)$ with Quickselect) and sums their values (time $O(n \cdot d)$). Thus, computing the score of all the V_i's takes $O(n^2 \cdot d)$. An additional term $O(n)$ is required to find the minimum score, but is negligible relatively to $O(n^2 \cdot d)$. □

Proposition 1 below states that, if $2f + 2 < n$ and the gradient estimator is accurate enough, (its standard deviation is relatively small compared to the norm of the gradient), then the Krum function is (α, f)-Byzantine-resilient, where angle α depends on the ratio of the deviation over the gradient.

Proposition 1. *Let* $V_1, \ldots, V_n$ *be any independent and identically distributed random* d*-dimensional vectors s.t* $V_i \sim G$*, with* $\mathbb{E}G = g$ *and* $\mathbb{E}\left\| G - g \right\|^2 = d\sigma^2$*. Let* $B_1, \ldots, B_f$ *be any* f *random vectors, possi-*

[3] Removing the square of the distances leads to the geometric median, we discuss this when optimizing Krum.

[4] One could suggest to remove the square in the computed distance, this would be the medoid, which we later compare to Krum but for which we have no theoretical guarantee. One could also pick the minimizer of the non-squared distances (geometric median), we also discuss this later.

[5] If two or more workers have the minimal score, we choose the one with the smallest identifier.

bly dependent on the V_i's. If $2f+2<n$ and $\eta(n,f)\sqrt{d}\cdot\sigma<\|g\|$, where

$$\eta(n,f) \underset{def}{=} \sqrt{2\left(n-f+\frac{f\cdot(n-f-2)+f^2\cdot(n-f-1)}{n-2f-2}\right)} = \begin{cases} O(n) & \text{if } f=O(n) \\ O(\sqrt{n}) & \text{if } f=O(1) \end{cases},$$

then the Krum function Kr *is (α,f)-Byzantine resilient where $0\le\alpha<\pi/2$ is defined by*

$$\sin\alpha = \frac{\eta(n,f)\cdot\sqrt{d}\cdot\sigma}{\|g\|}.$$

The condition on the norm of the gradient, $\eta(n,f)\cdot\sqrt{d}\cdot\sigma<\|g\|$, can be satisfied, to a certain extent, by having the (correct) workers compute their gradient estimates on mini-batches [23]. Indeed, averaging the gradient estimates over a mini-batch divides the deviation σ by the squared root of the size of the mini-batch.

Depending on the readers' interests, we give first the sketch of the proof, then we give the detailed proof.

Proof. (Sketch) Without loss of generality, we assume that the Byzantine vectors $B_1,\dots,B_f$ occupy the last f positions in the list of arguments of Kr, i.e., $\text{Kr}=\text{Kr}(V_1,\dots,V_{n-f},B_1,\dots,B_f)$. Let i_* be the index of the vector chosen by the Krum function. We focus on the condition *(i)* of (α,f)-Byzantine resilience (Definition 1).

Consider first the case where $V_{i_*}=V_i\in\{V_1,\dots,V_{n-f}\}$ is a vector proposed by a correct process. The first step is to compare the vector V_i with the average of the correct vectors V_j such that $i\to j$. Let $\delta_c(i)$ be the number of such V_j's.

$$\mathbb{E}\left\|V_i-\frac{1}{\delta_c(i)}\sum_{i\to\text{ correct }j}V_j\right\|^2 \le \frac{1}{\delta_c(i)}\sum_{i\to\text{ correct }j}\mathbb{E}\left\|V_i-V_j\right\|^2 \le 2d\sigma^2. \tag{3.1}$$

The last inequality holds because the right-hand side of the first inequality involves only vectors proposed by correct processes, which are mutually independent and follow the distribution of G.

Now, consider the case where $V_{i_*}=B_k\in\{B_1,\dots,B_f\}$ is a vector proposed by a Byzantine process. The fact that k minimizes the score implies that for all indices i of vectors proposed by correct processes

$$\sum_{k\to\text{ correct }j}\left\|B_k-V_j\right\|^2+\sum_{k\to\text{ byz }l}\|B_k-B_l\|^2 \le \sum_{i\to\text{ correct }j}\left\|V_i-V_j\right\|^2+\sum_{i\to\text{ byz }l}\|V_i-B_l\|^2.$$

Then, for all indices i of vectors proposed by correct processes

$$\left\|B_k-\frac{1}{\delta_c(k)}\sum_{k\to\text{ correct }j}V_j\right\|^2 \le \frac{1}{\delta_c(k)}\sum_{i\to\text{ correct }j}\left\|V_i-V_j\right\|^2+\frac{1}{\delta_c(k)}\underbrace{\sum_{i\to\text{ byz }l}\|V_i-B_l\|^2}_{D^2(i)}.$$

The term $D^2(i)$ is the only term involving vectors proposed by Byzantine processes. However, the correct process i has $n-f-2$ neighbors and $f+1$ non-neighbors. Therefore, there exists a correct process $\zeta(i)$ which is farther from i than every neighbor j of i (including the Byzantine neighbors). In particular, for all l such that $i \to l$, $\|V_i - B_l\|^2 \leq \|V_i - V_{\zeta(i)}\|^2$. Thus

$$\left\| B_k - \frac{1}{\delta_c(k)} \sum_{k \to \text{ correct } j} V_j \right\|^2 \leq \frac{1}{\delta_c(k)} \sum_{i \to \text{ correct } j} \left\| V_i - V_j \right\|^2 + \frac{n-f-2-\delta_c(i)}{\delta_c(k)} \left\| V_i - V_{\zeta(i)} \right\|^2. \quad (3.2)$$

Combining equations 3.1, 3.2, and a union bound yields $\|\mathbb{E}\mathrm{KR} - g\|^2 \leq \eta\sqrt{d}\|g\|$, which, in turn, implies $\langle \mathbb{E}\mathrm{KR}, g\rangle \geq (1-\sin\alpha)\|g\|^2$. Condition *(ii)* is proven by bounding the moments of KR with moments of the vectors proposed by the correct processes only, using the same technique as above. The full proof is provided in the supplementary material. □

Now we provide the detailed proof.

Proof. Without loss of generality, we assume that the Byzantine vectors $B_1, \ldots, B_f$ occupy the last f positions in the list of arguments of KR, i.e., $\mathrm{KR} = \mathrm{KR}(V_1, \ldots, V_{n-f}, B_1, \ldots, B_f)$. An index is *correct* if it refers to a vector among $V_1, \ldots, V_{n-f}$. An index is *Byzantine* if it refers to a vector among $B_1, \ldots, B_f$. For each index (correct or Byzantine) i, we denote by $\delta_c(i)$ (resp. $\delta_b(i)$) the number of correct (resp. Byzantine) indices j such that $i \to j$. We have

$$\begin{aligned} \delta_c(i) + \delta_b(i) &= n-f-2 \\ n-2f-2 \leq \delta_c(i) &\leq n-f-2 \\ \delta_b(i) &\leq f. \end{aligned}$$

We focus first on the condition *(i)* of (α, f)-Byzantine resilience. We determine an upper bound on the squared distance $\|\mathbb{E}\mathrm{KR} - g\|^2$. Note that, for any correct j, $\mathbb{E}V_j = g$. We denote by i_* the index of the vector chosen by the Krum function.

$$\begin{aligned} \left\|\mathbb{E}\mathrm{KR} - g\right\|^2 &\leq \left\| \mathbb{E}\left(\mathrm{KR} - \frac{1}{\delta_c(i_*)} \sum_{i_* \to \text{ correct } j} V_j \right) \right\|^2 \\ &\leq \mathbb{E} \left\| \mathrm{KR} - \frac{1}{\delta_c(i_*)} \sum_{i_* \to \text{ correct } j} V_j \right\|^2 \quad \text{(Jensen inequality)} \\ &\leq \sum_{\text{correct } i} \mathbb{E} \left\| V_i - \frac{1}{\delta_c(i)} \sum_{i \to \text{ correct } j} V_j \right\|^2 \mathbb{1}(i_* = i) \\ &+ \sum_{\text{byz } k} \mathbb{E} \left\| B_k - \frac{1}{\delta_c(k)} \sum_{k \to \text{ correct } j} V_j \right\|^2 \mathbb{1}(i_* = k) \end{aligned}$$

where $\mathbb{I}$ denotes the indicator function[6]. We examine the case $i_* = i$ for some correct index i.

$$\begin{aligned}
\left\| V_i - \frac{1}{\delta_c(i)} \sum_{i \to \text{correct } j} V_j \right\|^2 &= \left\| \frac{1}{\delta_c(i)} \sum_{i \to \text{correct } j} V_i - V_j \right\|^2 \\
&\leq \frac{1}{\delta_c(i)} \sum_{i \to \text{correct } j} \left\| V_i - V_j \right\|^2 \quad \text{(Jensen inequality)} \\
\mathbb{E} \left\| V_i - \frac{1}{\delta_c(i)} \sum_{i \to \text{correct } j} V_j \right\|^2 &\leq \frac{1}{\delta_c(i)} \sum_{i \to \text{correct } j} \mathbb{E} \left\| V_i - V_j \right\|^2 \\
&\leq 2d\sigma^2.
\end{aligned}$$

We now examine the case $i_* = k$ for some Byzantine index k. The fact that k minimizes the score implies that for all correct indices i

$$\sum_{k \to \text{correct } j} \left\| B_k - V_j \right\|^2 + \sum_{k \to \text{byz } l} \| B_k - B_l \|^2 \leq \sum_{i \to \text{correct } j} \left\| V_i - V_j \right\|^2 + \sum_{i \to \text{byz } l} \| V_i - B_l \|^2 .$$

Then, for all correct indices i

$$\begin{aligned}
\left\| B_k - \frac{1}{\delta_c(k)} \sum_{k \to \text{correct } j} V_j \right\|^2 &\leq \frac{1}{\delta_c(k)} \sum_{k \to \text{correct } j} \left\| B_k - V_j \right\|^2 \\
&\leq \frac{1}{\delta_c(k)} \sum_{i \to \text{correct } j} \left\| V_i - V_j \right\|^2 + \frac{1}{\delta_c(k)} \underbrace{\sum_{i \to \text{byz } l} \| V_i - B_l \|^2}_{D^2(i)} .
\end{aligned}$$

We focus on the term $D^2(i)$. Each correct process i has $n-f-2$ neighbors, and $f+1$ non-neighbors. Thus there exists a correct worker $\zeta(i)$ which is farther from i than any of the neighbors of i. In particular, for each Byzantine index l such that $i \to l$, $\| V_i - B_l \|^2 \leq \left\| V_i - V_{\zeta(i)} \right\|^2$. Whence

$$\begin{aligned}
\left\| B_k - \frac{1}{\delta_c(k)} \sum_{k \to \text{correct } j} V_j \right\|^2 &\leq \frac{1}{\delta_c(k)} \sum_{i \to \text{correct } j} \left\| V_i - V_j \right\|^2 + \frac{\delta_b(i)}{\delta_c(k)} \left\| V_i - V_{\zeta(i)} \right\|^2 \\
\mathbb{E} \left\| B_k - \frac{1}{\delta_c(k)} \sum_{k \to \text{correct } j} V_j \right\|^2 &\leq \frac{\delta_c(i)}{\delta_c(k)} \cdot 2d\sigma^2 + \frac{\delta_b(i)}{\delta_c(k)} \sum_{\text{correct } j \neq i} \mathbb{E} \left\| V_i - V_j \right\|^2 \mathbb{I}(\zeta(i) = j) \\
&\leq \left(\frac{\delta_c(i)}{\delta_c(k)} \cdot + \frac{\delta_b(i)}{\delta_c(k)} (n-f-1) \right) 2d\sigma^2 \\
&\leq \left(\frac{n-f-2}{n-2f-2} + \frac{f}{n-2f-2} \cdot (n-f-1) \right) 2d\sigma^2 .
\end{aligned}$$

[6] $\mathbb{I}(P)$ equals 1 if the predicate P is true, and 0 otherwise.

Putting everything back together, we obtain

$$\begin{aligned}\left\|\mathbb{E}\mathrm{KR} - g\right\|^2 &\leq (n-f)2d\sigma^2 + f\cdot\left(\frac{n-f-2}{n-2f-2} + \frac{f}{n-2f-2}\cdot(n-f-1)\right)2d\sigma^2 \\ &\leq \underbrace{2\left(n-f+\frac{f\cdot(n-f-2)+f^2\cdot(n-f-1)}{n-2f-2}\right)}_{\eta^2(n,f)} d\sigma^2.\end{aligned}$$

By assumption, $\eta(n,f)\sqrt{d}\sigma < \|g\|$, i.e., $\mathbb{E}\mathrm{KR}$ belongs to a ball centered at g with radius $\eta(n,f)\cdot\sqrt{d}\cdot\sigma$. This implies

$$\langle \mathbb{E}\mathrm{KR}, g\rangle \geq \left(\|g\| - \eta(n,f)\cdot\sqrt{d}\cdot\sigma\right)\cdot\|g\| = (1-\sin\alpha)\cdot\|g\|^2.$$

To sum up, condition *(i)* of the (α, f)-Byzantine resilience property holds. We now focus on condition *(ii)*.

$$\begin{aligned}\mathbb{E}\|\mathrm{KR}\|^r &= \sum_{\text{correct } i} \mathbb{E}\|V_i\|^r \mathbb{1}(i_* = i) + \sum_{\text{byz } k} \mathbb{E}\|B_k\|^r \mathbb{1}(i_* = k) \\ &\leq (n-f)\mathbb{E}\|G\|^r + \sum_{\text{byz } k} \mathbb{E}\|B_k\|^r \mathbb{1}(i_* = k).\end{aligned}$$

Denoting by C a generic constant, when $i_* = k$, we have for all correct indices i

$$\begin{aligned}\left\|B_k - \frac{1}{\delta_c(k)}\sum_{k\to\text{correct } j} V_j\right\| &\leq \sqrt{\frac{1}{\delta_c(k)}\sum_{i\to\text{ correct } j}\left\|V_i - V_j\right\|^2 + \frac{\delta_b(i)}{\delta_c(k)}\left\|V_i - V_{\zeta(i)}\right\|^2} \\ &\leq C\cdot\left(\sqrt{\frac{1}{\delta_c(k)}}\cdot\sum_{i\to\text{correct } j}\left\|V_i - V_j\right\| + \sqrt{\frac{\delta_b(i)}{\delta_c(k)}}\cdot\left\|V_i - V_{\zeta(i)}\right\|\right) \\ &\leq C\cdot\sum_{\text{correct } j}\left\|V_j\right\| \quad \text{(triangular inequality)}.\end{aligned}$$

The second inequality comes from the equivalence of norms in finite dimension. Now

$$\begin{aligned}\|B_k\| &\leq \left\|B_k - \frac{1}{\delta_c(k)}\sum_{k\to\text{correct } j} V_j\right\| + \left\|\frac{1}{\delta_c(k)}\sum_{k\to\text{correct } j} V_j\right\| \\ &\leq C\cdot\sum_{\text{correct } j}\left\|V_j\right\| \\ \|B_k\|^r &\leq C\cdot\sum_{r_1+\cdots+r_{n-f}=r}\|V_1\|^{r_1}\cdots\left\|V_{n-f}\right\|^{r_{n-f}}.\end{aligned}$$

Since the V_i's are independent, we finally obtain that $\mathbb{E}\|\mathrm{KR}\|^r$ is bounded above by a linear combination of terms of the form $\mathbb{E}\|V_1\|^{r_1}\cdots\mathbb{E}\left\|V_{n-f}\right\|^{r_{n-f}} = \mathbb{E}\|G\|^{r_1}\cdots\mathbb{E}\|G\|^{r_{n-f}}$ with $r_1+\cdots+r_{n-f} = r$. This completes the proof of condition *(ii)*. □

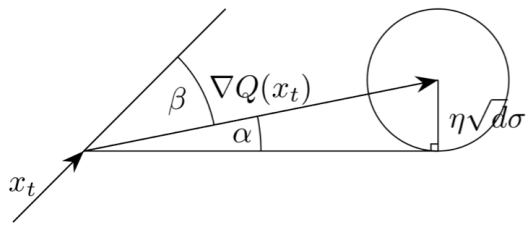


Figure 3.2 – Condition on the angles between x_t, $\nabla Q(x_t)$ and $\mathbb{E}\textsc{Kr}_t$, in the region $\|x_t\|^2 > D$.

3.3 Convergence Analysis

In this section, we analyze the convergence of the SGD using our Krum function defined in Section 3.2. The SGD equation is expressed as follows

$$x_{t+1} = x_t - \gamma_t \cdot \textsc{Kr}(V_1^t, \ldots, V_n^t)$$

where at least $n-f$ vectors among the V_i^t's are correct, while the other ones may be Byzantine. For a correct index i, $V_i^t = G(x_t, \xi_i^t)$ where G is the gradient estimator. We define the *local standard deviation* $\sigma(x)$ by

$$d \cdot \sigma^2(x) = \mathbb{E}\,\|G(x,\xi) - \nabla Q(x)\|^2 .$$

The following proposition considers an (*a priori*) non-convex cost function. In the context of non-convex optimization, even in the centralized case, it is generally hopeless to aim at proving that the parameter vector x_t tends to a local minimum. Many criteria may be used instead. We follow [23], and we prove that the parameter vector x_t almost surely reaches a "flat" region (where the norm of the gradient is small), in a sense explained below.

Proposition 2. *We assume that* (i) *the cost function Q is three times differentiable with continuous derivatives, and is non-negative, $Q(x) \geq 0$;* (ii) *the learning rates satisfy $\sum_t \gamma_t = \infty$ and $\sum_t \gamma_t^2 < \infty$;* (iii) *the gradient estimator satisfies $\mathbb{E}G(x,\xi) = \nabla Q(x)$ and $\forall r \in \{2,\ldots,4\}$, $\mathbb{E}\|G(x,\xi)\|^r \leq A_r + B_r\|x\|^r$ for some constants A_r, B_r;* (iv) *there exists a constant $0 \leq \alpha < \pi/2$ such that for all x*

$$\eta(n,f) \cdot \sqrt{d} \cdot \sigma(x) \leq \|\nabla Q(x)\| \cdot \sin\alpha;$$

(v) *finally, beyond a certain horizon, $\|x\|^2 \geq D$, there exist $\epsilon > 0$ and $0 \leq \beta < \pi/2 - \alpha$ such that $\|\nabla Q(x)\| \geq \epsilon > 0$ and $\frac{\langle x, \nabla Q(x)\rangle}{\|x\| \cdot \|\nabla Q(x)\|} \geq \cos\beta$. Then the sequence of gradients $\nabla Q(x_t)$ converges almost surely to zero.*

Conditions *(i)* to *(iv)* are the same conditions as in the non-convex convergence analysis in [23]. Condition *(v)* is a slightly stronger condition than the corresponding one in [23], and states that, beyond a certain horizon, the cost function Q is "convex enough", in the sense that the direction of the gradient is sufficiently close to the direction of the parameter vector x. Condition *(iv)*, however, states that the gradient estimator used by the correct workers has to be accurate

enough, i.e., the local standard deviation should be small relatively to the norm of the gradient. Of course, the norm of the gradient tends to zero near, e.g., extremal and saddle points. Actually, the ratio $\eta(n,f)\cdot\sqrt{d}\cdot\sigma/\|\nabla Q\|$ controls the maximum angle between the gradient ∇Q and the vector chosen by the Krum function. In the regions where $\|\nabla Q\| < \eta(n,f)\cdot\sqrt{d}\cdot\sigma$, the Byzantine workers may take advantage of the noise (measured by σ) in the gradient estimator G to bias the choice of the parameter server. Therefore, Proposition 2 is to be interpreted as follows: in the presence of Byzantine workers, the parameter vector x_t almost surely reaches a basin around points where the gradient is small ($\|\nabla Q\| \leq \eta(n,f)\cdot\sqrt{d}\cdot\sigma$), i.e., points where the cost landscape is "almost flat".

Note that the convergence analysis is based only on the fact that function KR is (α,f)-Byzantine resilient. We now give the complete proof of Proposition 2.

Proof. For the sake of simplicity, we write $\mathrm{KR}_t = \mathrm{KR}(V_1^t,\ldots,V_n^t)$. Before proving the main claim of the proposition, we first show that the sequence x_t is almost surely globally confined within the region $\|x\|^2 \leq D$.

Global confinement. Let $u_t = \phi(\|x_t\|^2)$ where

$$\phi(a) = \begin{cases} 0 & \text{if } a < D \\ (a-D)^2 & \text{otherwise} \end{cases}$$

Note that

$$\phi(b) - \phi(a) \leq (b-a)\phi'(a) + (b-a)^2. \tag{3.3}$$

This becomes an equality when $a, b \geq D$. Applying this inequality to $u_{t+1} - u_t$ yields

$$\begin{aligned} u_{t+1} - u_t \leq{}& \left(-2\gamma_t\langle x_t, \mathrm{KR}_t\rangle + \gamma_t^2\|\mathrm{KR}_t\|^2\right)\cdot\phi'(\|x_t\|^2) \\ &+ 4\gamma_t^2\langle x_t, \mathrm{KR}_t\rangle^2 - 4\gamma_t^3\langle x_t, \mathrm{KR}_t\rangle\|\mathrm{KR}_t\|^2 + \gamma_t^4\|\mathrm{KR}_t\|^4 \\ \leq{}& -2\gamma_t\langle x_t, \mathrm{KR}_t\rangle\phi'(\|x_t\|^2) + \gamma_t^2\|\mathrm{KR}_t\|^2\phi'(\|x_t\|^2) \\ &+ 4\gamma_t^2\|x_t\|^2\|\mathrm{KR}_t\|^2 + 4\gamma_t^3\|x_t\|\|\mathrm{KR}_t\|^3 + \gamma_t^4\|\mathrm{KR}_t\|^4. \end{aligned}$$

Let $\mathscr{P}_t$ denote the σ-algebra encoding all the information up to round t. Taking the conditional expectation with respect to $\mathscr{P}_t$ yields

$$\begin{aligned} \mathbb{E}(u_{t+1} - u_t|\mathscr{P}_t) \leq{}& -2\gamma_t\langle x_t, \mathbb{E}\mathrm{KR}_t\rangle + \gamma_t^2\mathbb{E}\left(\|\mathrm{KR}_t\|^2\right)\phi'(\|x_t\|^2) \\ &+ 4\gamma_t^2\|x_t\|^2\mathbb{E}\left(\|\mathrm{KR}_t\|^2\right) + 4\gamma_t^3\|x_t\|\mathbb{E}\left(\|\mathrm{KR}_t\|^3\right) + \gamma_t^4\mathbb{E}\left(\|\mathrm{KR}_t\|^4\right). \end{aligned}$$

Thanks to condition *(ii)* of (α,f)-Byzantine resilience, and the assumption on the first four moments of G, there exist positive constants A_0, B_0 such that

$$\mathbb{E}(u_{t+1} - u_t|\mathscr{P}_t) \leq -2\gamma_t\langle x_t, \mathbb{E}\mathrm{KR}_t\rangle\phi'(\|x_t\|^2) + \gamma_t^2\left(A_0 + B_0\|x_t\|^4\right).$$

Thus, there exist positive constant A, B such that

$$\mathbb{E}\left(u_{t+1} - u_t | \mathscr{P}_t\right) \leq -2\gamma_t \langle x_t, \mathbb{E}\mathrm{KR}_t \rangle \phi'(\|x_t\|^2) + \gamma_t^2 \left(A + B \cdot u_t\right).$$

When $\|x_t\|^2 < D$, the first term of the right hand side is null because $\phi'(\|x_t\|^2) = 0$. When $\|x_t\|^2 \geq D$, this first term is negative because (see Figure 3.2)

$$\langle x_t, \mathbb{E}\mathrm{KR}_t \rangle \geq \|x_t\| \cdot \|\mathbb{E}\mathrm{KR}_t\| \cdot \cos(\alpha + \beta) > 0.$$

Hence

$$\mathbb{E}\left(u_{t+1} - u_t | \mathscr{P}_t\right) \leq \gamma_t^2 \left(A + B \cdot u_t\right).$$

We define two auxiliary sequences

$$\begin{aligned} \mu_t &= \prod_{i=1}^{t} \frac{1}{1 - \gamma_i^2 B} \xrightarrow[t \to \infty]{} \mu_\infty \\ u'_t &= \mu_t u_t. \end{aligned}$$

Note that the sequence μ_t converges because $\sum_t \gamma_t^2 < \infty$. Then

$$\mathbb{E}\left(u'_{t+1} - u'_t | \mathscr{P}_t\right) \leq \gamma_t^2 \mu_t A.$$

Consider the indicator of the positive variations of the left-hand side

$$\chi_t = \begin{cases} 1 & \text{if } \mathbb{E}\left(u'_{t+1} - u'_t | \mathscr{P}_t\right) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Then

$$\mathbb{E}\left(\chi_t \cdot (u'_{t+1} - u'_t)\right) \leq \mathbb{E}\left(\chi_t \cdot \mathbb{E}\left(u'_{t+1} - u'_t | \mathscr{P}_t\right)\right) \leq \gamma_t^2 \mu_t A.$$

The right-hand side of the previous inequality is the summand of a convergent series. By the quasi-martingale convergence theorem [130], this shows that the sequence u'_t converges almost surely, which in turn shows that the sequence u_t converges almost surely, $u_t \to u_\infty \geq 0$.

Let us assume that $u_\infty > 0$. When t is large enough, this implies that $\|x_t\|^2$ and $\|x_{t+1}\|^2$ are greater than D. Inequality 3.3 becomes an equality, which implies that the following infinite sum converges almost surely

$$\sum_{t=1}^{\infty} \gamma_t \langle x_t, \mathbb{E}\mathrm{KR}_t \rangle \phi'(\|x_t\|^2) < \infty.$$

Note that the sequence $\phi'(\|x_t\|^2)$ converges to a positive value. In the region $\|x_t\|^2 > D$, we have

$$\begin{aligned} \langle x_t, \mathbb{E}\mathrm{KR}_t \rangle &\geq \sqrt{D} \cdot \|\mathbb{E}\mathrm{KR}_t\| \cdot \cos(\alpha + \beta) \\ &\geq \sqrt{D} \cdot \Big(\|\nabla Q(x_t)\| - \eta(n, f) \cdot \sqrt{d} \cdot \sigma(x_t)\Big) \cdot \cos(\alpha + \beta) \\ &\geq \sqrt{D} \cdot \epsilon \cdot (1 - \sin\alpha) \cdot \cos(\alpha + \beta) > 0. \end{aligned}$$

This contradicts the fact that $\sum_{t=1}^{\infty} \gamma_t = \infty$. Therefore, the sequence u_t converges to zero. This convergence implies that the sequence $\|x_t\|^2$ is bounded, i.e., the vector x_t is confined in a bounded region containing the origin. As a consequence, any continuous function of x_t is also bounded, such as, e.g., $\|x_t\|^2$, $\mathbb{E}\|G(x_t,\xi)\|^2$ and all the derivatives of the cost function $Q(x_t)$. In the sequel, positive constants K_1, K_2, etc... are introduced whenever such a bound is used.

Convergence. We proceed to show that the gradient $\nabla Q(x_t)$ converges almost surely to zero. We define

$$h_t = Q(x_t).$$

Using a first-order Taylor expansion and bounding the second derivative with K_1, we obtain

$$\left| h_{t+1} - h_t + 2\gamma_t \langle \mathrm{KR}_t, \nabla Q(x_t) \rangle \right| \leq \gamma_t^2 \|\mathrm{KR}_t\|^2 K_1 \text{ a.s.}$$

Therefore

$$\mathbb{E}\left(h_{t+1} - h_t | \mathscr{P}_t\right) \leq -2\gamma_t \langle \mathbb{E}\mathrm{KR}_t, \nabla Q(x_t) \rangle + \gamma_t^2 \mathbb{E}\left(\|\mathrm{KR}_t\|^2 | \mathscr{P}_t\right) K_1. \tag{3.4}$$

By the properties of (α, f)-Byzantine resiliency, this implies

$$\mathbb{E}\left(h_{t+1} - h_t | \mathscr{P}_t\right) \leq \gamma_t^2 K_2 K_1,$$

which in turn implies that the positive variations of h_t are also bounded

$$\mathbb{E}\left(\chi_t \cdot (h_{t+1} - h_t)\right) \leq \gamma_t^2 K_2 K_1.$$

The right-hand side is the summand of a convergent infinite sum. By the quasi-martingale convergence theorem, the sequence h_t converges almost surely, $Q(x_t) \to Q_\infty$.

Taking the expectation of Inequality 3.4, and summing on $t = 1, \ldots, \infty$, the convergence of $Q(x_t)$ implies that

$$\sum_{t=1}^{\infty} \gamma_t \langle \mathbb{E}\mathrm{KR}_t, \nabla Q(x_t) \rangle < \infty \text{ a.s.}$$

We now define

$$\rho_t = \|\nabla Q(x_t)\|^2.$$

Using a Taylor expansion, as demonstrated for the variations of h_t, we obtain

$$\rho_{t+1} - \rho_t \leq -2\gamma_t \langle \mathrm{KR}_t, \left(\nabla^2 Q(x_t)\right) \cdot \nabla Q(x_t) \rangle + \gamma_t^2 \|\mathrm{KR}_t\|^2 K_3 \text{ a.s.}$$

Taking the conditional expectation, and bounding the second derivatives by K_4,

$$\mathbb{E}\left(\rho_{t+1} - \rho_t | \mathscr{P}_t\right) \leq 2\gamma_t \langle \mathbb{E}\mathrm{KR}_t, \nabla Q(x_t) \rangle K_4 + \gamma_t^2 K_2 K_3.$$

The positive expected variations of ρ_t are bounded

$$\mathbb{E}\left(\chi_t \cdot \left(\rho_{t+1} - \rho_t\right)\right) \leq 2\gamma_t \mathbb{E}\langle \mathbb{E}\mathrm{KR}_t, \nabla Q(x_t)\rangle K_4 + \gamma_t^2 K_2 K_3.$$

The two terms on the right-hand side are the summands of convergent infinite series. By the quasi-martingale convergence theorem, this shows that ρ_t converges almost surely.

We have

$$\begin{aligned}\langle \mathbb{E}\mathrm{KR}_t, \nabla Q(x_t)\rangle &\geq \left(\|\nabla Q(x_t)\| - \eta(n,f)\cdot\sqrt{d}\cdot\sigma(x_t)\right)\cdot\|\nabla Q(x_t)\| \\ &\geq \underbrace{(1-\sin\alpha)}_{>0}\cdot\rho_t.\end{aligned}$$

This implies that the following infinite series converge almost surely

$$\sum_{t=1}^{\infty} \gamma_t \cdot \rho_t < \infty.$$

Since ρ_t converges almost surely, and the series $\sum_{t=1}^{\infty}\gamma_t = \infty$ diverges, we conclude that the sequence $\|\nabla Q(x_t)\|$ converges almost surely to zero. □

3.4 Experimental Evaluation

We report here on the evaluation of the convergence and resilience properties of Krum, as well as an optimized variant of it. We also discuss other variants of Krum.

We evaluate our algorithm on a distributed framework where we set some nodes to have an adversarial behavior of two kinds: *(a) The omniscient Byzantine workers:* workers have access to all the training-set (as if they breached into the other workers share of data). Those workers compute a rather precise estimator of the true gradient, and send the opposite value multiplied by an arbitrarily large factor. *(b) The Gaussian Byzantine workers:* Byzantine workers do not compute an estimator of the gradient and send a random vector, drawn from a Gaussian distribution of which we could set the variance high enough (200) to break averaging strategies.

On this distributed framework, we train two models with non-trivial (a-priori non-Convex) loss functions: a 4-layer convolutional network (ConvNet) with a final fully connected layer, and a classical multilayer perceptron (MLP) with two hidden layers, and on two tasks: spam filtering and image classification. We use cross-validation accuracy to compare the performance of different algorithms. The focus is on the Byzantine resilience of the gradient aggregation rules and not on the performance of the models per se.

Resilience to Byzantine processes. We consider the task of spam filtering (dataset *spambase* [110]). The learning model is a multi-layer perceptron (MLP) with two hidden layers. There are $n = 20$ worker processes. Byzantine processes propose vectors drawn from a Gaussian distribution with

mean zero, and isotropic covariance matrix with standard deviation 200. We refer to this behavior as *Gaussian Byzantine*. Each (correct) worker estimates the gradient on a mini-batch of size 3. We measure the error using cross-validation. Figure 3.3 shows how the error (y-axis) evolves with the number of rounds (x-axis).

In the first plot (left), there are no Byzantine workers. Unsurprisingly, averaging converges faster than Krum. In the second plot (right), 33% of the workers are Gaussian Byzantine. In this case, averaging does not converge at all, whereas Krum behaves as if there were no Byzantine workers. This experiment confirms that averaging does not tolerate (the rather mild) Gaussian Byzantine behavior, whereas Krum does.

The Cost of Resilience. As seen above, Krum slows down learning when there are no Byzantine workers. The following experiment shows that this overhead can be significantly reduced by slightly increasing the mini-batch size. To highlight the effect of the presence of Byzantine workers, the Byzantine behavior has been set as follows: each Byzantine worker computes an estimate of the gradient over the *whole* dataset (yielding a very accurate estimate of the gradient), and proposes the opposite vector, scaled to a large length. We refer to this behavior as *omniscient*.

Figure 3.4 displays how the error value at the 500-th round (y-axis) evolves when the mini-batch size varies (x-axis). In this experiment, we consider the tasks of spam filtering (dataset *spambase*) and image classification (dataset *MNIST*). The MLP model is used in both cases. Each curve is obtained with either 0 or 45% of omniscient Byzantine workers.

In all cases, averaging still does not tolerate Byzantine workers, but yields the lowest error when there are no Byzantine workers. However, once the size of the mini-batch reaches the value 20, Krum with 45% omniscient Byzantine workers is as accurate as averaging with 0% Byzantine workers. We observe a similar pattern for a ConvNet as provided in the supplementary material.

Multi-Krum. For the sake of presentation simplicity, we considered a version of Krum which selects only one vector among the vector proposed by the workers. We also propose a variant of Krum, we call *Multi-Krum*. Multi-Krum computes, for each vector proposed, the score as in the Krum function. Then, Multi-Krum selects the $m \in \{1, \ldots, n\}$ vectors $V_1^*, \ldots, V_m^*$ which score the best, and outputs their average $\frac{1}{m}\sum_i V_i^*$. Note that, the cases $m = 1$ and $m = n$ correspond to Krum and averaging respectively.

Figure 3.5 shows how the error (y-axis) evolves with the number of rounds (x-axis). In the figure, we consider the task of spam filtering (dataset *spambase*), and the MLP model (the same comparison is done for the task of image classification with a ConvNet and is provided in the supplementary material). The Multi-Krum parameter m is set to $m = n - f$. Figure 3.5 shows that Multi-Krum with 33% Byzantine workers is as efficient as averaging with 0% Byzantine workers.

From the practitionner's perspective, the parameter m may be used to set a specific trade-off between convergence speed and resilience to Byzantine workers.

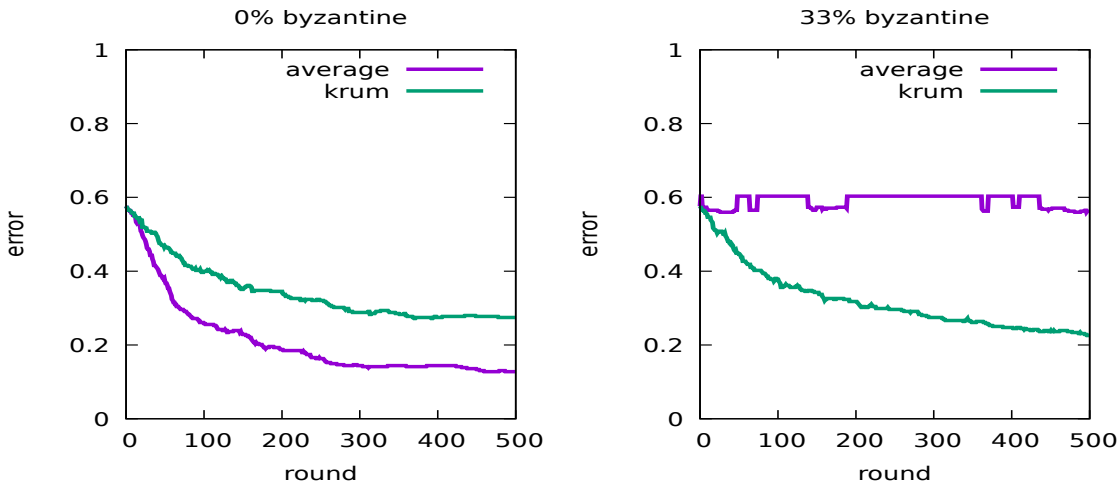


Figure 3.3 – Cross-validation error evolution with rounds, respectively in the absence and in the presence of 33% Byzantine workers. The mini-batch size is 3. With 0% Gaussian Byzantine workers, averaging converges faster than Krum. With 33% Gaussian Byzantine workers, averaging does not converge, whereas Krum behaves as if there were 0% Byzantine workers.

3.5 Beyond Krum

So far, a part from Krum, we presented the strongest variant of Krum: the Multi-Krum aggregation rule. We refer to this aggregation rule as *mKrum* in the following. In this section we present the other aggregation rules that we tested.

- The Medoid.

  This aggregation rule is an easily computable variant of the geometric median. The geometric median is known to have strong statistical robustness, however there exists no algorithm yet [39] to compute its exact value [7]. Recall that the geometric median of a set of d-dimensional vectors $V_1, \ldots, V_n$ is defined as follows:

  $$med(V_1, \ldots, V_n) = \arg\min_{x \in \mathbb{R}^d} \sum_{i=1}^{n} \| V_i - x \|$$

  The geometric median does not necessarily lie among the vectors $V_1, \ldots, V_n$. A computable alternative to the median are the medoids, which are defined as follows:

  $$medoids(V_1, \ldots, V_n) = \arg\min_{x \in \{V_1, \ldots, V_n\}} \sum_{i=1}^{n} \| V_i - x \|.$$

  A medoid is not unique, similarly to Krum, if more than a vector minimizes the sum, we will refer to the *Medoid* as the medoid with the smallest index.

[7] The computable approximate ϵ-median [39] introduces a new parameter (ϵ) that should be studied with respect to the risk of biasing the gradient estimator.

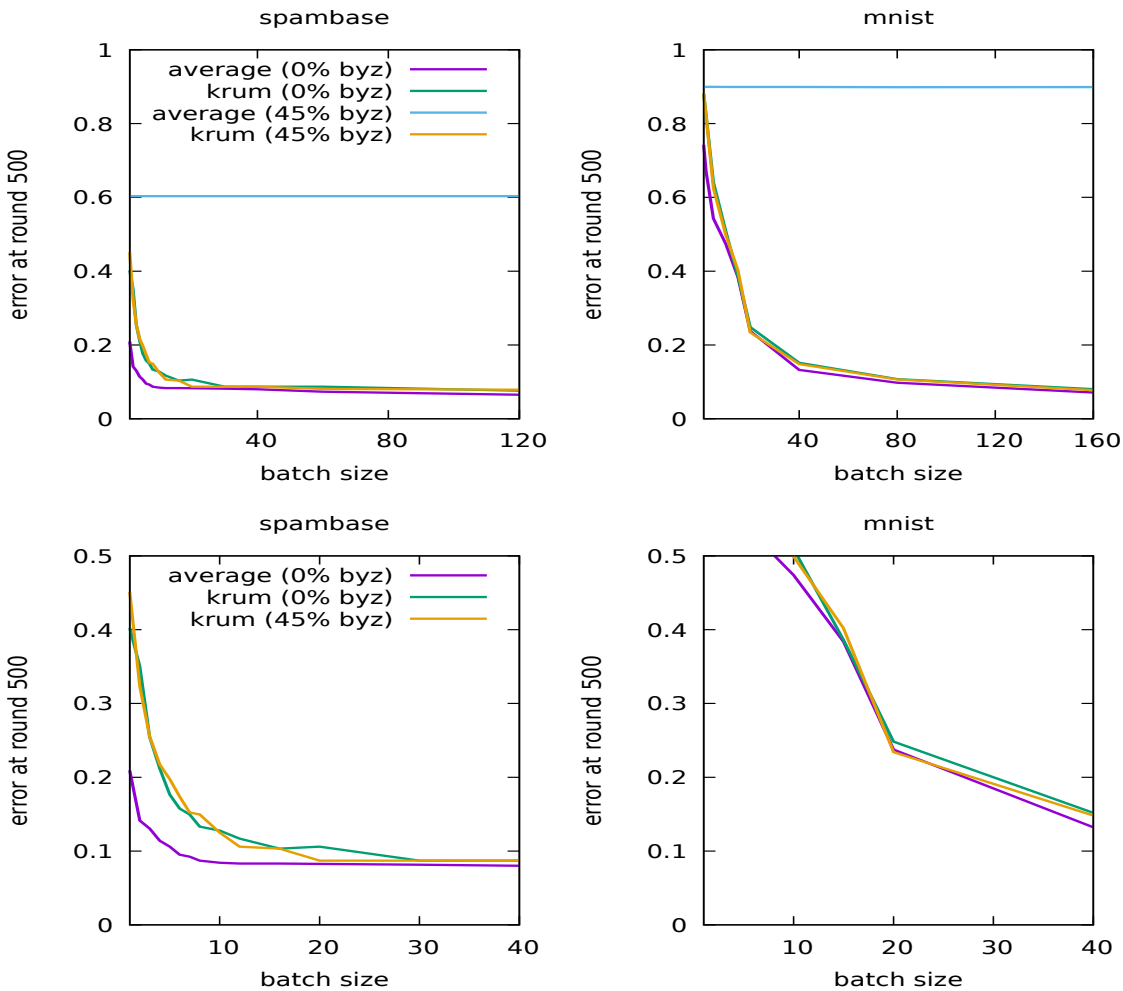


Figure 3.4 – Cross-validation error at round 500 when increasing the mini-batch size. The two figures on the rights are zoomed versions of the two on the left). With a reasonably large mini-batch size (arround 10 for MNIST and 30 for Spambase), Krum with 45% omniscient Byzantine workers is as accurate as averaging with 0% Byzantine workers.

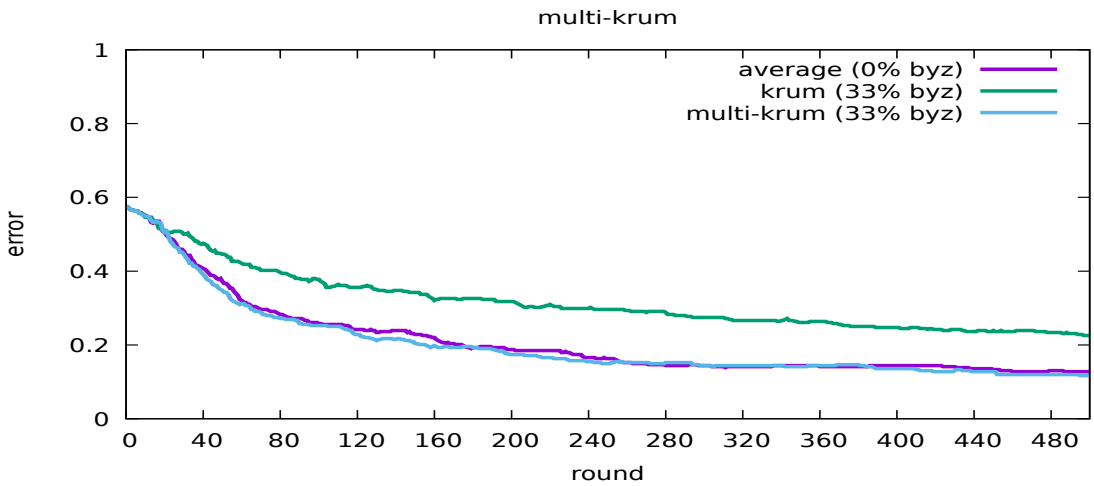


Figure 3.5 – Cross-validation error evolution with rounds. The mini-batch size is 3. Multi-Krum with 33% Gaussian Byzantine workers converges as fast as averaging with 0% Byzantine workers.

- $1-p$ Krum.

  In this aggregation rule, the parameter server chooses the average of the proposed vectors with probability p, and Krum with probability $1-p$. Moreover, we choose p to depend on the learning round. In our implementation $p_t = \frac{1}{\sqrt{t}}$, where t is the round number. With such a probability, and despite the presence of Byzantine workers, $1-p$ Krum has a similar proof of convergence as Krum: the probability of choosing Krum goes to 1 when $t \mapsto \infty$. The rational is to follow averaging in the early phases, to accelerate learning in the absence of Byzantine workers, while mostly following Krum in the later phases and guarantee Byzantine resilience [8].

Replacing an MLP by a ConvNet. In addition to what have been presented above, we see from Figure A.1 that, similarly to the situation on an MLP, mKrum is, despite attacks, comparable to a non-attacked averaging. In the same veine, in Figure A.2, we observe that like for an MLP, the ConvNet only requires a reasonably low batch size for Krum to perform (despite 45 % Byzantine workers) as good as a non-attacked averaging.

Optimizing Krum. In Figure A.3 we compare the different variants in the absence of Byzantine workers, we see that Multi-Krum is comparably fast to averaging, then comes 1-p Krum, while Krum and the Medoid are slower.

In the presence of Byzantine workers (Figure A.4), Krum, Medoid and 1-p Krum are similarly robust. Unsurprisingly, averaging is not resilient (no improvement over time). Multi-Krum

[8] Remember that the parameter server never knows if there are Byzantine workers or not. The latter can behave like correct workers in the beginning and fool any fraud detection measure.

outperforms all the tested aggregation rules.

3.6 Concluding Remarks

The Distributed Computing Perspective. Although seemingly related, results in d-dimensional approximate agreement [121, 78] cannot be applied to our Byzantine-resilient machine context. These forms of agreements would be tempting to use, as they guarantee that the vector which is agreed on lies in the convex hull of the proposed vectors from correct processes. In particular, this would satisfy the key intuition behind our solution, which is that the decided update points to the same half-space as the correct gradients. In [122], Mendes, Herlihy, Vaidya and Garg showed that these forms of agreement require a number of correct processes that grows as the product of the number of Byzantine ones multiplied by the dimension ($\Omega(f.d)$). Another reason why their results are reasons *not* to look into the usual agreement toolbox, in [121], Mendes and Herlihy proved that in asynchronous systems, these forms of agreements require a local computation by each worker that is in $O(n^d)$. While this cost seems reasonable for small dimensions, such as, e.g., mobile robots meeting in a $2D$ or $3D$ space, it becomes a real issue in the context of machine learning, where d may be as high as 160 billion [163] (making d a crucial parameter when considering complexities, either for local computations, or for communication rounds).

With our relaxation of what should the workers agree on, the expected time complexity of the Krum function is $O(n^2 \cdot d)$, which is only linear in d. The cost of our asynchronous Byzantine SGD is also only linear in d as we see later in this thesis.

A closer approach to ours has been recently proposed in [157, 158]. In [157], the study only deals with parameter vectors of dimension one, which is too restrictive for today's multi-dimensional machine learning. In [158], the authors tackle a multi-dimensional situation, using an iterated approximate Byzantine agreement that reaches consensus asymptotically. This is however only achieved on a finite set of possible environmental states and cannot be used in the continuous context of stochastic gradient descent.

The Statistics and Machine Learning View. Our work looks at the resilience of the aggregation rule using ideas that are close to those of [65], and somehow classical in theoretical statistics on the robustness of the geometric median and the notion of breakdown [47]. The closest concept to a breakdown in our work is the maximum fraction of Byzantine workers that can be tolerated, i.e. $\frac{n-2}{2n}$, which reaches the optimal theoretical value (1/2) asymptotically on n. It is known that the geometric median does achieve the optimal breakdown. However, no closed form nor an exact algorithm to compute the geometric median is known (only approximations are available [39] and their Byzantine resilience is an open problem.). An easily computable variant of the median is the *Medoid*, which is the proposed vector minimizing the sum of distances to all other proposed vectors. The Medoid can be computed with a similar algorithm to Krum. We show however in the supplementary material that the implementation of the Medoid is outperformed by multi-Krum.

4 Bulyan: When Convergence is Not Enough

4.1 Introduction

Since the publication of our work on Krum [21], an emerging line of research (of about seventy papers[1] over the past two years) looks at robustness through the lenses of (distributed) optimization. In all of these works, as we saw in the previous chapter, SGD can be proven to converge despite the presence of a (bounded) number of adversaries. The general recipe is to find a robust estimator for the gradient in the presence of adversaries, and to prefer that estimator over a mere linear combination of the provided gradients [44, 181], known for not being robust [145].

Accordingly, Byzantine–resilient aggregation rules were derived and proved to guarantee the convergence of SGD in this context [31, 21]. **But is convergence enough** in the non–convex, high dimensional case of neural networks?

In fact, the question of SGD convergence in the case of neural networks is neither new and unprecedented nor old and forgotten by the ML community. Two years ago, the convergence question sparked a hot debate when some, using timely examples, were blaming SGD to be *"brittle"*[2]. Others moderated that view, reminding us how blaming neural networks for their lack of (provable) convergence led the community to *"threw the baby with the bathwater back in the 1990s"*[3]; which arguably slowed research progress in the understanding of neural networks as a learning machine.

We show in this chapter that **convergence is not enough**. We look at the question from the robust distributed optimization point of view, where some workers can be Byzantine and we show that, whilst indeed neural networks usually benefit from the existence of many "similarly good" local minima [34], making SGD convergence a preferable requirement is clearly not sufficient in a Byzantine distributed setting. More precisely, we show that a single Byzantine worker can make any known Byzantine-resilient aggregation rule for SGD learn *ineffectual* models. It is

[1] Some of which we discuss in the final part of this thesis.

[2] Ali Rahimi, Test of Time Award Lecture. NIPS 2017.

[3] Yann LeCun. *"My Take on Ali Rahimi's Test of Time Award Talk at NIPS 2017"*. https://www2.isye.gatech.edu/~tzhao80/Yann_Response.pdf

important to note that we do not contradict the proof of convergence of these rules. We rather take advantage of the high dimensional and highly non–convex *landscape* of the loss function to make these rules *converge*, as they were proven to, but to *ineffectual* models. We provide an analytic understanding of how attackers could benefit from this *curse of dimensionality*, and propose a solution that enhances the Byzantine–resilience of SGD.

To start feeling the vulnerability, consider any Byzantine–resilient gradient aggregation rule based on a distance criteria. When the distance criteria relies solely on norms in the ℓ_p categories, with a *small* p (the Euclidean or ℓ_1 norms), a Byzantine gradient can both remain close to honest gradients and have one of its coordinates *poisoned*, e.g. set to a large value. These poisoned coordinates can take values on the order of $\Omega(\sqrt[p]{d})$, a large order given the dimension d of modern neural networks. And the gradient would still be seen as "legitimate" by the aggregation rule. Inversely, when the distance criteria involves norms closer to the infinite norm, the Byzantine worker can poison every coordinates while being exactly on the value that the aggregation rule will decide on. This way, the Byzantine worker can drive the aggregation to *converge*, but to the worst possible sub–optimum that the $\Omega(\sqrt[p]{d})$ (ℓ_p norm) or the $\Omega(d)$ (infinite norm) margins enable it to get. See section 4.3.1.

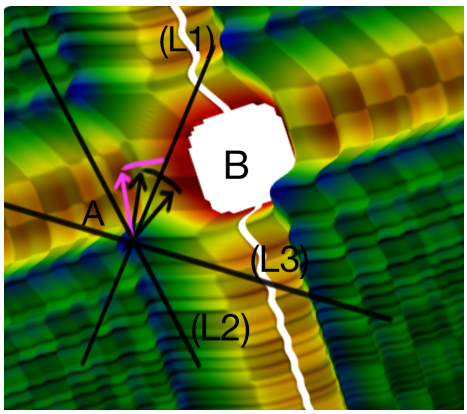


Figure 4.1 – In a non-convex situation, two correct vectors (black arrows) are pointing towards the deep optimum located in area B, both vectors belong to the plane formed by lines L1 and L2. A Byzantine worker (magenta) is taking benefit from the third dimension, and the non-convex landscape, to place a vector that is heading towards one of the bad local optimums of area A. This Byzantine vector is located in the plane (L1,L3). Due to the variance of the correct workers on the plane (L1,L2), the Byzantine one has a budget of about $\sqrt{3}$ times the disagreement of the correct workers, to put as a deviation towards A, on the line (L3), while still being selected by a weak Byzantine resilient GAR, since its projection on the plane (L1,L2) lies exactly on the line (L1), unlike that of the correct workers. In very high dimensions, the situation is amplified by $\sqrt{d}$.

Given any Byzantine–resilient rule $\mathscr{A}$, we propose a generic enhancement recipe we call *Bulyan of* $\mathscr{A}$, or simply *Bulyan*($\mathscr{A}$), that improves Byzantine–resilience in the following sense: if the vectors selected by $\mathscr{A}$ are in a *cone* of angle α around the true gradient, then the vectors selected by *Bulyan*($\mathscr{A}$) are in a *cone* of angle $\alpha' \leq \alpha$. Most importantly, we prove that *Bulyan*($\mathscr{A}$) drastically

reduces the *leeway* of Byzantine workers to a narrow $O(\frac{1}{\sqrt{d}})$ bound.

We also empirically evaluate the trade–offs induced by Bulyan, and compare its convergence speed with those of other aggregation rules. In particular, we show that with typically used values for the batch size, Bulyan is comparable to the speed of averaging (the fastest aggregation rule), which does not stand a single Byzantine worker.

The rest of the chapter is organized as follows. Section 4.2 briefly recalls the model of distributed SGD with Byzantine workers as introduced in Chapter 2, together with recalling a few Byzantine–resilient gradient aggregation rules that we need for Bulyan. Section 4.3 introduces our attack and analytically discusses how it affects the aforementioned gradient aggregation rules. Section 4.4 describes our algorithm, Bulyan, and proves our two main theoretical results. Section 4.5 highlights the practical impacts of our attack, and the beneficial effects of Bulyan on MNIST and CIFAR–10. Finally, Section 4.6 concludes with a few remarks.

4.2 Model for Bulyan

4.2.1 Distributed Stochastic Gradient Descent (DSGD)

As in the previous chapter, we follow the usual *parameter server* model used in distributed implementations of machine learning [44, 105, 106]. The system consists in $n+1$ processes: 1 *master* and n *workers*. There are $f \leq n$ *Byzantine*, i.e. adversarial, workers. Their role and capability are described in Section 4.2.2.

Let t be the current epoch, and x_t be the model parameters, let Q be the cost function we aim to minimize. Each honest worker $i \in \{1, \ldots, n-f\}$ draws i.i.d. (mini–batches of) samples ξ_i^t from the data set to compute an estimate $V_i^t = G(x_t, \xi_i^t)$ of the gradient $\nabla Q(x_t)$.

We assume that ∇Q is K–Lipschtiz and that Q is three–times differentiable. We also assume that G has a bounded variance σ, i.e. $\mathbb{E}_\xi \|G(x,\xi) - \nabla Q(x)\|^2 \leq \sigma^2$, and as assumed in [23, 21], for $r \in \{1, \ldots, 4\}$, the r–th statistical moments of G does not overgrow the r–th powers of the model:

$$\exists (A_r, B_r) \in (\mathbb{R}_+)^2, \mathbb{E}_\xi \|G(x,\xi)\|^r \leq A_r + B_r \|x\|^r$$

4.2.2 Adversary

The *adversary* in this model is an entity which controls f of the n workers. These adversarial workers, and the gradients they send, are called *Byzantine*. The goal of the adversary is to prevent the distributed SGD process from converging to a *satisfying state*. Ideally, a satisfying state with Byzantine workers should achieve an accuracy that is *comparable to* the one achieved without any Byzantine worker, all other things being equal; e.g. see Figure 4.4.

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 master (data and code)
- the *full* state of every worker (data and code)
- any data exchanged over any communication channel

Hence the adversary can leverage its knowledge of the master's state and the submitted gradients to build powerful attacks, as shown in sections 4.3.2 and 4.3.3. However, the adversary is not omnipotent. It cannot *directly* modify the state of the system, impersonate other workers, or delay communications. The adversary is only allowed to submit gradients via its f workers.

4.2.3 Gradient Aggregation Rules (GARs)

In this section, we describe the three studied *gradient aggregation rule*. A GAR, such as Krum from the previous chapter, transforms the workers' submitted gradients into a single, aggregated one. This aggregated gradient is then used to update the model parameters.

A very commonly used aggregation rule is *averaging* [1], which has several variants [181] that are also linear combinations. Yet linear combinations all have one major *flaw*: they give the adversary, described in Section 4.2.2, full control of the aggregated gradient (and hence of the model parameters), as proven in [21].

Contrary to linear combinations, the GARs studied here are all proven *Byzantine-resilient* in the sense of Definition 1 introduced in [21].

Brute

The *Brute* GAR requires that $n \geq 2f+1$. Informally, it 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 [145], and proven to have the optimal *breakdown point* of 50%.

Formally, let $(n, f) \in \mathbb{N}^2$ with $n \geq 2f+1$, let $(V_1 \dots V_{n-f}) \in (\mathbb{R}^d)^{n-f}$ be independent, identically distributed random vectors, with $V_i \sim \mathscr{G}$ and $\mathbb{E}[\mathscr{G}] = G$, let $(B_1 \dots B_f) \in (\mathbb{R}^d)^f$ be random vectors, possibly dependent between them and the vectors $(V_1 \dots V_{n-f})$, let $\mathscr{Q} = \{V_1 \dots V_n\}$ be the set of submitted gradients, let $\mathscr{R} = \{\mathscr{X} \mid \mathscr{X} \subset \mathscr{Q}, |\mathscr{X}| = n-f\}$ be the set of all the subsets of $\mathscr{Q}$ with a cardinality of $n-f$, and let $\mathscr{S} = \underset{\mathscr{X} \in \mathscr{R}}{\arg\min}\left(\max_{(V_i,V_j)\in\mathscr{X}^2}\left(\|V_i - V_j\|_p\right)\right)$.
Then, the aggregated gradient is given by $F = \frac{1}{n-f}\sum_{G\in\mathscr{S}} G$.

Since we use *Brute* as a benchmark when experimenting with small amount of workers, in Section 4.5.2, we also prove its (α, f)–Byzantine–resilience. The full proof is rather straightforward and is available in the Appendix.

As a side note, this rule can hardly be used in practical cases, as $|\mathscr{R}| = \frac{n!}{f!(n-f)!}$. For instance,

with $n = 57$ workers and $f = 27$, we have $|\mathscr{R}| \approx 1.4 \cdot 10^{16}$. Even with 10^9 measured subsets $\mathscr{X}$ per second, aggregating these 57 gradients would take more than 5 months.

Krum

The *Krum* GAR requires that $n \geq 2f + 3$. It was defined in the previous chapter as follows. Let $(V_1, \ldots, V_n)$ be the $n \geq 2f + 3$ received gradient, among which at most f are Byzantine gradients. For $i \neq j$, let $i \to j$ denote the fact that V_j belongs to the $n - f - 2$ closest vectors to V_i. Finally, let $s(V_i) = \sum_{i \to j} \left\| V_i - V_j \right\|_p^2$ be the *score* of V_i. Then, Krum outputs the vector V_k with the lowest score. (Recall that the (α, f)–Byzantine–resilience of Krum was proven in the previous chapter and in [21].)

The Geometric Median(s) - GeoMed

The third and last gradient aggregation rule we consider is the geometric median [145]. The exact geometric median is known to suffer from computational issues but can be approximated [39]. While we know from [145] that the Median has an optimal breakdown of 0.5, i.e, $n \geq 2f + 1$, it is not known however what would be α such that the Median is (α, f)-Byzantine-resilient. Empirically [21] and theoretically [31], variants of the Median were considered as GAR candidates. In particular, the Medoid, which is any minimizer *among* the proposed vectors of the sum of distances, can be used as a GAR and is easier to compute. Since there can be many minimizers, we will simply call GeoMed the Medoid of the proposed vectors with the smallest index.

4.3 Effective Attack on ℓ_p norm–based GARs

In this section, we describe a simple, yet effective attack passing the Byzantine–resilient GARs, such as the ones presented in Section 4.2.3. Actually, any ℓ_p-norm based GAR, where the chosen vector is the result of a distance minimization scheme, is affected.

4.3.1 Intuition

In high dimensions, the distance function, between two vectors $\|X - Y\|_p$, 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 the gradient estimation. Such "errors" are often *beneficial*, as they may allow the descent process to leave sub–optimal local minima [25]. In Byzantine–free distributed setups, gradient estimations *"disagree" **a bit** on each coordinate*[4].

In a vector space of dimension $d \gg 1$, the "bit of disagreement" on each coordinate translates into a distance $\|X - Y\|_p = O(\sqrt[p]{d})$. For the omniscient adversary described in Section 4.2.2, it

[4] This has been observed during the experiments.

translates into an opportunity to submit f Byzantine gradients that "disagree" **a lot**, as much as $O(\sqrt[p]{d})$, on only one coordinate with at least one non–Byzantine gradient. As the ℓ_p norm cannot answer the core question mentioned in the above paragraph, such Byzantine gradient could then be *selected* by a ℓ_p norm–based GAR.

Each gradient aggregation rule presented in Section 4.2.3 performs a linear combination of the selected gradient(s). Thus the final aggregated gradient might have one unexpectedly high coordinate. Depending on the learning rate (Figure 4.4), 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 4.5.2 clearly show this dependency on the learning rate, and indicate that this sub–space only offers sub–optimal to utterly *ineffective* models.

4.3.2 Attack on the Finite Norm, $p \geq 1$

The adversary defined in Section 4.2.2 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 master receives. Based on that capability, for each round, the adversary waits for $n-f$ non–Byzantine gradients to be received. Then it attacks.

Formally, let $\mathscr{Q} = \left\{V_1 \dots V_{n-f}\right\}$ be the set of submitted, non–Byzantine gradients, with $\forall i \in \left[1..n-f\right], V_i \in \mathbb{R}^d$.
Let $E = (0\dots0,1,0\dots0) \in \mathbb{R}^d$ be any coordinate to attack, and let $\mathscr{B}\left(\gamma\right) = \frac{1}{n-f}\left(\sum_{V\in\mathscr{Q}} V\right) + \gamma E$. By a simple linear regression, we estimate the highest value of γ, noted γ_m, such that $B = \mathscr{B}\left(\gamma_m\right)$ is selected by the aggregation rule. Finally, B is submitted by every Byzantine worker.

For each presented GAR, we reveal a 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 Brute and $n = 2f+3$ for Krum/GeoMed. The full details of the approximations are available in Appendix. We denote by $\bar{\delta}$ the average *folded* standard deviation on each coordinate of $\mathscr{G}$, and with p, q constants: $\gamma_m = O(\bar{\delta}\sqrt[p]{d})$ for Brute, and $\gamma_m = O(\bar{\delta}\sqrt[q]{f}\sqrt[p]{d})$ for Krum/GeoMed. The added dependence in $\sqrt[q]{f}$ for Krum/GeoMed comes from the *shape* of the score function, which naturally decreases the score of the Byzantine gradients as they all are identical.

As a side–note, an adversary does not necessarily need to know the submitted, non–Byzantine gradients $\mathscr{Q}$ with this attack. Indeed non–Byzantine gradients are assumed to be unbiased, so by the law of large numbers we have: $\lim_{|\mathscr{Q}|\to+\infty} \mathscr{B}\left(\gamma\right) = \mathbb{E}[\mathscr{G}] + \gamma E$. 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 γE to it.

4.3.3 Attack on the Infinite Norm

In the previous subsection, we have seen that γ_m vary as $\sqrt[p]{d}$. Yet, with $d \gg 1$ fixed: $\lim_{p\to+\infty} \sqrt[p]{d} = 1$. Basically, the *curse of dimensionality* exploited in the attack of Section 4.3.2 no longer exists with p large *enough*, or *infinite*.

One effective attack in the case of an *infinite* norm is simply to change the vector $E = (0\dots0,1,0\dots0)$ introduced in the previous subsection for $E = (1\dots1)$. The idea is that modifying non–maximal coordinates of a given vector does not *substantially* affect[5] the distance to the unbiased gradient for the modified vector. From this change on E, we proceed as in the previous subsection.

4.4 Bulyan

In addition to being Byzantine–resilient in the sense that it ensures convergence, our algorithm, *Bulyan*[6], also ensures that each coordinate is *agreed on* by a majority of vectors[7] that were selected by a Byzantine–resilient aggregation rule $\mathscr{A}$. This rule $\mathscr{A}$ can for example be *Brute*, *Krum*, a Medoid, the geometric median or any other Byzantine–resilient rule based on an ℓ^p norm or the infinite norm.

Let $\mathscr{A}$ be any (α, f)-Byzantine–resilient aggregation rule. Bulyan($\mathscr{A}$) requires $n \geq 4f+3$ received gradients in two steps. The first one is to *recursively* use $\mathscr{A}$ to select $\theta = n-2f$ gradients, namely:

1. With $\mathscr{A}$, choose, among the proposed vectors, the closest one to $\mathscr{A}$'s output; for Krum or the Medoid, this would be the exact output of $\mathscr{A}$.
2. Remove the chosen gradient from the "received set", and add it to the "selection set", noted $\mathscr{S}$.
3. Loop back to step 1 if $|\mathscr{S}| < \theta$, with $|\cdot|$ the cardinality.

With $n \geq 4f+3$, we ensure that there is a *quorum* of workers, i.e. $2f+3$, for each use of $\mathscr{A}$.

Since $\theta = n-2f \geq 2f+3$, this selection $\mathscr{S} = (S_1 \dots S_\theta)$ 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 $\beta = \theta - 2f \geq 3$, the second step is to *generate* the resulting gradient $G = (G[1] \dots G[d])$, so that for each of its coordinates $G[\cdot]$:

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

where: $\mathscr{M}[i] = \underset{\mathscr{R}\subset\mathscr{S},\,|\mathscr{R}|=\beta}{\operatorname{argmin}} \left(\sum_{X\in\mathscr{R}} |X[i] - \text{median}[i]| \right)$ and: $\text{median}[i] = \underset{m=Y[i],\,Y\in\mathscr{S}}{\operatorname{argmin}} \left(\sum_{Z\in\mathscr{S}} |Z[i] - m| \right)$.

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

[6] Bulyan, Ilyan or more commonly, Julian count of Ceuta, was a Byzantine general, stationed in north Africa, who betrayed the Byzantine empire, in this sense, he was "Byzantine to the Byzantines".

[7] Agreed through the gradients they submitted, of course.

Simply stated: each i-th coordinate of G is equal to the average of the β closest i-th coordinates to the median i-th coordinate of the θ selected gradients.

Let $\mathscr{C}$ be the computational cost of running $\mathscr{A}$ for each epoch at the master to aggregate the gradients.

Proposition 3. *(Cost of one Bulyan($\mathscr{A}$) aggregation.)*
(1) The average computational complexity of Bulyan($\mathscr{A}$) is $O\big((n-2f)\mathscr{C} + d\,n\big)$ for each epoch on the master.
(2) If $\mathscr{A}$ is GeoMed or Krum, this cost is $O\big(n^2 d\big)$.

Proof. (1) We iterate $\mathscr{A}$ as much as $\theta = n - 2f$ times to get the selected vectors, then we run quick–select to get each median component ($O(n)$ on each coordinate, i.e. $O(d\,n)$ times) and another quick–select to get the β closest coordinates (another $O(d\,n)$). Note: we use quick–select instead of quick–sort since we do not need ordered values, just the set of the β closest values.

For point (2) of the proposition, in fact, if we know more about how $\mathscr{A}$ is performed, we can get rid of the $n-2f$ multiplications when iterating $\mathscr{A}$: concretely, $\mathscr{A}$ relies on distance computations between proposed vectors, when we iterate it in the same epoch, we do not need to re-compute those distances and would amortize the cost. For instance, for Krum or GeoMed, Bulyan(Krum) and Bulyan(GeoMed) have a cost of $O\big(n^2 d + n\,d\big)$. In modern machine learning, the models are very large, and $d \gg n$ holds. Therefore Bulyan runs in the same $O\big(n^2 d\big)$ of the base GAR rule it is using[8]. □

Byzantine Leeway Reduction by Bulyan.

In the introduction of this chapter and in Section 4.3, we explained that the curse of dimensionality leaves the Byzantine worker, at a coordinate i, with a margin of $\Omega(f(d))$ 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 $O\left(\frac{\sigma}{\sqrt{d}}\right)$ of honest workers' coordinates. Therefore, narrowing the aforementioned margin to the desired $\mathscr{O}\left(\frac{1}{\sqrt{d}}\right)$.

Proposition 4. *Denote by Bu_t the vector chosen by Bulyan($\mathscr{A}$) at round t. Then for any dimension $i \in [1,d]$ and any honest worker k proposing gradient g_k, we have $\mathbb{E}|Bu_t[i] - g_k[i]| = \mathscr{O}\left(\frac{\sigma}{\sqrt{d}}\right)$.*

Proof. Let $\xi = (\xi_1, \cdots, \xi_{n-f})$ denote the random $(n-f)$-tupple of samples used by the honest workers. By assumption, the $\xi_k, k = 1 \cdots n-f$, are assumed to be i.i.d. Let $i \in [1..d]$ be any component. We denote by B any vector that is selected by Buylan(A) in the set $\mathscr{M}[i]$ (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, then $B[i]$ is among the closest β propositions to $median[i]$. We

[8] For more comparison, averaging (which is not Byzantine–resilient) already has a cost of $O(d\,n)$.

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 $\mathscr{M}[i]$ 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 $\mathscr{M}[i]$, i.e, $g_l[i] \leq B[i] \leq g_r[i]$. There are three cases:

1) $g_k[i] \in \left]-\infty, g_l[i]\right]$, then $|B[i] - g_k[i]| < |g_l[i] - g_k[i]|$

2) $g_k[i] \in \left]g_l[i], g_r[i]\right[$, then $|B[i] - g_k[i]| < |g_l[i] - g_r[i]|$

3) $g_k[i] \in \left[g_r[i], +\infty\right[$, then $|B[i] - g_k[i]| < |g_r[i] - g_k[i]|$

Denote by $\mathbb{1}_h$ the indicator function of each of the three situations $h = 1,2,3$, i.e. $\mathbb{1}_h = 1$ only if we are in case h, and $\mathbb{1}_h = 0$ otherwise. Then we have the following bound:

$$\begin{aligned}|B[i] - g_k[i]| < \mathbb{1}_1 |g_l[i] - g_k[i]| + \mathbb{1}_2 |g_l[i] - g_r[i]| \\ + \mathbb{1}_3 |g_r[i] - g_k[i]|\end{aligned}$$

Let $B_1, \cdots, B_\beta$ be the β elements of $\mathscr{M}[i]$, the previous inequality holds for every B_h, denote by $\mathbb{1}_{r,h}, r = 1\ldots3$ the corresponding indicator functions for each h, we have:

$$\begin{aligned}|Bu_t[i] - g_k[i]| &\leq \frac{1}{\beta}\sum_{h=1}^{\beta} |B_h[i] - g_k[i]| \\ &\leq \frac{1}{\beta}\sum_{h=1}^{\beta} \big(\mathbb{1}_{1,h} |g_l[i] - g_k[i]| \\ &\quad + \mathbb{1}_{2,h} |g_l[i] - g_r[i]| + \mathbb{1}_{3,h} |g_r[i] - g_k[i]|\big)\end{aligned}$$

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) $\mathbb{E}\|g_r - g_l\| = \mathbb{E}\|g_k - g_l\| = \mathbb{E}\|g_k - g_r\| \leq \mathbb{E}\|g_k - G\| + \mathbb{E}\|G - g_r\| = O(\sigma)$, where G is the unbiased estimator used by the honest workers with a bounded variance such that, component–wise (we divide by $\sqrt{d}$): $\mathbb{E}|Bu_t[i] - g_k[i]| = O(\frac{\sigma}{\sqrt{d}})$. □

Proposition 4 proves that Bulyan($\mathscr{A}$) 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($\mathscr{A}$) introduce an additional bias in gradient estimations? The answer, provided by Proposition 1, is *no*. We show that Bulyan($\mathscr{A}$) keeps the gradient estimation in the cone of angle α around the true gradient. In particular, Bulyan($\mathscr{A}$) is also provably convergent.

Corollary 1. *Let $\mathscr{A}$ be an (α, f)-Byzantine-resilient aggregation rule used by Bulyan. Then*

Bulyan($\mathscr{A}$) is also (α, f)–Byzantine–resilient.

Proof. This is an immediate consequence of the (α, f)–Byzantine–resilience of $\mathscr{A}$ (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 $\mathscr{A}$ on a set of vectors of cardinal $\geq 2f+2$. Let g be the true gradient, a triangle inequality applied between g, Bu and the β terms coming from the iterations of $\mathscr{A}$, call them $\mathscr{A}_k, k = 1, \cdots \beta$ gives: $\|Bu - g\| \leq \frac{1}{\beta}\|\mathscr{A}_k - g\|$. Given how $\mathscr{A}$'s iterations are performed (without re-sampling ξ), the $\mathscr{A}_k$ are themselves i.id 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($\mathscr{A}$), 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 Proposition 1, Bulyan is also (α, f)–Byzantine–resilient, by Proposition 2 of [21] guarantees almost sure convergence. □

4.5 Evaluation

We implemented the three (α, f)–Byzantine–resilient gradient aggregation rules presented in Section 4.2.3, along with the attack introduced in Section 4.3. We report in this section on the actual impact this attack can have, on the MNIST and CIFAR–10 problems, 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.

4.5.1 Overview of the Studied 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 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

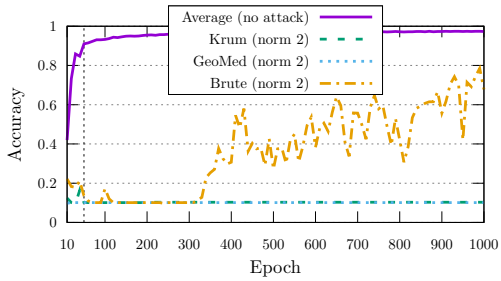


Figure 4.2 – MNIST: accuracy on the testing set up to epoch 1000, comparing the presented aggregation rules under our attack. The attack was maintained only up to epoch 50 (dotted line). The *average* is the reference: it is the accuracy the model would have shown if only non–Byzantine gradients had been selected.

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 $\eta\left(epoch\right) = \eta_0 \frac{r_\eta}{epoch + r_\eta}$. The initial learning rate η_0, the fading rate r_η, and the mini–batch size depend on each experiment.

The accuracy is always measured on the *testing set*.

4.5.2 Results

Attack on Brute, Krum and GeoMed

Figures 4.2 and 4.3 shows the impact of our attack on the aggregation rules presented in Section 4.2.3. The *average* rule computes the arithmetic mean of the submitted gradients.

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

Krum/GeoMed	30 non–Byzantines + 27 Byzantines
Brute	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 Brute), and for the worker counts:

Krum/GeoMed	21 non–Byzantines + 18 Byzantines
Brute	6 non–Byzantines + 5 Byzantines
Average	21 non–Byzantines + 0 Byzantines

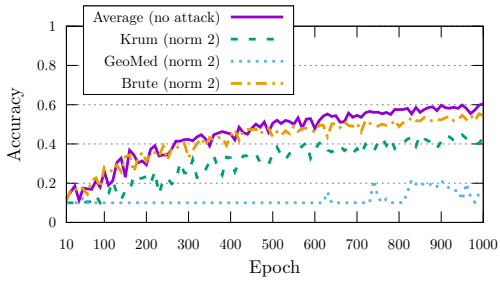


Figure 4.3 – CIFAR–10: accuracy on the testing set up to epoch 1000**, comparing the presented aggregation rules under our attack. The *average* is the reference: it is the accuracy the model would have shown if only non–Byzantine gradients had been selected.**

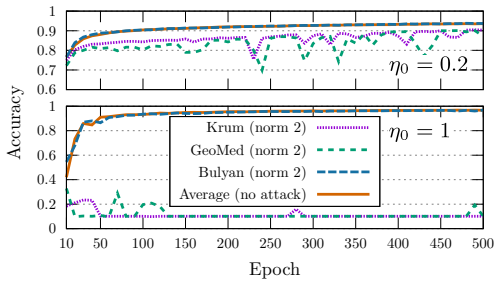


Figure 4.4 – MNIST: accuracy on the testing set up to 500 epochs for Krum, GeoMed, Bulyan ($\mathscr{A}$ = Krum) rules. This graph illustrates the impact of the learning rate, as described in Section 4.3.1.

In Figure 4.2, the attack is maintained only up to 50 epochs. As shown, and except for Brute, 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 epochs. In Figure 4.3, the attack is never stopped. Only Brute preserved the accuracy. Krum suffered a 33% decrease at epoch 1000, and GeoMed failed to produce a useful model.

Higher learning rates and lower batch sizes naturally extend the effectivity of our attack, by increasing both its *exploratory* capabilities and the variance of the non–Byzantine submissions. In the Appendix, we present slightly different initial parameters, for which the attack completely prevented any learning.

The effect of Bulyan

Figures 4.4 and 4.5, respectively for MNIST and CIFAR–10, compares Krum, GeoMed and Bulyan (with $\mathscr{A}$ = Krum).

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. Brute cannot be used with that many workers, see Section 4.2.3. In Figure 4.4, with $\eta_0 = 1$, Krum and GeoMed fail to prevent the attack from *pushing* the model into an *ineffective* state, despite the reduced

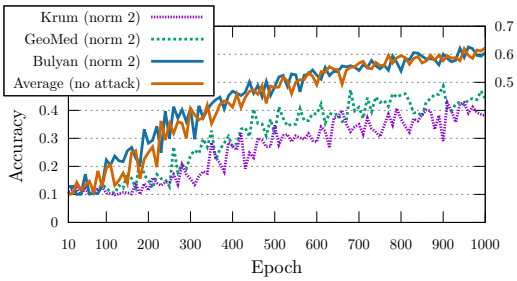


Figure 4.5 – CIFAR–10: accuracy on the testing set up to 1000 epochs for Krum, GeoMed, Bulyan ($\mathscr{A}$ = Krum) rules. The arithmetic mean of non–Byzantine gradients serves as reference.

proportion of Byzantine workers from roughly $\frac{1}{2}$, in Figure 4.2, to roughly $\frac{1}{4}$. 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 4.5, we do the same experiment with CIFAR–10. As with MNIST, only Bulyan is not affected by our attack.

The cost of Bulyan

For both MNIST and CIFAR–10, we use the same configuration as in the experiments of Section 4.5.2.

In Figure 4.6, 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 epoch. 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.

4.6 Concluding Remarks

In very high dimensions, and with highly–non convex cost functions, our work shows that the inaugural distributed–computing defenses [21, 31, 155] against poisoning attacks, tough provably converging, remain frail in the face of *curse of dimensionality* attacks. They might indeed converge, as promised, but to the worst possible region. To defend against that, we introduce Bulyan, which we theoretically prove to significantly reduce the adversarial leeway that causes this drift to sub–optimal models.

We empirically show that Bulyan, indeed avoids convergence to ineffectual models, and instead, ends up learning models that are comparable to a reasonable benchmark: a non–attacked averaging scheme. However, the question of finding "the best direction" possible for non–convex cost

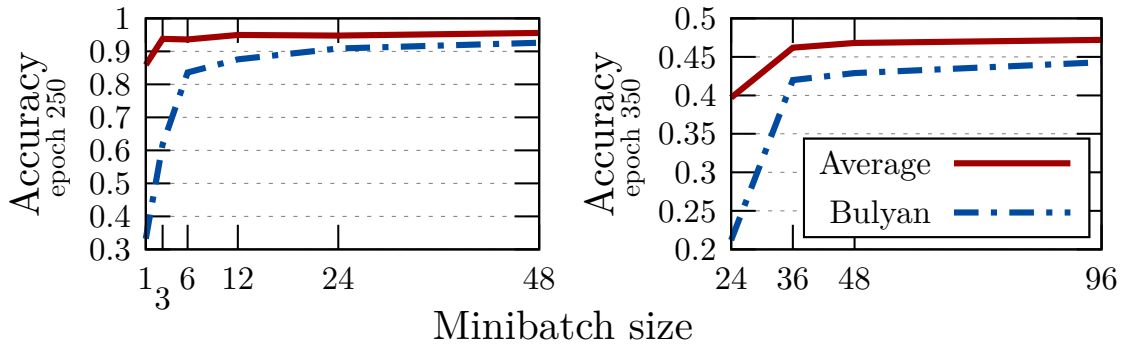


Figure 4.6 – MNIST (left), CIFAR–10 (right): Accuracy on the testing set at epoch 250 **for Average and Bulyan. There are** $n = 39$ **workers and no adversary, but** f **is declared to** 9**. This shows the trade–off between the Byzantine robustness of Bulyan and the loss in convergence speed it introduces.**

functions remains one of the most challenging ones in optimization for machine learning [35], especially when we stick to such a cheap, first–order method as SGD, and avoid expensive, Hessian–like, computations as pointed out by [144].

5 Kardam: Asynchronous Byzantine Gradient Descent

5.1 Introduction

The Byzantine distributed ML solutions we have seen so far assume a restrictive synchronous model. In each epoch, (1) all (honest) workers are supposed to use the exact same model to compute the gradient, and (2) the parameter server waits for a quorum of workers before aggregating their gradients. When networks are asynchronous, exhibiting heterogeneous (sometimes arbitrary) communication delays, synchronous solutions inevitably lead to slow convergence.

Asynchronous SGD algorithms, on the other hand, enable huge performance benefits despite heterogeneous communication delays [109, 112, 79, 105]. In short, such algorithms (1) allow workers to make use of a stale model as well as (2) update the model as soon as a new gradient is delivered (instead of waiting for a quorum). Nevertheless, none of these asynchronous algorithms tolerate any Byzantine behavior. In fact, all provably convergent asynchronous SGD algorithms assume that all the workers are permanently honest about their gradient, i.e., provide unbiased estimations of the actual gradient (Figure 5.1).

Combining asynchrony with Byzantine resilience is challenging. In particular, aggregating gradients that were computed on different models requires the knowledge of how the curvature of the cost function evolves with staleness. This curvature determines the window of synchrony within

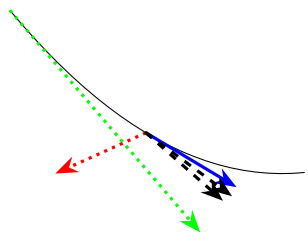


Figure 5.1 – The gradients computed by non-stale honest workers (black dashed arrows) are distributed around (and are on average equal to) the actual gradient (solid blue arrow) of the cost function (thin black curve). A Byzantine worker can propose an arbitrary poisoning vector (red dotted arrow). A honest but stale worker computes the correct gradient but for a stale version of the model (long green dotted arrow).

which a synchronous method can be transposed into an asynchronous context. Roughly speaking, the more locally curved the cost function is, the narrower this window and vice versa. Estimating the curvature requires heavy computations of the Hessian matrix ($O(d^2)$), not to mention the fact that this would deprive the parameter server from the most prominent advantage of asynchrony, namely updating the model as soon as a *single* gradient is delivered (i.e., the parameter server would need to aggregate a quorum).

In this chapter, we consider for the first time the situation where a significant fraction of workers ($\frac{f}{n}$) can be Byzantine (arbitrarily adversarial) and consider unbounded communication delays. Such situation corresponds to that of many realistic distributed platforms today. We present the first asynchronous Byzantine gradient descent algorithm, we call *Kardam*. Kardam leverages the Lipschitzness of the cost function to filter out gradients from potentially Byzantine workers, while prohibiting Byzantine workers from flooding the parameter server (which in turn would prevent honest workers from updating the model). Kardam also uses a dampening scheme that scales each gradient based on its staleness. The computation overhead for each update is negligible as the filtering component of Kardam is mostly scalar-based. The time complexity for each update computed in terms of the dimension d of a gradient is $O(d + fn)$. This complexity is the same as the standard complexity of an asynchronous SGD update ($O(d)$) for the very high-dimensional learning models of today ($d \gg (f, n)$). We prove the convergence of Kardam and precisely determine its convergence rate. In particular, we prove its self-stabilizing property using a refined version of the global confinement argument [23].

We implemented and deployed Kardam in a distributed setting and we report in this chapter on its in-depth empirical evaluation on the CIFAR-100 and EMNIST datasets. In particular, we evaluate the overhead of Kardam with respect to non Byzantine-resilient solutions. Kardam does not tamper with the learning procedure (i.e., include additional noise), yet it does induce a slowdown that we empirically show to be less than $\frac{f}{n}$, where f is the number of Byzantine failures tolerated and n the total number of workers (we also prove that theoretically). Finally, we show that the dampening component (when plugged on an asynchronous non Byzantine-resilient SGD solution) outperforms alternative staleness-aware asynchronous competitors in environments with honest workers.

The code (written by my co-authors G.D. and R.P.) to reproduce our experiments as well as a few additional results (varying f) is available at https://github.com/LPD-EPFL/kardam.

5.2 Model for Asynchronous SGD

As in the previous two chapters, we consider the general distributed model for machine learning, namely the parameter server [1, 107][1]. We assume that f of the n workers are Byzantine (behave arbitrarily). Following the traditional assumption in distributed computing, we assume that the identities of the Byzantine workers are unknown whereas f (in practice, an upper bound) is

[1] Classical techniques of state-machine replication [113] can be used to ensure that the parameter server is reliable.

known. Computation is divided into (infinitely many) asynchronous model updates (epochs).

Given the entangled aspect of asynchronous distributed computing, we provide a table of additional notations for this chapter.

Definition 2 (Time)**.** *The global epoch (denoted by t) represents the global logical clock of the parameter server (or equivalently the number of model updates). The local timestamp (denoted by l_p) for a given worker p, represents the epoch of the model that the worker receives from the server and computes the gradient upon. The difference $t - l_p$ can be arbitrarily large due to the asynchrony of the network.*

t	Epoch at the parameter server,incremented after each update.
l_p	Timestamp (given by the parameter server) of the model currently used by worker p.
$\boldsymbol{x}_t$	Model (parameter vector) at epoch t with dimensionality d.
γ_t	Learning rate at epoch t s.t $\sum_{t=1}^{\infty} \gamma_t = \infty$ and $\sum_{t=1}^{\infty} \gamma_t^2 < \infty$.
g_p	Each gradient is a tuple $[g_p, l]$ denoting that a worker p computed the gradient g_p w.r.t x_l.
$\lvert X \rvert$	Cardinality of a set X.
M	Number of gradients that the server waits for before updating the model parameters. ($M = 1$ in asynchrony).
$\mathscr{G}_t$	Set of gradients that the server receives in epoch t. Note that $\lvert\mathscr{G}_t\rvert = M$.
τ_{tl}	Staleness value for a gradient $[g, l]$ at epoch t ($\tau_{tl} \triangleq k - l$).
ξ	Mini-batch of training examples.
$Q(x)$	Cost function for a model x.
K	Global Lipschitz coefficient of ∇Q, i.e $K = \sup_{x,y \in \mathbb{R}^d} (\frac{\lVert \nabla Q(x) - \nabla Q(y) \rVert}{\lVert x - y \rVert})$.

Table 5.1 – The notations used in Kardam.

During each epoch t, the parameter server broadcasts the model $x_t \in \mathbb{R}^d$ to all the workers. A cost function Q reflects the quality of the model for the learning task. Each non-Byzantine worker p computes an estimate $g_p = G(x_{l_p}, \xi_p)$ of the actual gradient $\nabla Q(x_{l_p})$ of the cost function Q, where ξ_p is a random variable representing, for example, the sample (or a mini-batch of samples) drawn from the dataset at worker p. Each worker p sends the timestamp l_p (to declare which version of the model it used) and the gradient g_p. See Table 5.1 for notational details.

A Byzantine worker b proposes a gradient g_b which can deviate arbitrarily from $G(x_{l_b}, \xi_b)$ (see Figure 2.1). A Byzantine worker may have full knowledge of the system, including the gradients proposed by other workers. Byzantine workers can furthermore collude, as typically assumed

in the distributed computing literature [100, 113, 27]. Since the communication is assumed to be asynchronous, the parameter server takes into account the first gradient received at time t. The parameter server then either suspects the gradient and ignores it, or employs it to update the model and move to epoch $t+1$. We make the following assumptions about any honest worker p.

Assumption 1 (Unbiased gradient estimator)**.**

$$\mathbb{E}_{\xi_p} G(x_{l_p}, \xi_p) = \nabla Q(x_{l_p})$$

Assumption 2 (Bounded variance)**.**

$$\mathbb{E}_{\xi_p} \|G(x_{l_p}, \xi_p) - \nabla Q(x_{l_p})\|^2 \leq d\sigma^2$$

Assumptions 1 and 2 are common in the literature [23] and hold if the data used for computing the gradients is drawn uniformly and independently.

Assumption 3 (Linear growth of r-th moment)**.**

$$\mathbb{E}_{\xi_p} \|G(x, \xi_p)\|^r \leq A_r + B_r \|x\|^r \ \ \forall x \in \mathbb{R}^d, \ \ r = 2, 3, 4$$

Assumption 3 translates into "the r-th moment of the gradient estimator grows linearly with the r-th power of the norm of the model" as assumed in [23].

Assumption 4 (Lipschitz gradient)**.**

$$||\nabla Q(x_1) - \nabla Q(x_2)|| \leq K||x_1 - x_2||$$

Assumption 5 (Convexity in the horizon)**.** *We require that beyond a certain horizon, $\|x\| \geq D$, there exist $\epsilon > 0$ and $0 \leq \beta < \pi/2$ such that $\|\nabla Q(x)\| \geq \epsilon > 0$ and $\frac{\langle x, \nabla Q(x)\rangle}{\|x\| \cdot \|\nabla Q(x)\|} \geq \cos\beta$.*

Assumptions 4 and 5 are the same as in [21], the first is classic, the second is a slight refinement of a similar assumption in [23]. It essentially states that, beyond a certain horizon D in the parameter space, the opposite of the gradient points towards the origin.

Definition 3 (Byzantine resilience)**.** *Let Q be any cost function satisfying the assumptions above. Let A be any distributed SGD scheme. We say that A is Byzantine-resilient if the sequence $\nabla Q(x_t) = 0$ converges almost surely to zero, despite the presence of up to f Byzantine workers.*

5.3 Kardam

In this section, we present the two main components of our algorithm, *Kardam*[2], namely the filtering and the dampening components. We also establish the theoretical guarantees of each component.

5.3.1 Byzantine-resilient Filtering Component

The parameter server accepts a gradient g_p from worker p (i.e., updates the model with g_p) if g_p is accepted by the Byzantine-resilient filtering component of Kardam. This component itself consists of a *Lipschitz filter* followed by a *frequency filter* that we describe in the following.

Lipschitz filter. This filter can be viewed as a kinetic validation at the parameter server based on the empirical Lipschitzness.

Definition 4 (Empirical Lipschitz coefficient)**.** *The empirical Lipschitz coefficient at worker p is defined as* $\hat{K}_p = \frac{\|g_p - g_p^{prev}\|}{\|x_{l_p} - x_{l_p}^{prev}\|}$*. The empirical Lipschitz coefficient at the parameter server is defined with respect to a received gradient from worker p and an updated gradient from worker q at the previous epoch* $(t-1)$ *as* $\hat{K}_t^p = \frac{\|g_p - g_q\|}{\|x_t - x_{t-1}\|}$.

The empirical Lipschitz coefficient ($\hat{K}_p$) reflects the local empirical observation of the gradient evolution, normalized by the model evolution. Each worker p derives this coefficient between the current and the previous models used to compute the current and previous gradients of p respectively.

The Lipschitz filter accepts the candidate gradient g_p if the empirical Lipschitzness for g_p (Definition 4) is not suspicious, i.e., if it is smaller than a median empirical Lipschitzness of all the workers as follows.

$$\hat{K}_t^p \leq \hat{K}_t \triangleq quantile_{\frac{n-f}{n}}\{\hat{K}_p\}_{p \in P}$$

where $quantile_{\frac{n-f}{n}}$ represents the element that separates the $\frac{n-f}{n}$ fraction of workers with the smallest empirical Lipschitz values from the remaining $\frac{f}{n}$ fraction with the highest values (i.e., the $(100 \cdot \frac{n-f}{n})^{th}$ percentile). We highlight that there exist two honest workers p_1 and p_2 such that $\hat{K}_{p_1} \leq \hat{K}_t \leq \hat{K}_{p_2}$ since our single dimensional median is guaranteed to be bounded by values from any group of size $n-f$ (i.e., group of honest workers).

The complexity of the Lipschitz filter is $O(d+n)$ (computing distances on 2 d-dimensional vectors, then getting the median of n scalars, in $O(n)$ with quick-select).

Obviously, the Lipschitz filter will end up filtering fast workers (that reach the more curved regions of the cost functions before the others) or slow workers (that are delayed in a curved region while

[2]Kardam was a Bulgarian khan who pre-empted the Byzantine empire's invasion. He was the predecessor of *Krum* [21], the Bulgarian khan gave his name to the first provable solution for the synchronous Byzantine SGD problem.

everyone else is already in a less curved region). We note that this filter, roughly speaking, suspects f workers to be Byzantine and thus a pessimistic choice for f would increase the overhead of Kardam (filters more gradients due to a pessimistic choice for f).

Theorem 1 (Optimal Slowdown)**.** *We define the slowdown SL as the ratio between the number of updates from honest workers that pass the Lipschitz filter and the total number of updates delivered at the parameter server. We derive the upper and lower bounds of SL in the following.*

$$\frac{n-2f}{n-f} \leq SL \leq \frac{n-f}{n}$$

The lower and upper bounds are tight and hold when there are f Byzantine workers and no Byzantine workers respectively. Therefore Kardam achieves the optimal bounds with respect to any Byzantine-resilient SGD scheme and $n \approx 3f$ workers.

Proof. Any Byzantine-resilient SGD scheme assuming f Byzantine workers will at most use $\frac{n-f}{n}$ of the total available workers (upper bound). By definition, the Lipschitz filter accepts the gradients computed by $\frac{n-f}{n}$ of the total workers with empirical Lipschitzness below $\hat{K}_t$. If every worker is honest, then the filter accepts gradients from $\frac{n-f}{n}$ of the workers. We thus get the tightness of the upper bound for the slowdown of Kardam. For the lower bound, the Byzantine workers can know that putting a gradient proposition above $\hat{K}_t$ will get them filtered out and the parameter server will end up using only the honest workers available. The optimal attack would therefore be to slow down the server by getting tiny-Lipschitz gradients accepted while preventing the model from actually changing. This way, the Byzantine workers will make the server filter gradients from a total of f out of the $n-f$ honest workers, leaving only $n-2f$ useful workers for the server. □

Theorem 2 (Byzantine resilience in asynchrony)**.** *Let A be any distributed SGD scheme. If the maximum successive gradients that A accepts from a single worker and the maximum delay are both unbounded, then A cannot be Byzantine-resilient when $f \geq 1$.*

Proof. Without any restrictions, the parameter server could only accept successive gradients from the same Byzantine worker (without getting any update from any honest worker), for example, if the Byzantine worker is faster than any other worker (which is true by the definition of a Byzantine worker and by the fact that delays on (honest) workers are unbounded). This way, the Byzantine worker can force the parameter server to follow arbitrarily bad directions and never converge. Hence, without any restriction on the number of gradients from the workers, we prove the impossibility of asynchronous Byzantine resilience. Readers familiar with distributed computing literature might note that if asynchrony was possible for Byzantine SGD without restricting the number of successive gradients from a single worker, this could be used as an abstraction to solve asynchronous Byzantine consensus (that is impossible to solve [66]). This provides another proof (by contradiction) for our theorem. □

Given Theorem 2 and the objective of making Kardam Byzantine-resilient in an asynchronous

environment (i.e., while letting workers be arbitrarily delayed), we introduce the frequency filter.

Frequency filter. The goal of this filter is to limit the number of successive gradients [3] from a single worker to a value of f, thus not allowing the Byzantine workers to prevent honest workers from updating the model. Consider L as the list of workers who computed the last $2f$ accepted gradients. Assume that the candidate gradient g_q passes the Lipschitz filter. The frequency filter adds worker q at the end of L (i.e., at position $L[2f+1] = q$). If adding this candidate gradient g_q makes any set of workers of size f appear more than f times in L, then q is rejected, otherwise, q is accepted. For each worker p, the number of times p appeared in L is denoted by n_p. The frequency filter accepts a gradient from worker p if the following holds: $\sum_{p\in\theta} n_p \leq f$, where θ denotes the set of f workers with the f maxima of $\{n_p\}_{p=1}^{n}$. The time complexity of the frequency filter is $O(2f+1)$ to compute $\{n_p\}_{i=1}^{n}$ (going through the list L of size $2f+1$), in addition to $O(fn)$ to find the f maxima among $\{n_p\}_{i=1}^{n}$.

Lemma 3 (Limit of successive gradients)**.** *The frequency filter ensures that any sequence of length $2f+1$ consequently accepted gradients contains at least $f+1$ gradients computed by honest workers.*

Proof. Given any sequence of $2f+1$ consequently accepted gradients (L), we denote by S the set of workers that computed these gradients. The frequency filter guarantees that any f workers in S computed at most f gradients in L. At most f workers in S can be Byzantine, thus at least $f+1$ gradients in L are from honest workers. □

Given asynchrony (unbounded delays), we do not assume any upper bound on the norm of the model, the norm of the gradients or the values of the cost function (regularization schemes can make the loss grow arbitrarily and thereby the gradients norms). However, we assume (as in [23]) that the cost function is lower bounded by a positive scalar. This assumption holds for all the standard cost functions that are at least lower bounded by zero (e.g., square loss, cross-entropy or any norm-based cost). We denote ***Kar*** by the sequence of gradients accepted (i.e., not filtered) by Kardam, and by $\boldsymbol{Kar}_t$ the gradient accepted by Kardam in epoch t.

Theorem 3 (Correct cone and bounded statistical moments)**.** *If $N > 3f+1$ then for any $t \geq t_r$ (we show that $t_r \in O(\frac{1}{K\sqrt{|\xi|}})$ where $|\xi|$ is the batch-size of honest workers):*

$$\mathbb{E}(\|\boldsymbol{Kar}_t\|^r) \leq A'_r + B'_r\|x_t\|^r \text{ for any r=2, 3, 4 and}$$
$$\langle \mathbb{E}(\boldsymbol{Kar}_t), \nabla Q_t\rangle = \Omega(1 - \frac{\sqrt{d}\sigma}{\|\nabla Q(x_t)\|})\|\nabla Q(x_t)\|^2.$$

Expectations are on any randomness up to time t conditioned on the past.

We begin by providing a proof sketch for readers who are only interested by the general intuitions.

[3] One open problem left in our work is the extent to which this filter is too harsh for asynchronous schemes. For instance, it can at least lead to a randomly shuffled round-robin schedule.

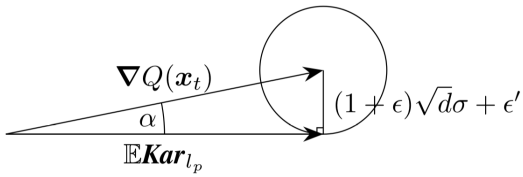


Figure 5.2 – If $\left\|\mathbb{E}\boldsymbol{Kar}_{l_p} - \nabla Q(x_t)\right\| \leq (1+\epsilon)\sqrt{d}\sigma + \epsilon'$ then $\langle \mathbb{E}\boldsymbol{Kar}_{l_p}, \nabla Q(x_t)\rangle$ is upper bounded by $(1-\sin\alpha)\|\nabla Q(x_t)\|^2$ where $\sin\alpha = \frac{(1+\epsilon)\sqrt{d}\sigma+\epsilon'}{\|\nabla Q(x_t)\|}$.

Proof. (Sketch) The frequency filter guarantees that there is always an update from a honest worker in any sequence of $f+1$ updates (Lemma 3), i.e., at any time t, there is an interval $t-i$ where $i < f+1$ such that the vector that passed the Lipschitz filter is a vector sent by an honest worker (therefore an unbiased estimation of the true gradient). With this in mind, and using triangle inequalities over a series of (at most f) previous updates, we prove inequalities on the r-th statistical moments of ***Kar***. Those inequalities are in turn plugged into the requirements for the (almost sure) global confinement argument of [23].

With the guarantees of almost sure global confinement, and using the Liptschitz properties, and (again) the existence of honest (unbiased) workers in the "recent past" as explained above, we find the lower-bound of the scalar product between the two desired vectors $\langle \mathbb{E}(\boldsymbol{Kar}_t), \nabla Q_t\rangle$ when their distance is small enough compared to their own norms (Figure 5.2). This finally shows that Kardam remains in a cone of an angle α that is upper bounded by $\arcsin(\frac{(1+\epsilon)\sqrt{d}\sigma+\epsilon'}{\|\nabla Q(x_t)\|})$ with appropriately chosen ϵ and ϵ'. □

We now provide the complete proof.

Proof. First of all, it is important to note that a Byzantine worker can lie about its Lipschitz coefficient without being able to fool the parameter server. The median Lipschitz coefficient is always bounded between the Lipschitz coefficients of two correct worker, and it is against that the gradient of the Byzantine worker would be tested to be filtered out if harmful and accepted if useful.

We start the proof of Theorem 3 by proving that Kardam acts as self-stabilizing mechanism that guarantees the global confinement of the parameter vector using the following remark.

Lemma 4 (Global Confinement)**.** *Let x_t the sequence of parameter models visited by **Kar**. There exist a constant $D > 0$ such that the sequence x_t almost surely verifies $\|x_t\| \leq D$ when $t \mapsto \infty$.*

Proof. (Global Confinement) Lemma 4 can be proven by using Remark 1 and the proof of confinement in [23].

Remark 1. *Let $r = 2,3,4$. There exist $A'_r \geq 0$ and $B'_r \geq 0$ such that:*
$(\forall t \geq 0)\mathbb{E}\|\boldsymbol{Kar}_t(x_t,\xi)\|^r \leq A'_r + B'_r\|x_t\|^r$

Proof. (Remark 1). Note that if $\boldsymbol{Kar}_t(x_t)$ comes from a honest worker, we have $\boldsymbol{Kar}_t(x_t,\xi) = G(x_t,\xi)$ therefore, $(\forall t \geq 0)\mathbb{E}\|\boldsymbol{Kar}_t(x_t,\xi)\|^r \leq A_r + B_r\|x_t\|^r$ since by assumption on the estimator G used by honest workers, we have
$(\forall x \in \mathbb{R})\mathbb{E}\|G(x,\xi)\|^r \leq A_r + B_r\|x\|^r$.

Let $t > 2f+1$ be any epoch at the parameter server. Because of the Lipschitz filter (passed by $\boldsymbol{Kar}_t$), there exists $i \leq f$ such that $\boldsymbol{Kar}_{t-i}(x_{t-i})$ comes from an honest worker. Therefore, $\|x_{t-i}\| \leq \|x_t\| + \sum_{l=1}^{i}\gamma'_{t-l}\boldsymbol{Kar}_{t-l}(x_{t-l}) \leq \|x_t\| + \sum_{l=1}^{i}\gamma_{t-l}\cdot\frac{\min(\boldsymbol{Kar}_0,\|\boldsymbol{Kar}_{t-l}(x_{t-l})\|)}{\|\boldsymbol{Kar}_{t-l}(x_{t-l})\|}\cdot\boldsymbol{Kar}_{t-l}(x_{t-l}) \leq f\cdot\boldsymbol{Kar}_0 + \|x_t\|$.

So, for $r = 2,3,4$ there exists C_r such that $\|x_{t-i}\|^r \leq (f\cdot\boldsymbol{Kar}_0)^r + C_r\|x_t\|^r$.

According to the Lipschitz criteria:

$$
\begin{aligned}
&\|\boldsymbol{Kar}_t(x_t)\| \\
&\leq K_t(\|x_t\| + \|x_{t-1}\|) + \|\boldsymbol{Kar}_{t-1}(x_{t-1})\| \\
&\leq \sum_{l=1}^{i} K_{t-l+1}(\|x_{t-l+1}\| + \|x_{t-l}\|) + \|\boldsymbol{Kar}_{t-i}(x_{t-i})\| \\
&\leq 2K\sum_{l=0}^{i}\|x_{t-l}\| + \|\boldsymbol{Kar}_{t-i}(x_{t-i})\| \\
&\leq 2K\sum_{l=0}^{i}\sum_{s=l}^{i-1}[\gamma'_{t-s}\cdot\|\boldsymbol{Kar}_{t-s}(x_{t-s})\| + \|x_{t-i}\|] + \|\boldsymbol{Kar}_{t-i}(x_{t-i})\| \\
&\leq 2K\sum_{l=0}^{i}\sum_{s=l}^{i-1}\gamma_{t-s}\|\boldsymbol{Kar}_{t-s}(x_{t-s})\|\cdot\frac{\min(\boldsymbol{Kar}_0,\|\boldsymbol{Kar}_{t-s}(x_{t-s})\|)}{\|\boldsymbol{Kar}_{t-s}(x_{t-s})\|} \\
&\quad + 2fK\|x_{t-i}\| + \|\boldsymbol{Kar}_{t-i}(x_{t-i})\| \\
&\leq Kf(f-1)\boldsymbol{Kar}_0 + 2fK\|x_{t-i}\| + \|\boldsymbol{Kar}_{t-i}(x_{t-i})\| \\
&= D + E\|x_{t-i}\| + F\|\boldsymbol{Kar}_{t-i}(x_{t-i})\|
\end{aligned}
$$

Where K is the global Lipschitz. (We do not need to know the value of K to implement ***Kar*** but we use it for the proofs.) Taking both side of the above inequality to the power r, we have the following for $r = 2\ldots4$ for constants D_r, E_r and F_r:

$$\|\boldsymbol{Kar}_t(x_t)\|^r \leq D_r + E_r\cdot\|x_{t-i}\|^r + F_r\cdot E\|\boldsymbol{Kar}_{t-i}(x_{t-i})\|^r$$

As $\boldsymbol{Kar}_{t-i}(x_{t-i})$ comes from an honest worker, using the Jensen inequality and the assumption on honest workers. We can take the expected value on ξ.

$$
\begin{aligned}
&\mathbb{E}\|\boldsymbol{Kar}_t(x_t)\|^r \\
&\leq D_r + E_r \cdot \|x_{t-i}\|^r + F_r[A_r + B_r\|x_{t-i}\|^r] \\
&= D_r + F_r A_r + \|x_{t-i}\|^r [E_r + F_r B_r] \\
&\leq D_r + F_r A_r + [(f \cdot \boldsymbol{Kar}_0)^r + C_r\|x_t\|^r] \cdot [E_r + F_r B_r] \\
&= D_r + F_r A_r + f^r \boldsymbol{Kar}_0^r [E_r + F_r B_r] + [E_r + F_r B_r] \cdot \|x_t\|^r
\end{aligned}
$$

We denote by $A'_r = D_r + F_r A_r + (f \cdot \boldsymbol{Kar}_0)^r \cdot [E_r + F_r B_r]$ and $B'_r = E_r + F_r B_r$, we obtain:

$\mathbb{E}\|\boldsymbol{Kar}_t(x_t)\|^r \leq A'_r + B'_r\|x_t\|^r$ □

Remark 1 shows that with ***Kar***, all the assumptions of Bottou [23] (Section 5.2) are holding even in the presence of Byzantine workers, and thus, the global confinement of x_t stated in Lemma 4. □

Remark 1 have proved the first part of Theorem 3 To continue the proof of this Theorem, the goal is to find a lower bound on the scalar product between Kardam and the real gradient of the cost. This is achieved via an upper bound on: $\|\mathbb{E}\boldsymbol{Kar}_t - \nabla Q_t(x_t)\|$. Let p the worker whose gradient estimation g_p was selected by Kardam to be the update for epoch t at the parameter server. According to Lemma 3, considering the latest $2f+1$ timestamps, at least $f+1$ of updates came from honest workers. Hence, there exists $i < f$ such that, $\boldsymbol{Kar}_{t-i}$ came from an honest worker. Hence, $\mathbb{E}\boldsymbol{Kar}_{t-i} = \nabla Q_{t-i}$. By applying the triangle inequality twice, we have:

$$
\begin{aligned}
\|\boldsymbol{Kar}_t - \nabla Q(x_t)\| \leq \|&\boldsymbol{Kar}_t - \boldsymbol{Kar}_{t-i}\| \\
&+ \|\boldsymbol{Kar}_{t-i} - \nabla Q(x_{t-i})\| \\
&+ \|\nabla Q(x_{t-i}) - \nabla Q(x_t)\|
\end{aligned}
$$

We know:

$$
\begin{aligned}
\|\boldsymbol{Kar}_{t-i} - \boldsymbol{Kar}_t\| &\leq \sum_{k=i}^{1} \|\boldsymbol{Kar}_{t-k} - \boldsymbol{Kar}_{t-k+1}\| \leq K \sum_{k=1}^{i} \|x_{t-k+1} - x_{t-k}\| \\
&\leq K \sum_{k=1}^{i} \gamma_{t-k} \|\boldsymbol{Kar}_{t-k}\| \leq i \cdot K \cdot \gamma_{t-i} \cdot \|\boldsymbol{Kar}\|_{max(t,i)}
\end{aligned}
$$

where, $\|\boldsymbol{Kar}\|_{max(t,i)}$ is the upper-bound on the norm of ***Kar*** in the list from $t-i$ to $t-1$. Since $i < f$, we have $\|\boldsymbol{Kar}_{t-i} - \boldsymbol{Kar}_t\| \leq fK\gamma_{t-i}\|\boldsymbol{Kar}\|_{max(t,i)}$. Since x_t is globally confined (Lemma 2), by continuous differentiability of Q, so will be $\|\nabla Q(x_{t,i})\|$, therefore $fK\|\boldsymbol{Kar}\|_{max(t,i)}$ is bounded, and multiplies γ_{t-i} in the right hand side of the last inequality, and we know from the hypothesis on the learning rate that $\lim_{t\to\infty} \gamma_t = 0$ (sequence of summable squares, therefore goes to zero).

Since $i < f$ (and obviously, f, as a global variable, is independent of t), then we also have $\lim_{t\to\infty}\gamma_{t-i} = 0$. This means that for every $\epsilon > 0$, eventually, the left hand-side of the above inequality is bounded by $\epsilon\|\boldsymbol{Kar}_{t-i} - \nabla Q(x_{t-i})\|$, more precisely, since γ_t is typically $O(\frac{1}{t})$, this will hold after t_r such that $t_r = \Omega(\frac{1}{\epsilon K})$.

By replacing in Formula 5.3.1, we get:

$$\begin{aligned}\|\boldsymbol{Kar}_t - \nabla Q(x_t)\| &\leq (1+\epsilon)\|\boldsymbol{Kar}_{t-i} - \nabla Q(x_{t-i})\| + \|\nabla Q(x_{t-i}) - \nabla Q(x_t)\| \\ &\leq (1+\epsilon)\|\boldsymbol{Kar}_{t-i} - \nabla Q(x_{t-i})\| + \sum_{s=1}^{i} K_{t-s}\gamma_{t-s}\|\nabla Q(x_{t-s})\| \\ &\leq (1+\epsilon)\|\boldsymbol{Kar}_{t-i} - \nabla Q(x_{t-i})\| + f\cdot K\cdot\gamma_{t-i}\cdot\|\nabla Q\|_{max(t,i)}.\end{aligned}$$

Where K_{t-s} is the real local Lipschitz coefficient of the loss function at epoch $t-s$. Let $j = \min(\frac{\sqrt{d}\sigma}{2}, \|\nabla Q(x_t)\| - \sqrt{d}\sigma)$, $C = \frac{j}{2\epsilon\sqrt{d}\sigma}$, $\epsilon' = \frac{j}{2.C}$. As $\lim_{t\to\infty}\gamma_t = 0$ and $\|\nabla Q\|_{max(t,i)}$ is bounded, there exist a time after which, the above quantity can be made bounded as

$$\|\boldsymbol{Kar}_t - \nabla Q(x_t)\| \leq (1+\epsilon)\|\boldsymbol{Kar}_{t-i} - \nabla Q(x_{t-i})\| + \epsilon'.$$

And hence:

$$\begin{aligned}\|\mathbb{E}(\boldsymbol{Kar}_t) - \nabla Q(x_t)\| &\leq \mathbb{E}(\|\boldsymbol{Kar}_t - \nabla Q(x_t)\|) \\ &\leq (1+\epsilon)\mathbb{E}(\|\boldsymbol{Kar}_{t-i} - \nabla Q(x_{t-i})\|) + \epsilon'.\end{aligned}$$

since $\boldsymbol{Kar}_{t-i}$ comes from a correct worker, we have:

$$\mathbb{E}(\|\boldsymbol{Kar}_{t-i} - \nabla Q(x_{t-i})\|) \leq \sqrt{d}\sigma$$

Therefore, $\|\mathbb{E}(\boldsymbol{Kar}_t) - \nabla Q(x_t)\| \leq (1+\epsilon)\sqrt{d}\sigma + \epsilon'$. Consequently, Kardam only selects vectors that live on average in the cone of radius α around the true gradient, where α is given by:

$\sin(\alpha) = \frac{(1+\epsilon)\sqrt{d}\sigma+\epsilon'}{\|\nabla Q(x_t)\|}$. (as long as $\|\nabla Q(x_t)\| > (1+\epsilon)\sqrt{d}\sigma + \epsilon'$, this has a sense)

Note:

- The $\sqrt{d}$ in $\|\nabla Q(x_t)\| > \sqrt{d}\sigma$ is not a harsh requirement, we are using the conventional notation where $\sqrt{d}\sigma$ is the upper bound on the variance, σ should be seen as the "component-wise" standard deviation, therefore, the norm of a non-trivial gradient is naturally larger than the vector-wise standard deviation of its estimator, which is typically $\sqrt{d}\sigma$.
- As long as the true gradient has *a nontrivial meaning* (it is larger than the standard deviation of its correct estimators), α is strictly bounded between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$, which means that as

long as there is no convergence to null gradients, Kardam is selecting vectors in the correct cone around the true gradient. Most importantly, this angle shrinks to zero when the variance is too small compared to the norm of the gradient, i.e., with large batch-sizes, Kardam boils down to be an unbiased gradient estimator. However, we only require the "component-wise" condition.

In fact, as long as $\|\nabla Q(x_t)\| > \sqrt{d}.\sigma$, we can consider small enough ϵ and ϵ' such that $D_1 = (1+\frac{3}{4C})\frac{\sqrt{d}\sigma}{\|\nabla Q(x_t)\|}$, $D_2 = \frac{1}{C} + \frac{C-1}{C}\frac{\sqrt{d}\sigma}{\|\nabla Q(x_t)\|}$, and $\sin(\alpha) = \min(D_1, D_2) < 1$. This indeed guarantees that $\alpha < \frac{\pi}{2}$, moreover, it is enough to take $C >> \frac{\nabla Q(x_t)\|}{\sqrt{d}\sigma}$ and α would satisfy $\sin(\alpha) \approx \frac{\sqrt{d}\sigma}{\|\nabla Q(x_t)\|}$.

Actually, in a list of L previous selected vectors, more than half of the vectors are from correct workers. (progress is made: liveness)

Consider a sublist of L from L_i to L_j. At the time of adding a worker in L_j, the frequency criteria was checked for the new addition to L. The active table at that time assure that in any new sublist of L, especially L_i^j), any f workers appear at most $\frac{j-i}{2}$ times. As the number of Byzantine workers is maximum f. in sublist L_i^j, the Byzantine workers did less than half of the updates. In other words, at least half of the updates come from honest workers. This proves the safety of Kardam.

The Byzantine workers may stop sending updates or send incorrect updates. In the case where the Byzantine workers stop sending updates, Kardam still guarantees liveness. The reason is that there are at least $2f+1$ honest workers who update the model. □

5.3.2 Staleness-aware Dampening Component

We now present the component of Kardam that enables staleness-aware asynchronous updates for the ML model. For the sake of clarity, we denote the time by $t' \triangleq t - tr$ (Theorem 3). We introduce the *M-soft-async* protocol where the server updates the model only after receiving M gradients. The update rule for Kardam is the following.

$$\begin{aligned} x_{t+1} &= x_t - \gamma_t \boldsymbol{Kar}_t \\ &= x_t - \gamma_t \sum_{[G(x_l;\xi_m),l]\in\mathcal{G}_t} \Lambda(\tau_{tl}) \cdot G(x_l;\xi_m) \end{aligned} \tag{5.1}$$

where $G(x_l;\xi_m)$ denotes the gradient w.r.t the model parameters x_l on the mini-batch ξ_m. We assume that every gradient passes the filtering scheme (Section 5.3.1) at the epoch t. Kardam requires $|\mathcal{G}_t| = M$ gradients for each update.

The difference between the standard SGD update rule and our Equation 5.1 illustrates how Kardam handles asynchronous updates. Kardam dampens each update depending on its staleness value (τ_{tl}). Kardam employs a decay function $\Lambda(\tau_{tl})$ such that $0 \le \Lambda \le 1$ to derive the dampening factor for each distinct value of staleness.

Definition 5 (Dampening function)**.** *We employ a bijective and strictly decreasing dampening*

function $\tau \mapsto \Lambda(\tau)$ with $\Lambda(0) = 1$.[4] *Note that every bijective function is also invertible, i.e., $\Lambda^{-1}(\nu)$ exists for every ν in the range of the Λ function.*

Let Λ_t be the set of Λ values associated with the gradients at timestamp t.

$$\Lambda_t = \{\Lambda(\tau_{tl}) \mid [g, l] \in \mathcal{G}_t\}$$

We partition the set $\mathcal{G}_t$ of gradients at timestamp t according to their Λ-value as follows.

$$\mathcal{G}_t = \bigsqcup_{\lambda \in \Lambda_t} \mathcal{G}_{t\lambda}$$
$$\mathcal{G}_{t\lambda} = \{[g, l] \in \mathcal{G}_t \mid \Lambda(\tau_{tl}) = \lambda\}$$

Therefore, the update equation can be reformulated as follows.

$$x_{t+1} = x_t - \gamma_t \sum_{\lambda \in \Lambda_t} \lambda \cdot \sum_{[G(x_l;\xi), l] \in \mathcal{G}_{t\lambda}} G(x_l; \xi)$$

Definition 6 (Adaptive learning rate)**.** *Given the Lipschitz constant K, the total number of timestamps T, and the total number of gradients in each timestamp as M, we define γ_t as follows.*

$$\gamma_t = \underbrace{\sqrt{\frac{Q(x_1) - Q(x^*)}{KTMd\sigma^2}}}_{\gamma} \cdot \underbrace{\frac{M}{\sum_{\lambda \in \Lambda_t} \lambda |\mathcal{G}_{t\lambda}|}}_{\mu_t} \tag{5.2}$$

where γ is the baseline component of the learning rate and μ_t is the adaptive component that depends on the amount of stale updates that the server receives at timestamp t. Moreover, μ_t incorporates the total staleness at any timestamp t based on the staleness coefficients (λ) associated with all the gradients received in timestamp t. $Q(x^*)$ (loss value at the optimum) can be assumed to be equal to zero.

Remark 2 (Correct cone)**.** *As a consequence of passing the filter and of Theorem 3, G satisfies the following.*

$$\langle \mathbb{E}_\xi G(x;\xi), \nabla Q(x) \rangle > \Omega((\|\nabla Q(x_t)\| - \sqrt{d}\sigma)\|\nabla Q(x_t)\|)$$

The theoretical guarantee for the convergence rate of Kardam depends on Assumptions 2,4 and Remark 2. These assumptions are weaker than the assumptions for the convergence guarantees in [183, 85]. In particular, due to unbounded delays and the potential presence of Byzantine workers, we only assume the unbiased gradient estimator $G(\cdot)$ for honest workers (Assumption 1). We instead employ (Remark 2) the fact that $G(\cdot)$ and $\nabla Q(x)$ make a lower bounded angle together (and subsequently a lower bounded scalar product) for all the workers. The classical unbiased assumption is more restrictive as it requires this angle to be exactly equal to 0, and the scalar product to be equal to $\|\nabla Q(x)\| \cdot \|G(x)\|$. Most importantly, we highlight the fact that those

[4] If $\Lambda(0) = 1$, then there is no decay for gradients computed on the latest version of the model, i.e., $\tau_{tl} = 0$.

assumptions are satisfied by Kardam, since every gradient used in this section to compute the ***Kar*** update has passed the Lipschitz filter of the previous section.

Theorem 4 (Convergence guarantee)**.** *We express the convergence guarantee in terms of the ergodic convergence, i.e., the weighted average of the $\mathscr{L}_2$ norm of all gradients ($||\nabla Q(x_t)||^2$). Using the above-mentioned assumptions, and the maximum adaptive rate $\mu_{\max} = \max\{\mu_1, \ldots, \mu_t\}$, we get the following bound on the convergence rate.*

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\|\nabla Q(x_t)\|^2 \leq \left(2+\mu_{\max}+\gamma KM\chi\mu_{\max}\right)\gamma Kd\sigma^2 + d\sigma^2 + 2DK\sigma\sqrt{d} + K^2D^2$$

under the prerequisite that

$$\sum_{\lambda\in\Lambda_t}\lambda^2|\Lambda_t|\Bigg\{K\gamma_t^2+\sum_{s=1}^{\infty}\Big(\sum_{\nu\in\Lambda_{t+s}}\gamma_{t+s}K^2\nu|\mathscr{G}_{t+s,\nu}|\Lambda^{-1}(\nu)\mathbb{I}_{(s\leq\Lambda^{-1}(\nu))}\gamma_t^2\Big)\Bigg\} \leq \sum_{\lambda\in\Lambda_t}\frac{\gamma_t\lambda}{|\mathscr{G}_{t\lambda}|} \tag{5.3}$$

where the Iverson indicator function is defined as follows.

$$\mathbb{I}_{(s\leq\Delta)} = \begin{cases} 1 & \text{if } s\leq\Delta \\ 0 & \text{otherwise.}\end{cases}$$

It is important to note that the prerequisite (Inequality 5.3) holds for any decay function Λ (since $\lambda < 1$ holds by definition) and for any standard learning rate schedule such that $\gamma_t < 1$. Various SGD approaches [183, 108, 180, 85] provide convergence guarantees with similar prerequisites.

Proof. We provide the convergence guarantee in terms of *ergodic convergence*—the weighted average of the $\mathscr{L}_2$ norm of all gradients ($||\nabla Q(x_t)||^2$). For the sake of clarity in the proofs, if X is a set, we also denote its cardinality by X.

Lemma 5. *Assume that, for all epochs $1 \leq t \leq T$*

$$\sum_{\lambda\in\Lambda_t}\left\{K\gamma_t^2|\Lambda_t|+\sum_{s=1}^{\infty}\sum_{\nu\in\Lambda_{t+s}}\gamma_{t+s}K^2\nu|\mathscr{G}_{t+s,\nu}|\Lambda^{-1}(\nu)\mathbb{I}_{(s\leq\Lambda^{-1}(\nu))}\gamma_t^2|\Lambda_t|\right\}\lambda^2 \leq \sum_{\lambda\in\Lambda_t}\frac{\gamma_t\lambda}{|\mathscr{G}_{t\lambda}|}$$

Then, the ergodic convergence rate is bounded as follows.

$$\frac{\sum_{t=1}^{T}\left(\gamma_t \sum_{\lambda\in\Lambda_t} \lambda\mathcal{G}_{t\lambda}\right)\mathbb{E}||\nabla Q(x_t)||^2}{\sum_{t=1}^{T}\gamma_t \sum_{\lambda\in\Lambda_t} \lambda\mathcal{G}_{t\lambda}} \leq \frac{2(Q(x_1)-Q(x^*))}{\sum_{t=1}^{T}\gamma_t \sum_{\lambda\in\Lambda_t} \lambda\mathcal{G}_{t\lambda}}$$
$$+\frac{\left(\sum_{t=1}^{T} K\gamma_t^2 \sum_{\lambda\in\Lambda_t} \lambda^2\mathcal{G}_{t\lambda} + \gamma_t K^2 \sum_{\lambda\in\Lambda_t} \lambda\mathcal{G}_{t\lambda} \sum_{j=t-\Lambda^{-1}(\lambda)}^{t-1} \gamma_j^2 \sum_{\lambda'\in\Lambda_j} \lambda'^2\mathcal{G}_{j\lambda'}\right)\cdot d\cdot\sigma^2}{\sum_{t=1}^{T}\gamma_t \sum_{\lambda\in\Lambda_t} \lambda\mathcal{G}_{t\lambda}}$$

Remark 3. *Given a list of vectors $u_1,\ldots,u_N$, we implicitly use the following inequality in our proof.*

$$\left\|\sum_{i=1}^{N} u_i\right\|^2 \leq N\cdot\sum_{i=1}^{N}\|u_i\|^2 \tag{5.3}$$

Proof. For the sake of concision, for every $m=[g,l]\in\mathcal{G}_{t\lambda}$, we denote by $\xi_{[t]}$ the set of ξ values that the server sends during epoch t. Let $\xi_{[t,*\neq m]}$ denote the set $\xi_{[t]}$ minus the variable ξ corresponding to message m. Additionally, $G[tm]\triangleq G(x_{t-\tau_{tl}};\xi)$ and $\nabla Q[tm]\triangleq\nabla Q(x_{t-\tau_{tl}})$.

A second order expansion of Q, followed by the application of the Lipschitz inequality to ∇Q yields the following.

$$Q(x_{t+1})-Q(x_t) \leq \langle\nabla Q(x_t), x_{t+1}-x_t\rangle + \frac{K}{2}\|x_{t+1}-x_t\|^2$$
$$\leq -\gamma_t\sum_{\Lambda_t}\lambda\mathcal{G}_{t\lambda}\langle\nabla Q(x_t), \frac{1}{\mathcal{G}_{t\lambda}}\sum_{\mathcal{G}_{t\lambda}} G[tm]\rangle + \frac{K}{2}\gamma_t^2\left\|\sum_{\Lambda_t}\lambda\sum_{\mathcal{G}_{t\lambda}} G[tm]\right\|^2$$

Taking the expectation and using the correct cone property, we have:

$$\mathbb{E}_{\xi_{[t]}}Q(x^{(t+1)})-Q(x_t) \leq -\gamma_t\sum_{\Lambda_t}\lambda\mathcal{G}_{t\lambda}\langle\nabla Q(x_t), \frac{1}{\mathcal{G}_{t\lambda}}\sum_{\mathcal{G}_{t\lambda}}\nabla Q[tm]\rangle$$
$$+\frac{K}{2}\gamma_t^2\mathbb{E}_{\xi_{[t]}}\left\|\sum_{\Lambda_t}\lambda\sum_{\mathcal{G}_{t\lambda}} G[tm]\right\|^2$$

Using $\langle a, b\rangle = \frac{||a||^2+||b||^2-||a-b||^2}{2}$, we obtain the following inequality.

$$\begin{aligned}
&\mathbb{E}_{\xi_{[t]}} Q(x_{t+1}) - Q(x_t) \leq -\frac{\gamma_t}{2}\sum_{\Lambda_t}\lambda\mathscr{G}_{t\lambda}\,\|\nabla Q(x_t)\|^2 \\
&-\frac{\gamma_t}{2}\sum_{\Lambda_t}\lambda\mathscr{G}_{t\lambda}\left\|\frac{1}{\mathscr{G}_{t\lambda}}\sum_{\mathscr{G}_{t\lambda}}\nabla Q[tm]\right\|^2 + \frac{K\gamma_t^2}{2}\underbrace{\mathbb{E}_{\xi_{[t]}}\left\|\sum_{\Lambda_t}\lambda\sum_{\mathscr{G}_{t\lambda}}G[tm]\right\|^2}_{S_1} \\
&+\frac{\gamma_t}{2}\sum_{\Lambda_t}\lambda\mathscr{G}_{t\lambda}\underbrace{\left\|\nabla Q(x_t) - \frac{1}{\mathscr{G}_{t\lambda}}\sum_{\mathscr{G}_{t\lambda}}\nabla Q[tm]\right\|^2}_{S_2}
\end{aligned}$$

We now define two terms S_1 and S_2 as follows.

$$\begin{aligned}
S_1 &= \mathbb{E}_{\xi_{[t]}}\left\|\sum_{\Lambda_t}\lambda\sum_{\mathscr{G}_{t\lambda}}(G[tm]-\nabla Q[tm]) + \sum_{\Lambda_t}\lambda\sum_{\mathscr{G}_{t\lambda}}\nabla Q[tm]\right\|^2 \\
&= \mathbb{E}_{\xi_{[t]}}\left\|\sum_{\Lambda_t}\lambda\sum_{\mathscr{G}_{t\lambda}}(G[tm]-\nabla Q[tm])\right\|^2 + \mathbb{E}_{\xi_{[m]}}\left\|\sum_{\Lambda_t}\lambda\sum_{\mathscr{G}_{t\lambda}}\nabla Q[tm]\right\|^2 \\
&+ 2\mathbb{E}_{\xi_{[t]}}\langle\sum_{\Lambda_t}\lambda\sum_{\mathscr{G}_{t\lambda}}(G[tm]-\nabla Q[tm]), \sum_{\Lambda_t}\lambda\sum_{\mathscr{G}_{t\lambda}}\nabla Q[tm]\rangle \\
&= \mathbb{E}_{\xi_{[t]}}\left\|\sum_{\Lambda_t}\lambda\sum_{\mathscr{G}_{t\lambda}}(G[tm]-\nabla Q[tm])\right\|^2 + \mathbb{E}_{\xi_{[t]}}\left\|\sum_{\Lambda_t}\lambda\sum_{\mathscr{G}_{t\lambda}}\nabla Q[tm]\right\|^2 \\
&+ 2\langle\sum_{\Lambda_t}\lambda\sum_{\mathscr{G}_{t\lambda}}(\nabla Q[tm]-\nabla Q[tm]), \sum_{\Lambda_t}\lambda\sum_{\mathscr{G}_{t\lambda}}\nabla Q[tm]\rangle \\
&= \underbrace{\mathbb{E}_{\xi_{[t]}}\left\|\sum_{\Lambda_t}\lambda\sum_{\mathscr{G}_{t\lambda}}(G[tm]-\nabla Q[tm])\right\|^2}_{A_1} + \underbrace{\mathbb{E}_{\xi_{[t]}}\left\|\sum_{\Lambda_t}\lambda\sum_{\mathscr{G}_{t\lambda}}\nabla Q[tm]\right\|^2}_{A_2}
\end{aligned}$$

Regarding A_2, applying Equation 5.3 yields the following inequality.

$$A_2 \leq \mathbb{E}_{\xi_{[t]}}\Lambda_t\cdot\sum_{\Lambda_t}\lambda^2\|\sum_{\mathscr{G}_{t\lambda}}\nabla Q[tm]\|^2 \leq \Lambda_t\cdot\sum_{\Lambda_t}\lambda^2\mathbb{E}_{\xi_{[t]}}\|\sum_{\mathscr{G}_{t\lambda}}\nabla Q[tm]\|^2$$

Regarding A_1, the term $\|\ldots\|^2$ is expressed as a scalar product and expanded as follows.

$$
\begin{aligned}
A_1 &= \mathbb{E}_{\xi_{[t]}} \sum_{\lambda,\lambda'\in\Lambda_t} \Big(\sum_{\substack{m\in\mathcal{G}_{t\lambda},\\ m'\in\mathcal{G}_{t\lambda'}}} \lambda\lambda' \cdot \langle G[tm]-\nabla Q[tm], G[tm']-\nabla Q[tm']\rangle \Big) \\
&= \text{diagonal } + \text{off-diagonal} \\
&= \sum_{\lambda\in\Lambda_t} \sum_{m\in\mathcal{G}_{t\lambda}} \lambda^2 \cdot \mathbb{E}_{\xi_{[t]}} \|G[tm]-\nabla Q[tm]\|^2 + \mathbb{E}_{\xi_{[t,m'\neq m]}} \big(\mathbb{E}_{\xi} \langle G[tm]-\nabla Q[tm], G[tm']-\nabla Q[tm']\rangle\big) \\
&\leq \sum_{\Lambda_t} \lambda^2 \mathcal{G}_{t\lambda} \cdot d \cdot \sigma^2 + d\cdot\sigma^2 + 2DK\sigma\sqrt{d} + K^2D^2
\end{aligned}
$$

The sum over the off-diagonal terms (i.e., $(\lambda, m) \neq (\lambda', m')$) is bounded by $d\cdot\sigma^2 + 2DK\sigma\sqrt{d} + K^2D^2$. Moreover, if $\lambda \neq \lambda'$, then $m \neq m'$ because $\mathcal{G}_{t\lambda}$ and $\mathcal{G}_{t\lambda'}$ are disjoint sets and thus for any off-diagonal pair $(\lambda, m), (\lambda, m')$ we have $m \neq m'$.

$$
\begin{aligned}
&\mathbb{E}_{\xi_{[t]}} \langle G[tm]-\nabla Q[tm], G[tm']-\nabla Q[tm']\rangle \\
&\quad= \mathbb{E}_{\xi_{[t,m'\neq m]}} \big(\mathbb{E}_{\xi} \langle G[tm]-\nabla Q[tm], G[tm']-\nabla Q[tm']\rangle\big) \\
&\quad= \mathbb{E}_{\xi_{[t,m'\neq m]}} \langle \mathbb{E}_{\xi} G[tm]-\nabla Q[tm], G[tm']-\nabla Q[tm']\rangle \\
&\quad= \mathbb{E}_{\xi_{[t,m'\neq m]}} (\langle \mathbb{E}_{\xi} G[tm], G[tm']\rangle - \langle \nabla Q[tm], G[tm']\rangle - \langle \mathbb{E}_{\xi} G[tm], \nabla Q[tm']\rangle + \langle \nabla Q[tm], \nabla Q[tm']\rangle) \\
&\quad\leq \mathbb{E}_{\xi_{[t,m'\neq m]}} (\|\mathbb{E}_{\xi} G[tm]\| \cdot \|G[tm']\| + \|\nabla Q[tm]\| \cdot \|G[tm']\| \\
&\qquad + \|\mathbb{E}_{\xi} G[tm]\| \cdot \|\nabla Q[tm']\| + \|\nabla Q[tm]\| \cdot \|\nabla Q[tm']\|) \\
&\quad\leq d\cdot\sigma^2 + 2DK\sigma\sqrt{d} + K^2D^2
\end{aligned}
$$

Hence, we obtain the following inequalities for S_1 and S_2.

$$
S_1 \leq \sum_{\Lambda_t} \lambda^2 \mathcal{G}_{t\lambda} \cdot d\cdot\sigma^2 + \Lambda_t \cdot \sum_{\Lambda_t} \lambda^2 \mathbb{E}_{\xi_{[t]}} \|\sum_{\mathcal{G}_{t\lambda}} \nabla Q[tm]\|^2 + d\cdot\sigma^2 + 2DK\sigma\sqrt{d} + K^2D^2
$$

$$
S_2 \leq \left\| \frac{1}{\mathcal{G}_{t\lambda}} \sum_{\mathcal{G}_{t\lambda}} \nabla Q(x_t) - \nabla Q[tm] \right\|^2
$$

Recall that, since $m = [g, l] \in \mathcal{G}_{t\lambda}$, we have $\nabla Q[tm] = \nabla Q(x_{t-\tau_{tl}})$. By applying the Lipschitz

inequality, we get:

$$
\begin{aligned}
S_2 &\leq K^2 \| x_t - x_{t-\Lambda^{-1}(\lambda)} \|^2 \\
&\leq K^2 \left\| \sum_{j=t-\Lambda^{-1}(\lambda)}^{t-1} x_{j+1} - x_j \right\|^2 \leq K^2 \left\| \sum_{j=t-\Lambda^{-1}(\lambda)}^{t-1} \gamma_j \sum_{\nu \in \Lambda_j} \nu \sum_{\mathscr{G}_{j\nu}} G[jm] \right\|^2 \\
&\leq K^2 \underbrace{\left\| \sum_{j=t-\Lambda^{-1}(\lambda)}^{t-1} \gamma_j \sum_{\nu \in \Lambda_j} \nu \sum_{\mathscr{G}_{j\nu}} (G[jm] - \nabla Q[jm]) \right\|^2}_{S_3 = \|a\|^2} \\
&+ K^2 \underbrace{\left\| \sum_{j=t-\Lambda^{-1}(\lambda)}^{t-1} \gamma_j \sum_{\nu \in \Lambda_j} \nu \sum_{\mathscr{G}_{j\nu}} \nabla Q[jm] \right\|^2}_{S_4 = \|b\|^2} + 2K^2 \langle a, b \rangle
\end{aligned}
$$

Hence, we obtain the following inequalities for S_3 and S_4.

$$
\begin{aligned}
\mathbb{E}_{\xi_{[j]},\ldots} S_3 &\leq \sum_{j=t-\Lambda^{-1}(\lambda)}^{t-1} \gamma_j^2 \sum_{\Lambda_j} \nu^2 \mathscr{G}_{j\nu} \cdot d \cdot \sigma^2 \quad \text{(cross-products vanish)} \\
\mathbb{E}_{\xi_{[j]},\ldots} S_4 &\leq \Lambda^{-1}(\lambda) \sum_{j=k-\Lambda^{-1}(\lambda)}^{t-1} \gamma_j^2 \Lambda_j \sum_{\Lambda_j} \nu^2 \mathbb{E} \left\| \sum_{\mathscr{G}_{j\nu}} \nabla Q[jm'] \right\|^2 \quad \text{(by Eq. 5.3).}
\end{aligned}
$$

Moreover, we have $\mathbb{E}_* \langle a, b \rangle = \langle \mathbb{E}_* a, b \rangle = 0$.

$$
\begin{aligned}
\mathbb{E} S_2 \leq K^2 &\sum_{j=t-\Lambda^{-1}(\lambda)}^{t-1} \gamma_j^2 \sum_{\Lambda_j} \nu^2 \mathscr{G}_{j\nu} \cdot d \cdot \sigma^2 \\
&+ K^2 \Lambda^{-1}(\lambda) \sum_{j=t-\Lambda^{-1}(\lambda)}^{t-1} \gamma_j^2 \Lambda_j \sum_{\Lambda_j} \nu^2 \mathbb{E} \left\| \sum_{\mathscr{G}_{j\nu}} \nabla Q[jm'] \right\|^2
\end{aligned}
$$

$$
\begin{aligned}
\mathbb{E}_{\xi_{[t]}} Q(x_{t+1}) - Q(x_t) \leq &- \frac{\gamma_t}{2} \sum_{\Lambda_t} \lambda \mathscr{G}_{t\lambda} \| \nabla Q(x_t) \|^2 \\
&+ \sum_{\Lambda_t} \left(\frac{K \gamma_t^2 \Lambda_t \lambda^2}{2} - \frac{\gamma_t \lambda}{2 \mathscr{G}_{t\lambda}} \right) \mathbb{E} \left\| \sum_{\mathscr{G}_{t\lambda}} \nabla Q[tm] \right\|^2 \\
&+ \left(\frac{K \gamma_t^2}{2} \sum_{\Lambda_t} \lambda^2 \mathscr{G}_{t\lambda} + \frac{\gamma_t K^2}{2} \sum_{\Lambda_t} \lambda \mathscr{G}_{t\lambda} \sum_{j=t-\Lambda^{-1}(\lambda)}^{t-1} \gamma_j^2 \sum_{\Lambda_j} \nu^2 \mathscr{G}_{j\nu} \right) \cdot d \cdot \sigma^2 \\
&+ \frac{\gamma_t K^2}{2} \sum_{\Lambda_t} \lambda \mathscr{G}_{t\lambda} \Lambda^{-1}(\lambda) \sum_{j=t-\Lambda^{-1}(\lambda)}^{t-1} \gamma_j^2 \Lambda_j \sum_{\Lambda_j} \nu^2 \mathbb{E} \left\| \sum_{j\nu} \nabla Q[jm'] \right\|^2
\end{aligned}
$$

Summing for $t = 1, \dots, T$, we arrive at the following inequality.

$$
\begin{aligned}
&\mathbb{E}Q(x_{t+1}) - Q(x_1) \leq -\sum_t \frac{1}{2}\left(\gamma_t \sum_{\Lambda_t} \lambda \mathcal{G}_{t\lambda}\right) \|\nabla Q(x_t)\|^2 \\
&+ \sum_t \left(\frac{K\gamma_t^2}{2} \sum_{\Lambda_t} \lambda^2 \mathcal{G}_{t\lambda} + \frac{\gamma_t K^2}{2} \sum_{\Lambda_t} \lambda \mathcal{G}_{t\lambda} \sum_{j=t-\Lambda^{-1}(\lambda)}^{t-1} \gamma_j^2 \sum_{\Lambda_j} \nu^2 \mathcal{G}_{j\nu}\right) \cdot d \cdot \sigma^2 \\
&+ \sum_t \sum_{\Lambda_t} \left(\frac{K\gamma_t^2 \Lambda_t \lambda^2}{2} - \frac{\gamma_t \lambda}{2\mathcal{G}_{t\lambda}}\right) \mathbb{E}\left\|\sum_{\mathcal{G}_{t\lambda}} \nabla Q[tm]\right\|^2 \\
&+ \sum_t \left(\sum_{s=1}^{\infty} \sum_{\Lambda_{t+s}} \gamma_{t+s} K^2 \nu \mathcal{G}_{t+s,\nu} \Lambda^{-1}(\nu) \mathbb{I}(s \leq \Lambda^{-1}(\nu))\right) \frac{\gamma_t \Lambda_t \lambda^2}{2} \mathbb{E}\left\|\sum_{\mathcal{G}_{t\lambda}} \nabla Q[tm]\right\|^2
\end{aligned}
$$

The last term comes from the following observation.

$$
\begin{aligned}
&\sum_{t=1}^{T} \sum_{\Lambda_t} \sum_{s=1}^{\infty} Q_t^{\lambda} Z_{t-s} \mathbb{I}(s \leq \Lambda^{-1}(\lambda)) = \sum_{s=1}^{\infty} \sum_{t=1}^{T} \sum_{\Lambda_t} Q_t^{\lambda} Z_{t-s} \mathbb{I}(s \leq \Lambda^{-1}(\lambda)) \\
&= \sum_{s=1}^{\infty} \sum_{l=1-s}^{T-s} \sum_{\Lambda_{l+s}} Q_{l+s}^{\lambda} Z_l \mathbb{I}(s \leq \Lambda^{-1}(\lambda)) = \sum_{s=1}^{\infty} \sum_{t=1}^{T} \sum_{\Lambda_{t+s}} Q_{t+s}^{\lambda} Z_t \mathbb{I}(s \leq \Lambda^{-1}(\lambda)) \\
&= \sum_{t=1}^{T} \left(\sum_{s=1}^{\infty} \sum_{\Lambda_{t+s}} Q_{t+s}^{\lambda} \mathbb{I}(s \leq \Lambda^{-1}(\lambda))\right) Z_t
\end{aligned}
$$

Since the two last terms sum to a non-positive value, we arrive at the following inequality.

$$
\begin{aligned}
&\sum_t \frac{1}{2}\left(\gamma_t \sum_{\Lambda_t} \lambda \mathcal{G}_{t\lambda}\right) \|\nabla Q(x_t)\|^2 \leq Q(x_1) - Q(x^*) \\
&+ \sum_t \left(\frac{K\gamma_t^2}{2} \sum_{\Lambda_t} \lambda^2 \mathcal{G}_{t\lambda} + \frac{\gamma_t K^2}{2} \sum_{\Lambda_t} \lambda \mathcal{G}_{t\lambda} \sum_{j=t-\Lambda^{-1}(\lambda)}^{t-1} \gamma_j^2 \sum_{\Lambda_j} \nu^2 \mathcal{G}_{j\nu}\right) \cdot d \cdot \sigma^2 + O\left(\frac{1}{K \cdot \sqrt{|\xi|}}\right)
\end{aligned}
$$

□

We first recall Definition 6, which introduces the adaptive learning rate schedule, before we prove Theorem 4 via employing Lemma 5. Due to the choice of the learning rate (Definition 6), the inequality in Theorem 4 reduces to the following inequality.

$$
\frac{1}{T} \sum_{t=1}^{T} \mathbb{E}\|Q(x_t)\|^2 \leq S_5 + S_6 + S_7
$$

First, we obtain the following equality for S_5.

$$S_5 = \frac{2(Q(x_1) - Q(x^*))}{\sum_{t=1}^{T} \gamma_t \sum_{\lambda \in \Lambda_t} \lambda \mathcal{G}_{t\lambda}} = \frac{2\gamma^2 KTM \cdot d \cdot \sigma^2}{\gamma TM} = 2\gamma K \cdot d \cdot \sigma^2$$

Regarding S_6, we obtain the following inequality.

$$\begin{aligned}
S_6 &= \frac{\sum_{t=1}^{T} K\gamma_t^2 \sum_{\lambda \in \Lambda_t} \lambda^2 \mathcal{G}_{t\lambda}}{\sum_{t=1}^{T} \gamma_t \sum_{\lambda \in \Lambda_t} \lambda \mathcal{G}_{t\lambda}} \cdot d \cdot \sigma^2 = \frac{K\gamma^2 \sum_{t=1}^{T} \mu_t^2 \sum_{\lambda \in \Lambda_t} \lambda^2 \mathcal{G}_{t\lambda}}{\gamma TM} \cdot d \cdot \sigma^2 \\
&\leq \frac{K\gamma^2 \sum_{t=1}^{T} \mu_t^2 \sum_{\lambda \in \Lambda_t} \lambda \mathcal{G}_{t\lambda}}{\gamma TM} \cdot d \cdot \sigma^2 \text{ (since } \lambda^2 \leq \lambda \leq 1) \\
&\leq \frac{K\gamma^2 TM\mu_{\max}}{\gamma TM} \cdot d \cdot \sigma^2 = \mu_{\max}\gamma K \cdot d \cdot \sigma^2
\end{aligned}$$

Finally, we obtain the following inequality for S_7.

$$\begin{aligned}
S_7 &= \frac{\sum_{t=1}^{T} \gamma_t K^2 \sum_{\lambda \in \Lambda_t} \lambda \mathcal{G}_{t\lambda} \sum_{j=t-\Lambda^{-1}(\lambda)}^{t-1} \gamma_j^2 \sum_{\lambda' \in \Lambda_j} \lambda'^2 M_{j\lambda'}}{\gamma TM} \cdot d \cdot \sigma^2 \\
&\leq \frac{K^2\gamma^3 \sum_{t=1}^{T} \mu_t \sum_{\lambda \in \Lambda_t} \lambda \mathcal{G}_{t\lambda} \sum_{j=t-\Lambda^{-1}(\lambda)}^{t-1} \mu_j^2 \sum_{\lambda' \in \Lambda_j} \lambda'^2 M_{j\lambda'}}{\gamma TM} \cdot d \cdot \sigma^2 \\
&\leq \frac{K^2\gamma^3 \sum_{t=1}^{T} \sum_{\lambda \in \Lambda_t} \lambda \mathcal{G}_{t\lambda} M\Lambda^{-1}(\lambda)\mu_{\max}}{\gamma TM} \cdot d \cdot \sigma^2 \\
&\leq \frac{K^2\gamma^3 \sum_{t=1}^{T} \sum_{\lambda \in \Lambda_t} \mathcal{G}_{t\lambda} M\chi\mu_{\max}}{\gamma TM} \cdot d \cdot \sigma^2 \leq \frac{K^2\gamma^3 TM^2\chi\mu_{\max}}{\gamma TM} \cdot d \cdot \sigma^2 \\
&\leq \gamma^2 K^2 M\chi\mu_{\max} \cdot d \cdot \sigma^2
\end{aligned}$$

Hence, we prove the ergodic convergence rate.

$$\frac{1}{T}\sum_{t=1}^{T} \mathbb{E}\|\nabla Q(x_t)\|^2 \leq \left(2 + \mu_{\max} + \gamma KM\chi\mu_{\max}\right) \cdot \gamma K \cdot d \cdot \sigma^2 + d \cdot \sigma^2 + 2DK\sigma\sqrt{d} + K^2D^2$$

□

Theorem 5 (Convergence time complexity)**.** *Given any mini-batch size $|\xi|$, the number of gradients M the server waits for before updating the model, and the total number of epochs T, the time complexity for the convergence of Kardam is:*

$$O\left(\frac{\mu_{\max}}{\sqrt{T|\xi|M}} + \frac{\chi\mu_{\max}}{T} + d\sigma^2 + 2DK\sigma\sqrt{d} + K^2D^2\right)$$

where χ denotes a constant such that for all τ_{tl}, the following inequality holds:

$$\tau_{tl} \cdot \Lambda(\tau_{tl}) \leq \chi \tag{5.4}$$

Theorem 5 highlights the relation between the staleness and the convergence time complexity. This time complexity is linearly dependent on the decay bound (χ) and the maximum adaptive rate (μ_{max}).

We now prove Theorem 5 by employing Theorem 4 along with Definition 6.

Proof. Substituting the value of γ from Definition 6 in RHS of Theorem 4, we get the following.

$$\begin{aligned}
&\left(2 + \mu_{\max} + \gamma K M \chi \mu_{\max}\right) \cdot \gamma K \cdot d \cdot \sigma^2 + d \cdot \sigma^2 + 2DK\sigma\sqrt{d} + K^2 D^2 \\
&= O\left(\frac{\mu_{\max}}{\sqrt{T \cdot |\xi| \cdot M}} + \frac{\chi \cdot \mu_{\max}}{T} + d \cdot \sigma^2 + 2DK\sigma\sqrt{d} + K^2 D^2\right)
\end{aligned}$$

Note that $\sigma = O(1/\sqrt{|\xi|})$ (Assumption 2) and therefore the bound is also dependent on n.

□

Remark 4 (Dampening comparison)**.** *Given two dampening functions $\Lambda_1(\tau) = \frac{1}{1+\tau}$ and $\Lambda_2(\tau) = exp(-\alpha \sqrt[\beta]{\tau})$, and the convergence time complexity from Theorem 5, $\Lambda_2(\tau)$ converges faster than $\Lambda_1(\tau)$ when $\frac{\beta}{e} < \alpha \leq \frac{ln(\tau+1)}{\sqrt[\beta]{\tau}}$.*

We also empirically highlight Remark 4 by comparing these two functions in Figure C.2 where DynSGD [85] employs Λ_1 and Kardam employs Λ_2.

Proof. (of Remark 4) From Inequality 5.4, we have the following for Λ_1 and Λ_2.

$$\begin{aligned}
\chi_1 &= \max_{\tau}\left\{\frac{\tau}{\tau+1}\right\} \\
\chi_2 &= \max_{\tau}\left\{\tau \cdot exp(-\alpha \sqrt[\beta]{\tau})\right\}
\end{aligned}$$

The maximum value of $\{\tau \cdot exp(-\alpha \sqrt[\beta]{\tau})\}$ is $\left(\frac{\beta}{e\alpha}\right)^{\beta}$ when $\tau = \left(\frac{\beta}{\alpha}\right)^{\beta}$. We get that $\chi_1 \geq \chi_2$ when the following holds.

$$\frac{\tau}{\tau+1} \geq \left(\frac{\beta}{e\alpha}\right)^{\beta}$$

Hence, from the above inequality, we get the following.

$$\tau \geq \frac{1}{\left(\frac{e\alpha}{\beta}\right)^{\beta} - 1}$$

Note that since $\tau > 0$, we get $\left(\frac{e\alpha}{\beta}\right)^{\beta} > 1$ which leads to the following lower bound on α.

$$\alpha > \frac{\beta}{e} \tag{5.5}$$

Furthermore, for the μ_{max} terms, we compare the values between the two dampening functions.

$$\mu_1 = \max_{\tau}\left\{\frac{M}{\sum_{\Lambda_t} \lambda \cdot |\mathcal{G}_{t\lambda}|}\right\} = \max_{\tau}\left\{\frac{M}{\sum_{\tau} \frac{1}{\tau+1} \cdot |\mathcal{G}_{t\lambda}|}\right\}$$
$$\mu_2 = \max_{\tau}\left\{\frac{M}{\sum_{\Lambda_t} \lambda \cdot |\mathcal{G}_{t\lambda}|}\right\} = \max_{\tau}\left\{\frac{M}{\sum_{\tau} exp(-\alpha \sqrt[\beta]{\tau}) \cdot |\mathcal{G}_{t\lambda}|}\right\}$$

Hence, for $\mu_1 \geq \mu_2$, we need to show that $\frac{1}{\tau+1} \leq exp(-\alpha \sqrt[\beta]{\tau})$, i.e., $\tau + 1 \geq exp(\alpha \sqrt[\beta]{\tau})$. The relation holds for any α with the upper bound as follows.

$$\alpha \leq \frac{ln(\tau+1)}{\sqrt[\beta]{\tau}} \tag{5.6}$$

From Inequalities 5.5 and 5.6, we get the following.

$$\frac{\beta}{e} < \alpha \leq \frac{ln(\tau+1)}{\sqrt[\beta]{\tau}}$$

One possible setting is $\beta \approx 1.85$ when $1 \leq \tau \leq 10$, $\beta \approx 3.1$ when $11 \leq \tau \leq 33$, and $\beta \approx 4$ when $34 \leq \tau \leq 75$. Given these values of β and τ, $\Lambda_2(\tau)$ has a smaller convergence time complexity (Theorem 5) than $\Lambda_1(\tau)$. Hence, $\Lambda_2(\tau)$ converges faster than $\Lambda_1(\tau)$.

□

5.4 Concluding Remarks

Kardam is, to the best of our knowledge, the first asynchronous distributed SGD algorithm that tolerates Byzantine behavior. In the following, we discuss papers that either address asynchrony or Byzantine behavior.

Asynchronous stochastic gradient descent. SGD is used widely in ML solutions due to its convergence guarantees with low time complexity per update, low memory cost, and robustness against noisy gradients. Several variants of SGD have been proposed to improve the convergence rate and the robustness against noise. Stale-synchronous parallel [79, 41] or bulk-synchronous [184, 28] variants typically target settings with limited staleness due to the limited performance variability among the computing devices. Other approaches consider variance minimization by importance sampling [3]. The theoretical guarantees underlying these approaches assume synchronous

updates as well as a specific formula to compute a gradient norm on each sample, which is only valid for multilayer perceptrons. The scheduler in [180] assumes all workers to be constantly available, which makes the algorithm not applicable to our setting with Byzantine workers. [85] recently introduced a stale-synchronous parallel (SSP) heterogeneity-aware algorithm. SSP algorithms assume bounded staleness while Kardam guarantees convergence without any such bound (i.e., asynchronous parallel). Additionally, Kardam provides the flexibility of choosing the appropriate dampening function according to the expected staleness distribution while being Byzantine tolerant and asynchronous. We show both theoretically (Remark 4) and empirically (Figure C.1) that an exponential dampening function leads to a faster convergence. [89] recently proposed an elegant optimizer to predict the optimal SGD variant based on the expected cost per iteration and the estimated number of iterations. This estimation does not however account for stale updates. Our convergence analysis for Kardam could be employed to estimate the number of iterations for different dampening functions and hence to predict the optimal staleness-aware SGD variant. (Lock-freedom) Kardam, put *before* lock-free solutions such as Hogwild [143] would not break the convergence requirements (since the purpose of Kardam is to preserve them despite Byzantine workers). However, Kardam and Hogwild do not commute.

Second order methods. These methods rely on computing the Hessian matrix instead of the Lipschitz factor (Kardam filtering component). They were not specifically designed for Byzantine resilience but can in fact be employed for that purpose. However, unlike our scalar-based Lipschitz filter ($\mathcal{O}(d)$ time complexity that is already within the usual cost of an SGD update), they suffer from the curse of dimensionality. Moreover, the parameter server does no less than $\Omega(d^2)$ verifications on the Hessian matrix or on the gradient covariance matrix. In the presence of a cheap (constant size K) heuristic, the parameter server will let the Byzantine worker with a margin of $d^2 - K$ open coordinates to use for an attack. Since $d \gg K$ the heuristic alternative clearly hampers Byzantine resilience.

The differentiability lenses of Lipschitz. A central piece of our work is to filter out suspected vectors based on their (lack of) similar Lipschitzness with the median behavior. We prove that this *filtering* idea is sound, given that a significant fraction ($\Omega(\frac{n-f}{n})$) of workers will almost surely pass it and that Byzantine workers passing it are not harmful. In fact, leveraging the Lipschitzness properties, in the differentiable context of gradient-based learning, is not an uncommon idea. It was used in different contexts, for example, to understand fine-grained robustness, i.e robustness of the model to internal errors at the level of neurons and/or weights, this was done in [49, 51, 55] proving a tight upper bound on the Lipschitz coefficient of neural networks, and deriving an exponential dependency with the depth and a polynomial dependency with the Lipschitz coefficient of the activation function used in each layer. In the same time, Lipschitzness was leveraged to compute spectral bounds as in [37, 13] both of which observed the same exponential dependency on the depth. In fact, manipulating differentiable objects is what makes the world of learning fundamentally different from the usual world of distributed computing, where the focus is on combinatorial and discrete structures. The differentiability of learning algorithms acts as a source of relaxation to solve a distributed computing task (estimating a gradient, distributively) in asynchrony and in the presence of Byzantine workers. The shorter the time it takes for Kardam

to self-stabilize (t_r) the better in term of the speed of convergence. As we prove in Theorem 3, t_r is shorter with a larger global Lipschitz coefficient, i.e., steeper cost functions. Nevertheless, the cost function cannot be controlled. Yet, t_r can be decreased by increasing the batch size per worker, which is no surprise in learning theory (increasing the batch size is one of the most unavoidable taxes [21, 26, 56] for increasing robustness). In practice, our experiments show no significant impact from t_r in the absence of actual Byzantine workers. In their presence, Kardam remains, to the best of our knowledge, the first provably Byzantine-resilient option to run SGD asynchronously.

An open problem is how to combine ideas from Bulyan and those from Kardam and secure asynchronous SGD despite very high dimensional models and strong adversaries. Finally, the Byzantine question remains unexplored in asynchronous machine learning *beyond* gradient-based algorithms. We argue[5] that the core idea we present –filtering on quantiles from the recent past– could have applications to any approach where updates arrive with suspicion on either staleness or malicious behavior.

[5] And we illustrate that in our ongoing work where the ideas of Kardam are applied on the *models* instead of the *gradients*:
El-Mahdi El-Mhamdi; Rachid Guerraoui and Arsany Guirguis (2019). Fast Byzantine Machine Learning with Unreliable Servers (under submission).

Robust Learning Machines Part II

6 Preliminaries

6.1 Robustness *Within* the Model

Up to this chapter, this thesis dealt with robustness from a coarse-grained perspective: the unit of failure is a worker, receiving its copy of the model and estimating gradients, based on either local data or delegated data from a server. The nature of the model itself is not important, the distributed system can be training models spanning a large range from simple regression to deep neural networks. As long as this training is using gradient-based learning, our algorithms to aggregate gradients, *Krum, Bulyan* and to filter them, *Kardam*, provably ensures convergence when a simple majority of nodes are not compromised by an attacker.

A natural question to consider is the fine-grained view: is the model itself robust to internal perturbations? In the case of neural networks, this question can somehow be tied to neuroscience considerations: could some neurons and/or synapses misbehave individually without harming the global outcome? We formulated this question in what follows and prove a tight upper bound on the resulting global error when a set of nodes is removed or is misbehaving [51]. One of the many practical consequences [55] of such fine-grained view is the understanding of memory cost reduction trade-offs in deep learning. Such memory cost reduction can be viewed as the introduction of precision errors at the level of each neuron and/or synapse [51].

Other approaches to robustness within the model tackled adversarial situations in machine learning with a focus on adversarial examples (during inference) [64, 166, 65] instead of adversarial gradients (during training) as we did for *Krum*. Robustness to adversarial input can be viewed through the fine-grained lens we introduced in [51], for instance, one can see perturbations of pixels in the inputs as perturbations of neurons in layer zero. It is important to note the orthogonality and complementarity between the fine-grained (model/input units) and the coarse-grained (gradient aggregation) approaches. Being robust, as a model, either to adversarial examples or to internal perturbations, does not necessarily imply robustness to adversarial gradients during training. Similarly, being distributively trained with a robust aggregation scheme such as *Krum* does not necessarily imply robustness to internal errors of the model or adversarial input perturbations that would occur later during inference. For instance, the theory we developed in the

previous chapters work is agnostic to the model being trained or the technology of the hardware hosting it, as long as there are gradients to be aggregated.

In the remainder, we stop being agnostic about the *learning machine* that is being trained, assume it to be a neural network and study its fine-grained robustness.

For didactic reasons, we present our results in an incremental manner. We consider first a network with a single layer and focus on the crashes of neurons, then we generalize to a multilayer network with Byzantine (arbitrary [100]) failures of neurons. We show that if the transmission capacity of synapses is unlimited, no neural network can tolerate the presence of a single Byzantine neuron. Inspired by results from biophysics [114] and neuroscience [6], we consider, however, synapses with a limited transmission capacity, and give a bound on the Byzantine failures of neurons as a function of this capacity. Finally we show how bounds on synapses failures can be derived from bounds on neurons failures.

We discuss several applications of our results. The first is a bound on the effect on output accuracy of reducing the computational precision per neuron. This effect has been recently highlighted experimentally [88], however, without any theoretical explanation. The second is a synchronization scheme that reduces the waiting time for neurons. The third is a quantification of the trade-off between robustness and ease of learning. We also discuss how to adapt our bounds to convolutional networks.

The rest of this part is organized as follows. In Section 6.2 we model a neural network as a distributed system and state the robustness criteria. In Section 7.1 we prove an upper bound on crash failures in the case of single-layer neural network. We use this proof as a starting point for the generalization to Byzantine failures and given in Section 7.2, first for neurons and then for synapses in the multilayer case. We discuss in Section 7.3 some applications of our results. We conclude the chapter by discussing other models and future work.

6.2 Model

6.2.1 Viewing a Neural Network as a Distributed System

We distinguish two main kinds of components of a neural network.

Neurons. These are the computing nodes (processes) of a neural network. They are either correct, in which case they execute their assigned computation (see below), or they fail, in which case they can stop computing (crash) or even send an arbitrary value (Byzantine faults). The failure of any neuron is independent from the failure of any other. Neurons communicate via message-passing [113] through synchronous point-to-point communication channels called *synapses*.

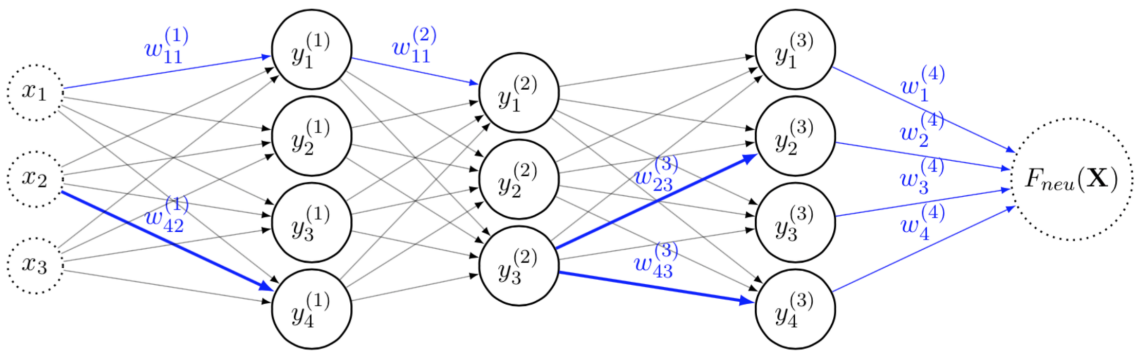


Figure 6.1 – A (feed forward) neural network (solid nodes and edges), with $d = 3$, $L = 3$, $N_2 = 3$ and $N_1 = N_3 = 4$. Input and output nodes (dotted) are not considered as parts of the network, but as its clients. For readability, only some synaptic weights are represented (bold blue). $\mathbf{X} = (x_1, \ldots, x_d)$.

Synapses. These are the communication channels connecting the neurons. Just like neurons, synapses are either correct, in which case they transmit the signal provided to them (see below) or fail, in which case they stop transmitting the signals, or they transmit arbitrary signals.The failure of a synapse is also independent from that of other synapses and neurons. Synapses are weighted[1]. The weight models the importance a neuron j gives to the signals emitted by a neuron i at the other end of the synapse and is therefore also called the weight from neuron i to neuron j. Faults at synapses can then be modeled as errors in the value of the weight: a *crashed synapse* is viewed as weighted by value 0 (stops transmitting), whereas a *Byzantine synapse* transmits any other value than the nominal value it is supposed to transmit, within its *capacity*.

Indeed, synapses have a *bounded transmission capacity*. This assumption is supported by two important works in biophysics [114] and neuroscience [6]. Hence if a faulty neuron corrupts the value it is supposed to send, the transmitted value is limited by the highest amount of electric charge flow the synapse can transport to the next neuron.

Hypothesis 1. *(Bounded transmission) There exists an upper bound $C \in \mathbb{R}^{*+}$ such that, for any input and any Byzantine neuron, the value transmitted by any synapse from that Byzantine neuron is bounded by C in absolute value.*

[1] The weights are determined by the initial learning phase, when training the network.

When Assumption 1 is not satisfied, we say that the network has *unbounded transmission*.

Neural Computation. The effectiveness of feed-forward neural networks relies on a fundamental theorem [81] that guarantees their universal approximating power with as few[2] as one single layer.

Let ϵ be any positive real number (an accuracy level), and F any continuous function mapping $[0,1]^d$ to $[0,1]$. The goal is to build an approximation of F with accuracy ϵ (as constructed in the classical model of a multilayer perceptron [77]) which we abstract in the following description:

Neurons are distributed over a series of layers. We denote by L the number of layers, each identified with index l and containing N_l neurons. Any neuron of layer $l-1$ is said to be *on the left* of any neuron of layer l (layer l is *on the right* of layer $l-1$). Each neuron *fires* (broadcasts) a signal (message) to all the neurons of the layer on its right. Neuron j at layer l receives the sum given by $s_j^{(l)}$ of Equation 6.1, where $y_i^{(l-1)}$ and $w_{ji}^{(l)}$ denote respectively the output value at neuron i of layer $l-1$, and the weight of the synapse from that same neuron to neuron j of the next layer l. To define its own output $y_j^{(l)}$, neuron j of layer l in turn injects the sum given by Equation 3 into a non-linear activation function, called a *squashing* function, φ, after adding a bias[3].

$$F_{neu}(\mathbf{X}) = \sum_{i=1}^{N_L} w_i^{(L+1)} y_i^{(L)}(\mathbf{X}) \tag{6.1}$$

$$\text{with } y_j^{(l)} = \varphi(s_j^{(l)})(l \geq 1);\ y_j^{(0)}(\mathbf{X}) = x_j \tag{6.2}$$

$$\text{and } s_j^{(l)} = \sum_{i=1}^{N_{l-1}} w_{ji}^{(l)} y_i^{(l-1)} \tag{6.3}$$

Definition 7. *(Approximation) We denote by $A = C([0,1]^d, [0,1])$ the space of continuous functions mapping $[0,1]^d$ to $[0,1]$. F_{neu} as defined by Equation 6.1 is said to be a neural ϵ-approximation of a target function $F \in A$ if we have: $\forall \boldsymbol{X} \in [0,1]^d$: $\|F(\boldsymbol{X}) - F_{neu}(\boldsymbol{X})\| \leq \epsilon$.*

Universality. We recall the universality theorem for a single layer network[4]: Let d be any integer and $\varphi : \mathbb{R} \to [0,1]$ a strictly-increasing continuous function, such that $\lim_{x\to-\infty}\varphi(x) = 0$ and $\lim_{x\to+\infty}\varphi(x) = 1$. Given any function $F \in A$ and $\epsilon > 0$, there exist an integer $N(\epsilon)$, and a set of coefficients $(w_{ji}^{(1)})_{1\leq j\leq N(\epsilon)}^{1\leq i\leq d}$ and $(w_i^{(2)})_{1\leq i\leq N(\epsilon)}$ such that F_{neu} defined in Equation 6.1 is a neural ϵ-approximation of F.

[2] Note that universality for L=1 is harder to obtain than for $L > 1$: fewer layers to approximate the target function.

[3] Following the usual notational convenience [77], we omit the bias in the computation model. This is done without loss of generality, considering an additional constant neuron (value = 1) in each layer. During the learning phase – when building the network – instead of learning its bias value, the neuron of layer l just learns the weight given to the constant neuron of layer $l-1$. As this weight is always multiplied by 1, the weight serves as the bias, around which the activation function is be centered.

[4] The interested reader can refer to the proof of [81].

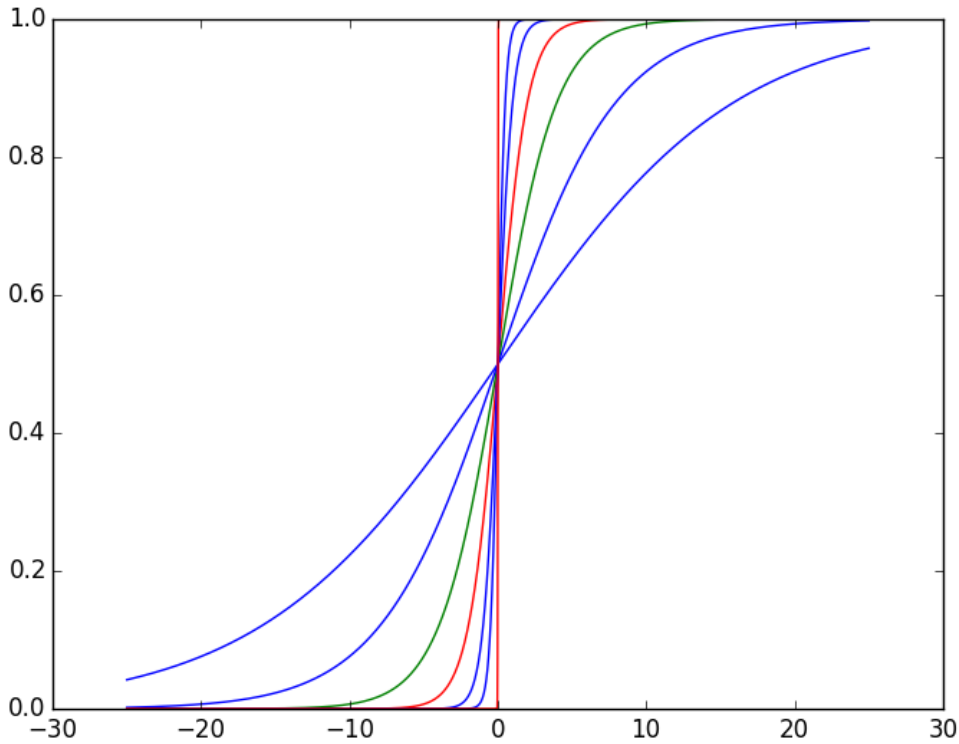


Figure 6.2 – The profile of a sigmoid function, centered around 0 and tuned with several values of K. The larger is K, the steeper is the slope and the more discriminating is the activation function at each neuron.

Activation Function. This function, denoted φ is the essence of the non-linearity of neural networks. The universality theorem holds for any non-constant, bounded and monotonically increasing activation function φ. Yet, two main popular choices for φ in machine learning applications are the logistic function *sigmoid* given by: $sigmoid(x) = \frac{1}{1+e^{-x}}$ and the hyperbolic tangent *tanh*.

In this chapter, we only impose on φ the conditions of the universality theorem, and consider that φ is K-*Lipschitzian*, meaning that $K = \sup|\frac{\varphi(x)-\varphi(y)}{x-y}|_{x\neq y}$ exists and is a finite real number.

Moreover, K can be tuned. Consider for instance the commonly used function, *sigmoid*, this function is $\frac{1}{4}$-Lipschitzian but can be tuned to be K-*Lipschitzian* (Figure 6.2) , by taking in this case $x \mapsto \varphi(4Kx)$ as the K-tuned activation function. (The detailed derivation of the Lipschizness of φ is given in a companion technical report [50].). In this chapter we consider, without loss of generality, *sigmoid* as the choice for φ.

6.2.2 Failures and Robustness

Definition 8. *(Failures) We say that a neuron i in layer l crashes when neuron i stops sending values, in which case $y_i^{(l)}$ is considered*[5] *to be equal to* 0 *by other neurons*[6]*. We say that neuron i is Byzantine, when $y_i^{(l)}$ is arbitrary.*

Definition 9. *(Robustness) We say that a neural ϵ-approximation F_{neu} of a target function F realized by N neurons tolerates N_{fail} faulty neurons, if for any subset of neurons $I_{fail} \subset \{1, \cdots, N\}$ of size N_{fail}, we can modify F_{neu} for the failing neurons according to Definition 8 and still ϵ-approximate F by F_{neu}.*

Lemma 6. *With unbounded transmission, no neural network can tolerate a single Byzantine neuron.*

Proof. As a consequence of the neural computation and definitions 8 and 9, if the transmission is unbounded, a Byzantine neuron at layer L sending a value higher than ϵ plus the difference between the nominal F_{neu} and the contribution of the remaining neurons breaks the ϵ-approximation as stated in Definition 7. □

6.2.3 Over-Provisioning

Using the universality theorem, we can define a minimal number of neurons $N_{min}(\epsilon)$ below which the neural network cannot yield an ϵ-approximation of F. By definition of $N_{min}(\epsilon)$, if a neural network is built with $N_{min}(\epsilon)$ neurons, the network cannot tolerate any crashed neuron.

Clearly, neural networks are not robust, they do not tolerate any neuron failure when built with the minimal amount of neurons. But, as we discussed in the introduction, this is usually not the case: they are over-provisioned [101] and contain more than $N_{min}(\epsilon)$ neurons. With the work of Barron [12], we know that $N_{min}(\epsilon) = \Theta(\frac{1}{\epsilon})$ and that given N neurons, a network can achieve an error in the order of $\frac{1}{N}$ when N is large. Instead of looking at over-provisioned networks as containing more than $N_{min}(\epsilon)$, we consider the quality of the approximation they are providing: given $\epsilon' \leq \epsilon$ and F_{neu} a neural ϵ'-approximation of F, F_{neu} is also a neural ϵ-approximation of F and is said to be an over-provisioned ϵ-approximation of F.

In the following, we set the conditions under which a neural network, realizing an ϵ'-approximation ($\epsilon' \leq \epsilon$) of F, can tolerate N_{fail} failures and keep realizing an ϵ-approximation of F. For convenience, all the bounds on the failures are stated in terms of ϵ and ϵ'.

[5] The strictly-increasing activation function φ does not allow a correct neuron to output value 0.
[6] Remember that we assume synchronous transmission.

7 Fault Tolerance in Neural Networks

7.1 Single-layer Neural Networks

For didactic reasons, we first start with the case of a single layer neural network. We translate the fact that the network tolerates the crash of N_{fail} neurons as given by Definition 9 to an inequation, which we combine with an estimation of the distance between the value of the damaged network and the nominal value of the output (which we recall is close to the target by a distance ϵ'). We end up with an upper bound on N_{fail}.

To prove that the bound is tight, we look at the worst failure case. Intuitively, this corresponds, following the tradition in distributed computing [113], to an adversary killing "key neurons": those with highest weights, and looking at an input were those same neurons were instrumental: broadcasting the highest possible value $y_j^{(1)}$, as close to 1 as possible.

Proposition 5. *Let F be any function mapping $[0,1]^d$ to $[0,1]$. Let ϵ and ϵ' be any two positive real numbers such that $0 < \epsilon' \leq \epsilon$. For any neural ϵ'-approximation F_{neu} of F (Definition 7) and any integer N_{fail}: If $N_{fail} \leq \frac{\epsilon-\epsilon'}{w_m}$ where $w_m = max(\|w_i^{(2)}\|, i \in [1,N])$ is the maximum norm of a weight from the single layer to the output node, then F_{neu} is a neural ϵ-approximation of F that tolerates N_{fail} crashed neurons (Definition 9). The bound on N_{fail} is tight.*

Proof. **Upper bound.** Applying the universality theorem[1] on F and ϵ', let F_{neu} be a neural ϵ'-approximation of F with N the number of neurons of F_{neu} and w_m the maximal weight from the single layer of F_{neu} to the output.

Let N_{fail} be any integer such that $N_{fail} \leq \frac{\epsilon-\epsilon'}{w_m}$. Denote by F_{fail} any of the modified values of the neural function F_{neu} after N_{fail} neurons crash: $F_{fail} = \sum_{i=1, i \notin I_{fail}}^{N} w_i^{(2)} y_i$, where I_{fail} is a a set

[1] The existence of a neural approximation for a given target function is taken here as granted by the universality theorem. One might wonder how do neurons, viewed as distributed processes, build the network (i.e put the correct weights to their linking synapses) in the first place to approximate that target function. This is done, during the learning phase, via the back-propagation algorithm [168]: neurons communicate in the reverse direction (from the output to the input) and re-adjust the weights *locally* according to the error value they are given by the output client.

containing N_{fail} crashed neurons. Let $\mathbf{X} \in [0,1]^d$ be any input vector. By the triangle inequality:

$$\begin{aligned} \|F(\mathbf{X}) - F_{fail}(\mathbf{X})\| \leq \|F(\mathbf{X}) - F_{neu}(\mathbf{X})\| \\ + \|F_{neu}(\mathbf{X}) - F_{fail}(\mathbf{X})\|. \end{aligned} \tag{7.1}$$

Since F_{neu} is an ϵ'-approximation of F we have:

$$\|F(\mathbf{X}) - F_{neu}(\mathbf{X})\| \leq \epsilon'. \tag{7.2}$$

From the definition of F_{fail}, we have: $\|F_{neu}(X) - F_{fail}(\mathbf{X})\| = \|\sum_{i=1, i\in I_{fail}}^{N} w_i^{(2)} y_i(\mathbf{X})\|$. Using another triangle inequality on norms we get:

$$\|F_{neu}(\mathbf{X}) - F_{fail}(\mathbf{X})\| \leq \sum_{i=1, i\in I_{fail}}^{N} \|w_i^{(2)}\| y_i(\mathbf{X}). \tag{7.3}$$

By definition of w_m and the hypothesis on the bounded activation function, $\|w_i^{(2)}\| \leq w_m$ and $y_i(\mathbf{X}) \leq 1$ for all $\mathbf{X}$ and i. Inequality 7.3 becomes:

$$\|F_{neu}(\mathbf{X}) - F_{fail}(\mathbf{X})\| \leq \sum_{i=1, i\in I_{fail}}^{N} w_m = N_{fail} w_m \tag{7.4}$$

Merging inequalities 7.1, 7.2 and 7.4 we obtain: $\|F(\mathbf{X}) - F_{fail}(\mathbf{X})\| \leq N_{fail}.w_m + \epsilon'$.

Since $N_{fail} \leq \frac{\epsilon-\epsilon'}{w_m}$, we have $\|F(\mathbf{X}) - F_{fail}(\mathbf{X})\| \leq \epsilon$.

Therefore the upper bound on N_{fail} : $N_{fail} \leq \frac{\epsilon-\epsilon'}{w_m}$, guarantees that F_{fail}, the neural function obtained from F_{neu} with N_{fail} crashed neurons is still an ϵ-approximation of F.

Tightness. Let N_{fail} be any integer such that $N_{fail} > \frac{\epsilon-\epsilon'}{w_m}$ and assume that F_{neu} tolerates the crash of N_{fail}. Let $\delta = N_{fail} - \frac{\epsilon-\epsilon'}{w_m}$, by the initial assumption, $\delta > 0$.

Consider ϵ' to be the supremum on the approximation with which the over-provisioned neural network F_{neu} approximates F, i.e $\epsilon' = sup_{\mathbf{X}\in[0,1]^d}(\|F(\mathbf{X}) - F_{neu}(\mathbf{X})\|)$.

Consider the equality cases as well as the limit cases (close to equality) for the key inequalities that lead to the upper bound on N_{fail}: In 7.1, equality occurs iff $F(\mathbf{X}) - F_{neu}(\mathbf{X})$ and $F_{neu}(\mathbf{X}) - F_{fail}(\mathbf{X})$ are positively proportional (equality case of the triangle inequality). In 7.3, equality occurs iff the weights of the crashed neurons are positively proportional. Assume an input and choice of crashed neurons satisfying both of these equality cases.

Let α be any positive real number. To be close to the limit case of Inequality 7.2, we can chose

inputs $\mathbf{X}$ such that $\|F(X) - F_{neu}(X)\| > \epsilon' - \frac{\alpha}{2}$, those inputs exist otherwise ϵ' is not the supremum error achieved by F_{neu} or F is not a continuous function and we will have a contradiction.

In 7.4, the limit case corresponds to crashed neurons being those with maximal weights and inputs such that the neurons in I_{crash} all output a value close to 1: let $\mathbf{X}$ be an input satisfying the previous equality and limit cases such that for any neuron i in I_{fail}, we have $y_i(\mathbf{X}) > max(1 - \frac{\alpha}{2}, 1 - \frac{\alpha}{2(\epsilon-\epsilon')})$, i.e $y_i(\mathbf{X})$ close to 1. With this worst-case choice of input and crashed neurons, we obtain: $\|F(\mathbf{X}) - F_{fail}(\mathbf{X})\| = \|F(X) - F_{neu}(X)\| + \|\sum_{i=1, i\in I_{fail}}^{N} w_i^{(2)} y_i(\mathbf{X})\| > \epsilon' - \frac{\alpha}{2} + max(1 - \frac{\alpha}{2}, 1 - \frac{\alpha}{2(\epsilon-\epsilon')}) N_{fail}.w_m$.

Thus, in case of more crashes than allowed by the upper bound of the proposition ($N_{fail} > \frac{\epsilon-\epsilon'}{w_m}$) leads to:

$\|F(\mathbf{X}) - F_{fail}(\mathbf{X})\| > \epsilon' - \frac{\alpha}{2} + max(1 - \frac{\alpha}{2}, 1 - \frac{\alpha}{2(\epsilon-\epsilon')})(\epsilon - \epsilon' + \delta w_m) = \epsilon - \frac{\alpha}{2} + max(-\frac{\alpha}{2}, -\frac{\alpha}{2(\epsilon-\epsilon')})(\epsilon - \epsilon' + \delta w_m) + \delta w_m = \epsilon - \frac{\alpha}{2} - min(\frac{\alpha}{2}, \frac{\alpha}{2(\epsilon-\epsilon')})(\epsilon - \epsilon' + \delta w_m) + \delta w_m$.

If $\epsilon - \epsilon' \geq 1$ then $min(\frac{\alpha}{2}, \frac{\alpha}{2(\epsilon-\epsilon')}) = \frac{\alpha}{2(\epsilon-\epsilon')})$ and the latter inequality leads to $\|F(\mathbf{X}) - F_{fail}(\mathbf{X})\| > \epsilon - \frac{\alpha}{2} - \frac{\alpha}{2(\epsilon-\epsilon')}(\epsilon - \epsilon' + \delta w_m) + \delta w_m = \epsilon - \alpha + \delta'$, where $\delta' = \delta w_m(1 - \frac{\alpha}{2(\epsilon-\epsilon')}) > 0$ (for small α).

If $\epsilon - \epsilon' < 1$ then $min(\frac{\alpha}{2}, \frac{\alpha}{2(\epsilon-\epsilon')}) = \frac{\alpha}{2}$ and the latter inequality leads to $\|F(\mathbf{X}) - F_{fail}(\mathbf{X})\| > \epsilon - \frac{\alpha}{2} - \frac{\alpha}{2}(\epsilon - \epsilon' + \delta w_m) + \delta w_m = \epsilon - \frac{\alpha}{2}(1 + (\epsilon - \epsilon' + \delta w_m)) + \delta w_m$ and since $\epsilon - \epsilon' + \delta w_m > 0$ this implies that $\|F(\mathbf{X}) - F_{fail}(\mathbf{X})\| > \epsilon - \frac{\alpha}{2} + \delta w_m$, which also leads to:

$\|F(\mathbf{X}) - F_{fail}(\mathbf{X})\| > \epsilon - \alpha + \delta'$ ($\delta' < \delta w_m$ for small α and $-\alpha < -\frac{\alpha}{2}$) since $\alpha > 0$).

In the inequality $\|F(\mathbf{X}) - F_{fail}(\mathbf{X})\| > \epsilon - \alpha + \delta'$.

α is any positive real number, for which we can chose an input leading to the inequality. Since F and F_{fail} are continuous, we can take the previous inequality to the limit $\alpha \mapsto 0$ and we abtain an inequality that contradict the assumption that F_{neu} tolerates the crash of N_{fail} neurons.

Therefore, by contradiction, the bound is tight. □

7.2 Multilayer Networks and Byzantine Failures

This section generalizes Proposition 5. While that proposition says that we can derive a tight bound on how many neurons can crash without losing ϵ-accuracy, it does not capture the situation where neurons can send values different from those expected, whether this difference is arbitrary or controlled. The latter situation is that of correct neurons in a multilayer network: if a correct neuron has faulty neurons on its left[2], the output value of this neuron embeds some imprecision. The aim is to evaluate how the loss of accuracy propagates through layers and bound it on the output.

[2]The conventions of *left* and *right* are defined in the neural computation described in Section 6.2.1.

7.2.1 Forward Error Propagation

Proposition 6 below says that, when errors occur at f_l neurons of layer l, the effect is transmitted by all correct neurons at any layer l' between layer l and the output. This leads, in the worst case, to a series of multiplications, as many times as there are layers on the right before reaching the output, i.e $(L-l)$ times, by the Lipschitz constant, by the number of correct neurons at layer l', i.e $(N_{l'} - f_l')$, by the maximum weight $w_m^{(l')}$, by f_l and by the bound C of Assumption 1. The previous products are summed over the layers. As a calculation convention for the rest of the chapter, we consider an $(L+1)$-th layer consisting of the output node with $N_{L+1} = 1$ correct neuron and $f_{L+1} = 0$ failing neurons (though it is not part of the neural network, unlike the $(L+1)$-th sets of synapses which are part of the network). Finally, the effect of a failure on the output increases exponentially with the depth of the layer (dependency on K^{L-l}).

Denote by Fep the quantity described above and given by the following equation:

$$Fep = C \sum_{l=1}^{L} \left(f_l K^{L-l} w_m^{(L+1)} \prod_{l'=l+1}^{L} (N_{l'} - f_{l'}) w_m^{(l')} \right).$$

Note that Fep has a polynomial dependency on K as observed in Figure 7.1.

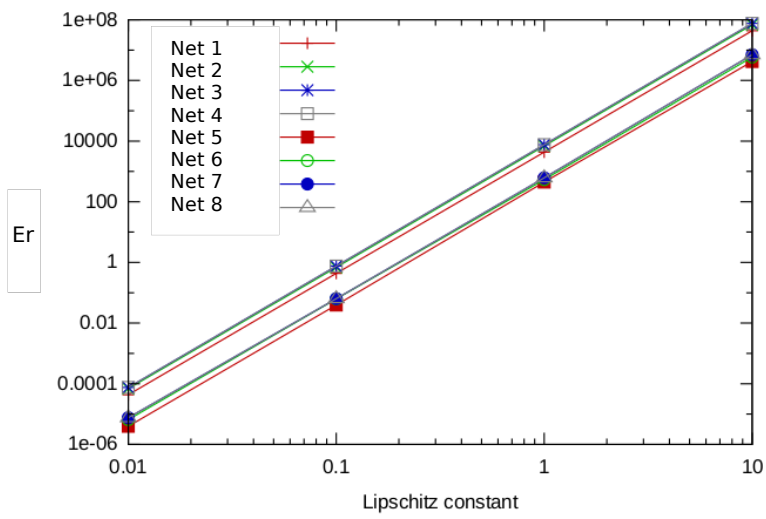


Figure 7.1 – Experimental values of the error (Er) at the output of several neural networks, affected with similar amount of neuron failures, plotted against the Lipschitz constant in a log scale.

Proposition 6. *Consider a neural network containing L layers. If in each layer l, f_l neurons, among the N_l neurons, are affected by errors such that any neuron j within layer l broadcasts an output $y_j^{(l)} + \lambda_j^l$ to the next layer instead of the nominal $y_j^{(l)}$, then the effect on the output is bounded as*

follows:

$$\|F_{neu}(\boldsymbol{X}) - F_{\lambda}(\boldsymbol{X})\| \leq Fep \tag{7.5}$$

where F_{neu} is the nominal neural function, F_{λ} the neural function accounting for the errors $\lambda_j^{(l)}$, and $w_m^{(l)} = max(|w_{ji}^{(l)}|, (j,i) \in [1, N_l][1, N_{l-1}])$ is the maximum norm of the weights of the incoming synapses to layer l. The bound (7.5) is tight.

Proof. We proceed by induction on L.

Initiation.

Let $N_{fail} = f_1$ be the number of neurons failing in the single layer of the network, let I_{fail} be the set containing those neurons, we have:

$\|F_{neu}(\mathbf{X}) - F_{\lambda}(\mathbf{X})\| = \|\sum_{i \in I_{fail}} w_i^{(2)}(y_i^{(1)} + \lambda_i^{(1)})\|$

Which, by the triangle inequality leads to:

$\|F_{neu}(\mathbf{X}) - F_{\lambda}(\mathbf{X})\| \leq \sum_{i \in I_{fail}} \|w_i^{(2)}(y_i^{(1)} + \lambda_i^{(1)})\|$, equality cases occur for positively proportional terms (**Condition 1**). Applying Assumption 1 and the definition of $w_m(2)$ gives us:

$$\|F_{neu}(\mathbf{X}) - F_{\lambda}(\mathbf{X})\| \leq f_1 w_m^{(2)} C \tag{7.6}$$

Equality cases occur for inputs such that $y_i^{(1)} + \lambda_i^{(1)} = C$ (**Condition 2**) and when the failing neurons are all linked to the output with the maximal weight $w_m^{(2)}$ (**Condition 3**).

We observe that Inequation 7.6 is the $(L = 1)$ version of Proposition 2, and that similarly, due to the worst case of failures (i.e when Conditions 1 to 3 are simultaneously occurring), the bound is tight.

Induction step.

Assume Proposition 6 holds for networks with up to some number of layers $L \geq 1$.

Now consider a network consisting of $(L+1)$ layers. The layered structure of the network enables us to see each of the N_{L+1} neurons of the $(L+1)^{th}$ layer, first as an output to an L-layer network (all the nodes to the left of that neuron), and second, after applying the activation function, as a neuron in a single-layer neural network (consisting of the $(L+1)^{th}$ layer alone).

In this last $(L+1)^{th}$ layer, we can distinguish two subsets of neurons:

1. (Failing neurons at layer L+1) A subset of f_{L+1} failing neurons, that yields, as in the initiation step (sigle layer), an error of at most $f_{L+1} w_m^{(L+2)} C$.

2. (Correct neurons at layer L+1) A subset of $N_{L+1} - f_{L+1}$ correct neurons. Those neurons transmit to the output side (their right side), in addition to their nominal value, the error E of the L-layer neural network *on the left of layer (L+1)*, multiplying it by at most the maximum synaptic weight from layer L to layer $(L+1)$, $w_m^{(L+1)}$ and the Lipschitz constant K, yielding an error of at most $E(N_{L+1} - f_{L+1})K$.

By the induction hypothesis we have:

$E \leq C \sum_{l=1}^{L} f_l K^{L-l} \prod_{l'=l+1}^{L+1} (N_{l'} - f'_l) w_m^{(l')}$

As the output node is linear (not part of the neural network and not performing any non-linear activation function), the errors mentioned in 1 and 2 are added and yield a total error bounded as follows:

$$\begin{aligned} \|F_{neu}(\mathbf{X}) - F_\lambda(\mathbf{X})\| &\leq f_{L+1} w_m^{(L+2)} C + (N_{L+1} - f_{L+1}) K E \\ &\leq C \sum_{l=1}^{L+1} f_l K^{L+1-l} \prod_{l'=l+1}^{L+2} (N_{l'} - f'_l) w_m^{(l')} \end{aligned}$$

which is the desired bound for an $(L+1)$-layer network. The equality case follows from considering the inter-occurrence of the equality cases at all the contributing parts, in case no constraint on the network is set to avoid it.

By induction, Proposition 6 is true for any integer $L \geq 1$. □

7.2.2 Tight Bound on Neuron Failures

With the notations used before, in a network of L layers, each layer l containing N_l neurons, we consider $N_{fail} = (f_l)_{l=1}^L$ as the distribution per layer of Byzantine neurons (f_l being the contribution of layer l to N_{fail}). Using Proposition 6, and the same reasoning as in the proof of Proposition 5, it is possible to derive a tight bound, not on the total number of failures as in single layer case, but on the failures per layer distribution[3] $N_{fail} = (f_l)_{l=1}^L$.

Corollary 3. *Let F be any continuous function mapping $[0,1]^d$ to $[0,1]$, let ϵ and ϵ' be any two positive real numbers such that $0 < \epsilon' \leq \epsilon$. Given any F_{neu} that is an L-layer neural ϵ'-approximation of F with N_l neurons per layer l, given $N_{fail} = (f_l)_{l=1}^L$ such that $\forall l\, f_l < N_l$ and*

$$Fep \leq \epsilon - \epsilon'. \tag{7.7}$$

[3] We use the natural extension of Definition 9 to this generalization from an integer N_{fail} to an L-tuple of failures per layer.

Then F_{neu} tolerates the distribution of Byzantine neurons N_{fail}. The bound (7.7) is tight.

Proof. Denote by F_{fail} the output of the network after N_{fail} failures, using Proposition 6 we have: $\|F_{neu}(\mathbf{X}) - F_{fail}(\mathbf{X})\| \leq C \sum_{l=1}^{L} f_l K^{L-l} \prod_{l'=l+1}^{L+1} (N_{l'} - f'_l) w_m^{(l')}$. Combining this with inequalities 7.1 and 7.2 we obtain: $\|F(\mathbf{X}) - F_{fail}(\mathbf{X})\| \leq \epsilon' + C \sum_{l=1}^{L} f_l K^{L-l} \prod_{l'=l+1}^{L+1} (N_{l'} - f'_l) w_m^{(l')}$. If N_{fail} satisfies inequality 7.7 then we have $\|F(\mathbf{X}) - F_{fail}(\mathbf{X})\| \leq \epsilon$. This proves the upper bound. Tightness follows the worst case reasoning on the equality and limit cases as done in the proof of Proposition 5. □

To better appreciate the message of Proposition 3, one has to bare in mind that the left-hand side of Equation 7 comes from the *forward error propagation* due to the failure distribution N_{fail} (Proposition 2) while the right-hand side is the the maximal error permitted by the over-provision.

Note that, in Proposition 6, small K and small weights reduce the propagating error Fep, which translates in Proposition 3 to: the smaller the K and the weights, the easier it is to satisfy the condition with large f_l. This sets the basis for the trade-off on tuning K or reducing the weights, as we stated in the introduction and as we discuss in the last section. Note also that in the case of crashes without Byzantine neurons, Assumption 1 is not necessary and C can be replaced by the maximum of the activation function (1 in case of $sigmoid$), which is the maximum value a neuron can send. Note finally that Lemma 2 can also be derived as a limit case of Proposition 3: $N_{fail} \xrightarrow{C \to \infty} 0$.

7.2.3 The Failure of Synapses

The following lemma links errors at synapses to errors at neurons. Again we use the convention that layer $L+1$ corresponds to the output node, in addition to the convention that layer 0 corresponds to input nodes.

Lemma 7. *In any L-layer neural network, an error of value $\lambda_{ji}^{(l)}$ at the synapse from neuron i of layer $l-1$ to neuron j of layer l is at worst, equivalent to an error at neuron i of value $C.K$.*

Proof. Let l be a layer in the neural network, and let i and j be any neurons from $l-1$ and l respectively.

An error of value $\lambda_{ji}^{(l)}$ in the synapse from neuron i to neuron j yields a received sum at neuron j, noted $s_{\lambda,j}^{(l)}$ and given by Equation 6.1 as follows:

$$\begin{aligned} s_{\lambda,j}^{(l)} &= \sum_{k=1,k\neq i}^{N_{l-1}} w_{jk}^{(l)} y_k^{(l-1)} + w_{ji}^{(l)} y_i^{(l-1)} + \lambda_{ji}^{(l)} \\ &= \sum_{k=1}^{N_{l-1}} w_{jk}^{(l)} y_k^{(l-1)} + \lambda_{ji}^{(l)} \end{aligned}$$

Therefore, by K-Lipschitzness of the activation function, the output error of neuron j is bounded as follows:

$$|error| = |\varphi(s^{(l)}_{\lambda,j}) - \varphi(s^{(l)}_j)| \leq K.|s^{(l)}_{\lambda,j} - s^{(l)}_j| = K.|\lambda^{(l)}_{ji}|$$

In the worst case the transmitted error $|\lambda^{(l)}_{ji}|$ is equal to C following Assumption 1 and the bound $|error| \leq C.K$ is tight.

□

Corollary 4. *Given $N_{fail} = (f_l)_{l=1}^{L+1}$, the distribution of Byzantine synapses, with f_l being the number of failing ones linking layer $l-1$ to layer l: If $C \sum_{l=1}^{L+1} f_l K^{L+1-l} w^{(l)}_m \prod_{l'=l+1}^{L+1} (N_{l'} - f'_l) w^{(l')}_m \leq \epsilon - \epsilon'$ then F_{neu} tolerates the distribution of Byzantine synapses $N_{fail} = (f_l)_{l=1}^{L+1}$. This bound is tight.*

Proof. Lemma 7 implies that the failure of a distribution N_{fail} of synapses in an L-layer network is equivalent, in the worst case, to the failure of a distribution N_{fail} of neurons in an $L+1$ network. Applying Proposition 3, the result follows. □

7.2.4 Reduced Over-provisioning

Our condition, under which over-provisioned networks can be robust (Proposition 3), concerns networks reaching a precision ϵ' finer than the one they are required to keep ϵ (i.e $\epsilon' < \epsilon$). One can wonder how hard it is to reach ϵ' (i.e, how precise should the over-provisioned network be to tolerate N_{fail}). The following corollary establishes the feasibility of building robust networks that can be arbitrarily close to non robust ones ($\epsilon' - \epsilon$ being arbitrarily small).

Corollary 5. *Let N_{fail} be any set of L integers as described in 7.2.2, $\epsilon > 0$ any precision level and F any target function. Then for every $\epsilon' > 0$ such that $\epsilon' < \epsilon$, there exist a neural network approximating F with precision ϵ' and preserving precision ϵ under failure distribution N_{fail}.*

Proof. The existence of a network ϵ'-approximating F is guaranteed by the universal approximation theorem [81] applied to ϵ' and F. Let $w^{(l)}_m$ be the maximal weights at each layer of this network, for the robustness constraint, let $(N_l)_{1\leq l\leq L}$ be any set of integers, large enough such that the condition of Proposition 3 is satisfied with N_{fail}, $w^{(l)}_m$ and $\epsilon - \epsilon'$. Then, following Proposition 3, this network is an ϵ-approximation of F that tolerates the failure distribution N_{fail}. □

7.3 Applications

7.3.1 Reducing Memory Cost

When implementing neural networks in hardware, reducing memory cost typically goes with reducing the precision with which each neuron performs its local computation. However, reducing

this local precision impacts accuracy. Recently, experimental results [88] reported interesting trade-offs between cost reduction and accuracy of the output. We provide here the first theoretical result quantifying those trade-offs.

In the case of a neural network containing L layers where the cost reduction implies a maximum error of λ_l per layer l, the accuracy degradation in the output is bounded by a sum similar to what we give in Proposition 3. This application is not a direct consequence of Proposition 3 but can be more specifically derived from the observations made in the proof of Proposition 6, in which we replace the uniform bound C on the transmission capacity by local bounds λ_l per layer. We get the following proposition.

Proposition 7. *If in each layer l, the implementation induces an error at each neuron j of layer l bounded by λ_l, then the effect on the output is bounded as follows:*

$$\|F_{neu}(\boldsymbol{X}) - F_\lambda(\boldsymbol{X})\| \leq \sum_{l=1}^{L} K^{L-l} \lambda_l \prod_{l'=l}^{L} N_{l'} w_m^{(l'+1)} \tag{7.8}$$

where F_{neu} is the nominal neural function, F_λ the neural function accounting for the errors λ_l, $w_m^{(l)} = max(\|w_{ji}^{(l)}\|, (j,i) \in [1, N_l][1, N_{l-1}])$ the maximum norm of the weights of the incoming synapses to layer l, and K the Lipschitz coefficient of the activation function. Inequality 7.8 is tight.

Proof. The proof is similar to that of Proposition 6. We proceed by induction on L, the number of layers in a neural network.

Initiation. In a single-layer neural network with N neurons, if each neuron i introducing an error λ_i, and all errors bounded by C, then the difference between the nominal output and the output affected by errors is given by $\sum_{i=1}^{N} \lambda_i w_i^{(2)}$ where $w_i^{(2)}$ is the weight from neuron i to the output, therefore the total error is bounded by $NCw_m^{(2)}$ where $w_m^{(2)}$ is the maximum weight from the single layer to the output. This is the base case ($L = 1$) of our induction.

Induction step. Let L be an integer such that we have the result of Inequality 7.8 for every network of L layers. Let $E_L = \sum_{l=1}^{L} K^{L-l} \lambda_l \prod_{l'=l}^{L} N_{l'} w_m^{(l'+1)}$.

Consider now a network of $L+1$ layers. As for the proof of Proposition 6, every neuron i of layer $L+1$ is the output of an L layer network. Neuron i therefore receives a sum affected by an error of at most E_L to which it adds its own error λ_i^{L+1}, and applies the activation function which multiplies the total error by at most K.

The total error at each neuron i is therefore bounded by $KE_l + K\lambda_{L+1}$. Applying the base case at the final output of the network, we bound the final error by $N_{L+1} w_m^{(L+2)} (KE_l + K\lambda_{L+1})$ which is equal to $\sum_{l=1}^{L+1} K^{L-l} \lambda_l \prod_{l'=l}^{L+1} N_{l'} w_m^{(l'+1)}$ and proves the induction. □

7.3.2 Boosting Computations

Consider a network where neurons do not have the same reactive speed to inputs, but can be reset instantly to ignore their actual computation. Each time a neuron receives a *sufficient* amount of information from its preceding input layer, it sends a reset to the slow neurons (in the preceding layer) instead of waiting for their values and move on with its own computation, adopting value 0 for the slow neurons. Proposition 3 gives a sense of that *sufficient* amount of information from which we derive the following corollary.

Corollary 6. *Following the notation of Proposition 3 in the crash case ($C = 1$ as explained in 7.2.2), If F_{neu} is an ϵ'-approximation of F, then given any family of integers f_l satisfying the conditions of Proposition 3, then each neuron of layer l has to wait only for $N_{l-1} - f_{l-1}$ signals from layer $l-1$ to send a value to layer $l+1$, as well as a reset to the missing neurons at layer $l-1$, while guaranteeing a correct ϵ-approximation of F at the output.*

Proof. The corollary is a direct consequence of Proposition 3. □

7.3.3 Balancing Robustness and Ease of Learning

Improving the robustness of a neural network can be viewed as minimizing Fep (the right hand term of the inequality in Proposition 6) during the learning scheme.

This would ensure that the neural network has learned the optimal weight distribution and is taking full advantage of the over-provisioning. Clearly, over-provisioning to guarantee ϵ'-accuracy impacts the amount of data needed for learning without over-fitting. This creates a dilemma that somehow resembles the famous bias/variance dilemma [68] in machine learning. In our case, this corresponds to a robustness/ease-of-learning dilemma. The trade-off has two forms we detail below.

Trade-off on the Lipschitz constant of the activation function (K). Choosing a low value of K leads to satisfying the inequalities of propositions 3 and 4 with high numbers of faults (dependency on K^{L-l}). But one should recall that K is an estimate of how sharp the discrimination between inputs at the level of a single neuron is (Figure 6.2). Therefore, for a network with a low-K activation function, the learning time and the number of necessary neurons can be higher than with a high-K activation function, for the latter is more discriminating.

Trade-off on synaptic weights. Like for the Lipschitz-constant K, one can note in propositions 3 and 4 (multiplications by the weight) that imposing low weights leaves some room for higher numbers of faults while still satisfying the bound. Achieving this goes through increasing the number of neurons. Intuitively, more neurons are needed to sum to the desired value, if the weights are lower.

7.4 Concluding Remarks

We established tight bounds relating the output accuracy loss of a neural network to failures of its neurons and synapses. Our bounds are derived from a quantity, Fep, the forward error propagation (given in Proposition 6), relating the propagation of imprecision in a neural network to specific parameters of that network, namely, weights, transmission capacity of synapses, coefficient of Lipschitzness of the activation function and number of neurons per layer. The bounds provide a theoretical explanation for some of the cost reduction strategies observed experimentally [88]. Leveraging these bounds, we provided a scheme to boost the synchronization of neural networks and we identified key trade-offs between robustness on the one hand, and learning cost on the other hand.

Whilst our results were established in the context of a feed forward neural network, the underlying methodology (propositions 1, 2 and 3) can be applied to other neural computing models. In the case of the convolutional network model [102], the neurons have a limited receptive field (they are not connected to all other neurons in the adjacent layers like in the feed-forward case), and the weights have a periodic distribution within each layer reducing the size of the set of synapses, which leads in turn to less restrictive bounds (i.e tolerating larger amount of failures). More precisely, the maximal weight constraint $w_m^{(l)}$ appearing in propositions 2 and 3 will run only[4] on the $R(l)$-different values of the weights from layer $l-1$ to layer l, $R(l)$ being the size of the receptive field of layer l (i.e to how many neurons of layer $l-1$ each neuron of layer l is connected).

More generally, we believe that the methodology that led to our results can be applied to any distributed system that is organised as weighted directed graph. In these systems, processes achieve a global computation while putting weights on the values of each other and (unlike classical settings in distributed computing [113]) are not performing the same computation in each node . These systems do not need to agree on values, as long as a similar condition to Assumption 1 applies (bounded channel capacity). In the last part of this thesis, we introduce one of our ongoing efforts to apply this methodology to biological (metabolic) networks.

To conclude, it is important to note that our results were established independently of the chosen learning scheme. An appealing research direction is to consider a specific learning scheme taking the forward error propagation as an additional minimization target which would reduce the impacts of failures. To our knowledge, there has been one single attempt to theoretically formulate such an optimization problem [132], but it only minimizes the effect of the crash of a *single* neuron. Our bounds help formulate the distributed optimization problem for a multiple neurons and synapses, which opens the question of the computational cost of building a neural network that achieves a given robustness constraint with such a learning scheme.

[4] With the notation of propositions 2 and 3, let us consider a convolutional network as a multilayer feed forward neural network such as for each synapse connecting a neuron i in layer l to a neuron j in layer $l-1$ not belonging to the receptive field of i, we have $w_{ji}^{(l)} = 0$. Together with the weight sharing property of convolutional networks this leads to the equality between the maximal weight in absolute value over a layer $w_m^{(l)}$ and the maximum weight in absolute value over a single receptive field.

Conclusion Part III

8 Summary and Future Work

8.1 Robust Distributed Learning

This thesis can be summarized into four technical contributions.

8.1.1 Byzantine Resilient SGD

In Chapter 2, we started by noting that the actual methods of aggregation gradients in distributed SGD are vulnerable. We formulated a condition on gradient aggregation in order to preserve convergence despite the presence of Byzantine workers.

In Chapter 3, we introduced Krum, which we proved to satisfy the aforementioned condition. We discussed Krum' computational cost, which is only linear in the dimension. We explained in the conclusion why the toolbox of traditional distributed computing is not suitable for the high dimensional situation of machine learning.

8.1.2 High Dimensional Vulnerabilities in Distributed Non-Convex Optimization

In Chapter 4, we showed that *convergence is not enough.* In high dimension $d \gg 1$, an adversary can build on the loss function's non–convexity to make SGD converge to *ineffective* models. More precisely, we brought to light that existing Byzantine–resilient schemes leave a *margin of poisoning* of $\Omega\big(f(d)\big)$, where $f(d)$ increases at least like $\sqrt{d}$. We introduced *Bulyan*, and proved it significantly reduces the attacker's leeway to a narrow $\mathcal{O}(\frac{1}{\sqrt{d}})$ bound.

8.1.3 Asynchronous Byzantine Resilient SGD

In Chapter 5, we introduced *Kardam*, the first distributed asynchronous stochastic gradient descent (SGD) algorithm that copes with Byzantine workers. Kardam consists of two complementary components: a filtering and a dampening component. The first is scalar-based and ensures resilience against $\frac{1}{3}$ Byzantine workers. Essentially, this filter leverages the Lipschitzness of cost

functions and acts as a self-stabilizer against Byzantine workers that would attempt to corrupt the progress of SGD. The dampening component bounds the convergence rate by adjusting to stale information through a generic gradient weighting scheme. We prove that Kardam guarantees almost sure convergence in the presence of asynchrony and Byzantine behavior, and we derive its convergence rate.

We empirically showed that Kardam does not introduce additional noise to the learning procedure but does induce a slowdown (the cost of Byzantine resilience) that we both theoretically and empirically showed to be less than f/n, where f is the number of Byzantine failures tolerated and n the total number of workers. We also empirically observed that the dampening component is interesting in its own right for it enables to build an SGD algorithm that outperforms alternative staleness-aware asynchronous competitors in environments with honest workers.

8.1.4 Neural Networks as a Distributed System

In Chapter 6 and Chapter 7, we view a multilayer neural network as a distributed system of which neurons can fail independently, and we evaluate its robustness in the absence of any (recovery) learning phase. We give tight bounds on the number of neurons that can fail without harming the result of a computation. To determine our bounds, we leverage the fact that neural activation functions are Lipschitz-continuous. Our bound is on a quantity, we call the *Forward Error Propagation*, capturing how much error is propagated by a neural network when a given number of components is failing, computing this quantity only requires looking at the topology of the network, while experimentally assessing the robustness of a network requires the costly experiment of looking at all the possible inputs and testing all the possible configurations of the network corresponding to different failure situations, facing a discouraging combinatorial explosion.

8.2 Back to Real Life Motivations

At first glance, the Byzantine–failure model might seem specific to distributed computing, and too pessimistic in a general ML context, for it assumes that some workers actively try to fool SGD towards the worst possible direction. In fact, we argue that this failure model is also relevant for poisoning attacks on single machines, as well as learning with unreliable data in a centralized, single–worker setting.

For instance, even when there is no *group of workers* whose estimated gradients are aggregated, an SGD update is still an attempt to *aggregate* knowledge from data and update the model subsequently. Consider a centralized, single learner, drawing data from a distribution containing a fraction of corrupt samples. The 0.5 breakdown limit for unbiased estimators, initially formulated [145] without computability considerations, establishes a formal limit to poisoning attacks: they are impossible to counter if at least half of the data is not i.i.d. from the (desired) training distribution, even with a convex cost function in small dimensions.

Going back to our introductory motivation, one should view the accounts of a social network as *workers* [1] who, by behaving on the platform, generate *gradients*, which are in turn used by a server to modify the recommendations. In a recent interview [2], Guillaume Chaslot, a former member of the Youtube recommendations team gave a striking illustration of what a Byzantine worker looks like in real life and what damage it can cause. In particular, he described how *a tiny minority* of paedophiles on Youtube could (even without aiming to) influence the recommendation of child content on the larger portion of the 3 billion users. These particular "Byzantine" users spend significantly more time on child content and hence generate gradients with larger impact. Spotting these gradients in the very high dimensional space of Youtube parameters would be hopeless, but our methods could (provably) help filtering them out of the learning, as long as the majority of Youtube users are not paedophiles.

8.3 Bridging the two Views of the Thesis

It is important to recall that our approach tackles what happens *while* training, with the goal of avoiding bad models due to poisoning attacks. We did not address the problem of evasion attacks, i.e. attacking an already trained model with adversarial inputs.

From a distributed computing point of view, there seems to be two complementary views on robustness to be made: (1) a coarse-grained robustness (quality of the gradient aggregation scheme), and (2) a fine-grained robustness (quality of the model w.r.t single parameters). For the first view, the unit of failure would be a single machine, hosting a copy of the model (or significant parts of it) and attacked in its attempt to estimate gradients. For the second view, the models used by ML, neural networks for example, can themselves be viewed as distributed computing objects[138], where the units of failure are individual neurons and weight values. The question was explored[90] in the 1990s with unexpected connections with todays' popular tools such as the dropout algorithm, which was initially derived with a (distributed computing) fault-tolerance purpose[91], then independently re-discovered as a robust regularization scheme to reduce over-fitting 20 years latter[152]. Recently, this gap-bridging between robustness from the distributed computing point of view, and robustness, from a learning performance perspective is being revived, with questions such as: how much error will be *forward-propagated* if units of the model are erroneous?

We argue that poisoning attacks should be studied both from a fine-grained, and coarse-grained perspective. For instance, the curse of dimensionality attack (Chapter 4) is an example of a coarse-grained problem (the whole system being wrong about the model) arising from a fine-grained (single-parameter) attack. An interesting question can therefore be posed: given bounds on how much an error in an individual weight value can influence the output of a model, can we compute the impact of a poisoning attack of one single component of the gradient? Subsequently, could this link be leveraged to discover the theoretical limits of *any* defense against poisoning?

[1] In fact, a worker could be any "unit" that produces gradients.
[2] https://www.youtube.com/watch?v=-bDho-i5Bqg

8.4 Revisiting our Hypotheses and Future Work

While our algorithms enable the replacement of the brittle averaging of gradient by more robust gradients, and led, among other things to the first version of the popular TensorFlow framework that is Byzantine resilient and could communicate over UDP[3]; it remains handicapped by the hypotheses we made to prove Byzantine resilience.

8.4.1 Systems for Robust Machine Learning

Over the past two years, a growing body of work, beyond our own, e.g., [4, 29, 171, 16, 45, 156, 76, 115, 174, 173, 178, 17, 177, 97, 175, 43, 30, 73, 111, 128, 75, 169, 161, 142, 127, 176, 42, 87, 141, 160, 84, 174, 67, 172, 179, 11, 165], took up the challenge of *Byzantine-resilient ML*. This thesis is part of this larger enterprise. Data poisoning [19, 170] attacks, which can have disastrous consequences [150, 124, 149, 48], represent, as we argued, a special case of this broader challenge. Beyond data poisoning, the Byzantine failure model, as originally introduced in [100], encompasses crashes, software bugs, hardware defects, message omissions, and even worse, hacked machines.

So far, all the work on Byzantine-resilient ML assumed that a fraction of workers could be Byzantine, but the server is however assumed to be always *honest* and *failure-free*. While the work produced by this line of research is useful with a trusted server, none consider the general learning scheme when the server could misbehave.

A natural way to prevent the parameter server from being a *single point–of–failure* is to *replicate* it. But this poses the problem of how to synchronize the replicas[4]. A classical technique is *state machine replication* [146] (SMR), which provides the abstraction of one parameter server[5], while benefiting from the resilience of the multiplicity of underlying replicas. But the SMR approach raises two issues. First, applying SMR to a distributed *stochastic gradient descent* (SGD) would lead to a potentially huge *overhead* as, in order to maintain the same state, replicas would need to agree on a total order of the model updates, which would induce frequent exchanges (and retransmissions) of gradients and parameter vectors, that can be several hundreds of MB large [93]. Second, agreement (on a total order) is known to be impossible in *asynchronous networks* [66] and hence, one needs to assume a bound on communication delays. But estimating this bound is challenging. A large conservative bound would imply slow reactions to message omissions and crashes, whereas a small bound could be easily exploited by an adversary that could congest

[3] TensorFlow communicates over TCP. Communicating over UDP would be faster, but causes errors that averaging does not support. Our algorithms enable the fast communication over UDP by being robust to the errors caused by UDP communication.
Georgios Damaskinos, El-Mahdi El-Mhamdi; Rachid Guerraoui; Arsany Guirguis and Sébastien Rouault (2019). AGGREGATHOR: Byzantine Machine Learning via Robust Gradient Aggregation. In The Conference on Systems and Machine Learning (SysML).

[4] A problem that we did not address in this thesis, by simply assuming the server to be reliable via some other protocol.

[5] This would allow the transparent reuse of algorithms that tolerate Byzantine workers [21, 29, 57].

parts of the network [38], jeopardizing convergence.

In one of our ongoing follow-ups [53] to this thesis, we argue that the *Byzantine SGD problem*, even when neither the workers nor the servers are trusted and when the network is asynchronous, is **strictly easier** than the general state machine problem [100] since total ordering of updates is not required in the context of ML applications: only convergence to a good final cross-accuracy is needed. We thus follow a different route where we do not require all replicas of a parameter server to maintain the same state. Instead, we allow mildly diverging parameters (which have proven beneficial in other contexts [181, 5, 96]) and present a new way to *contract* them in a *distributed* manner. In a second follow-up, we investigate the same problem in the synchronous setting, at we show that the filtering techniques presented in this thesis in the context of asynchronous networks (Kardam) can inspire fast solutions to the Byzantine servers problem[52].

However, we should acknowledge that both of the aforementioned follow-ups require, so far, constraining statistical conditions on the correct servers that we should alleviate. Specifically, either correct models (held by correct servers) are assumed to be somehow "aligned", or correct gradients (generated by correct workers) need drastically low variance values.

8.4.2 Better Theory for Better Guarantees

The weaknesses we just mentioned are a legacy of some of the hypothesis we needed to make the GARs of this thesis work. Specifically, we required the variance of the correct workers to be sufficiently low.

In Chapter 3, we experimentally showed that having a reasonably high (but not unusually high) batch size suffices to make Krum (and its more practical variant Multi-Krum) converge almost as fast as the non Byzantine resilient averaging.

In Chapter 4, we exploited our own assumptions in the high dimensional situation, which helped develop Bulyan that strengthens not only Krum, but any similar GAR. In the few days before this thesis was submitted, an interesting paper[14] was accepted to NeurIPS 2019. The authors exploit the assumptions of Krum (namely the low variance assumption) to craft attacks that are in the same spirit as the ones we presented in Chapter 4 with even less computational requirement from the adversary. Theses assumption are unfortunately data and model dependent, which makes it hard to theoretically assess if they hold a priori, when the model is an un-interpretable deep neural network.

An urgent direction we should follow is to take a step back from the non-convex world of deep learning, and formulate a better theory of Byzantine resilience, first in the simpler world of convex loss functions (as initiated by [4]). This theory should not be agnostic to the model being trained (as ourselves and most of the community did so far), but connect the assumptions that are needed for Byzantine resilience to the properties of the model. Then we can go beyond the convex case, e.g. by incorporating architecture specific properties in these assumptions and possibly leverage the recent results on neural kernels [33, 83].

8.4.3 Robust Learning Machines

Fault Tolerance as a Part of Theoretical NNs Research

Understanding the inner working of artificial neural networks (NNs) is currently one of the most pressing questions[103] in machine learning. As of now, neural networks are the backbone of the most successful machine learning solutions[148, 98]. They are deployed in safety-critical tasks in which there is little room for mistakes [62, 154]. Nevertheless, such issues are regularly reported since attention was brought to the NNs vulnerabilities over the past few years[148, 18, 125, 57].

Understanding complex systems requires understanding how they can tolerate failures of their components. This has been a particularly fruitful method in systems biology, where the mapping of the full network of metabolite molecules is a computationally quixotic venture. Instead of fully mapping the network, biologists improved their understanding of biological networks by studying the effect of deleting some of their components, one or a few perturbations at a time[40, 71]. Biological systems in general are found to be fault tolerant[131], which is thus an important criterion for biological plausibility of mathematical models.

Neuromorphic hardware

Current Machine Learning systems are bottlenecked by the underlying computational power [7]. One significant improvement over the now prevailing CPU/GPUs is *neuromorphic hardware*. In this paradigm of computation, each neuron is a physical entity [60], and the forward pass is done (theoretically) at the speed of light. However, components of such hardware are small and unreliable, leading to small random perturbations of the weights of the model [162]. Thus, robustness to weight faults is an overlooked concrete Artificial Intelligence (AI) safety problem [9]. Since we ground the assumptions of our model in the properties of NH and of biological networks, our fundamental theoretical results can be directly applied in these computing paradigms.

Research on NNs fault tolerance

In the 2000s, the fault tolerance of NNs was a major motivation for studying them [77, 92, 15]. In the 1990s, the exploration of microscopic failures was fueled by the hopes of developing neuromorphic hardware (NH) [120, 32, 139]. Taylor expansion was one of the tools used for the study of fault tolerance [74, 129]. Another line of research proposes sufficient conditions for robustness [137]. However, most of these studies are either empirical or are limited to simple architectures [162]. In addition, those studies address the worst case [18], which is known to be more severe than a random perturbation. Recently, fault tolerance was studied experimentally as well. DeepMind proposes to focus on neuron removal [126] to understand NNs. NVIDIA [104] studies error propagation caused by micro-failures in hardware [10]. In addition, mathematically similar problems are raised in the study of generalization [133, 134] and robustness [167].

Limitations of our results

In this part of the thesis, we heavily relied on the assumption that the neural networks are of *minimal size*. This assumption is useful to prove the tightness of our bounds, however, practically built neural networks are far from being minimal. In one of the ongoing follow-ups[54], we successfully got rid of this hypothesis. However, we require other hypotheses that we argue (and experimentally show) are more reasonable. Specifically, we look at the continuous limit regime, where adjacent neurons are assumed to have almost similar weights. We obtain new bounds that are closer to the empirical error, even for non-minimal networks.

8.4.4 Biological Networks

The abstraction of weighted directed graphs with non linear units is not unique to neural networks. Many other biological networks can be reasonably modelled as such. In fact, many of these networks can be also viewed as *learning machines*[6]. During this thesis, we initiated a collaboration with biologist colleagues in an effort to apply our results to models of metabolic networks. Our preliminary results[58] show the potential for our bounds to predict the most critical nodes in such a network. These nodes are the ones that are responsible for the highest forward-propagated error. Specifically, our model of error propagation could shed light on key concepts in systems biology such as *essential genes* or evolvable essential genes.

[6] What Bodies Think About: Bioelectric Computation Outside the Nervous System. Michael Levin NeurIPS 2018 (Invited Talk).

Appendix

A Krum

A.1 Multi-Krum and Krum with varying batch-size

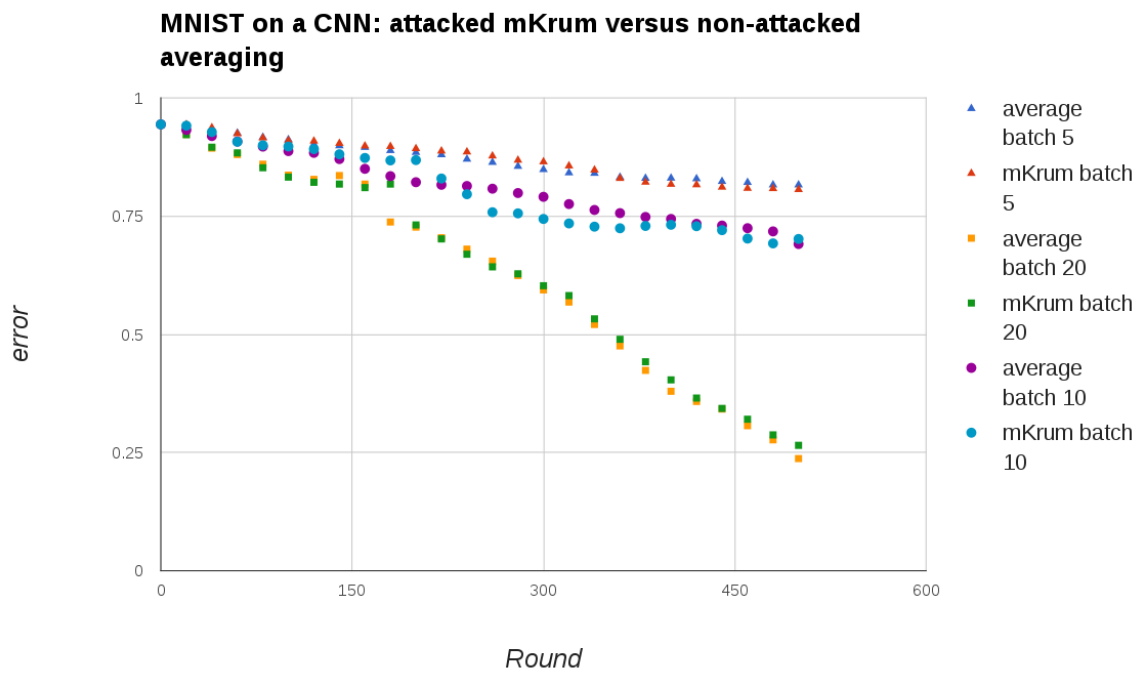


Figure A.1 – Comparing an averaging aggregation with 0% Byzantine workers to mKrum facing 45% omniscious Byzantine workers for the ConvNet on the MNIST dataset. The cross-validation error evolution during learning is plotted for 3 sizes of the size of the mini-batch.

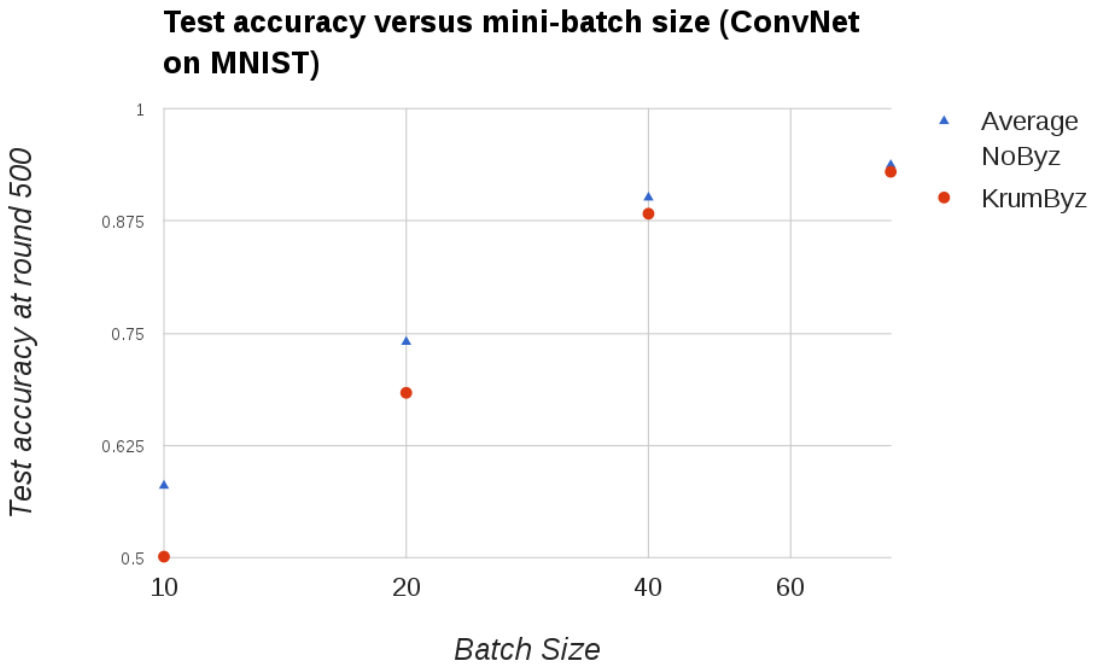


Figure A.2 – Test accuracy after 500 rounds as a function of the mini-batch size for an averaging aggregation with 0% Byzantine workers for the ConvNet on the MNIST dataset versus mKrum facing 45% of omniscious Byzantine workers.

A.2 Other Krum variants

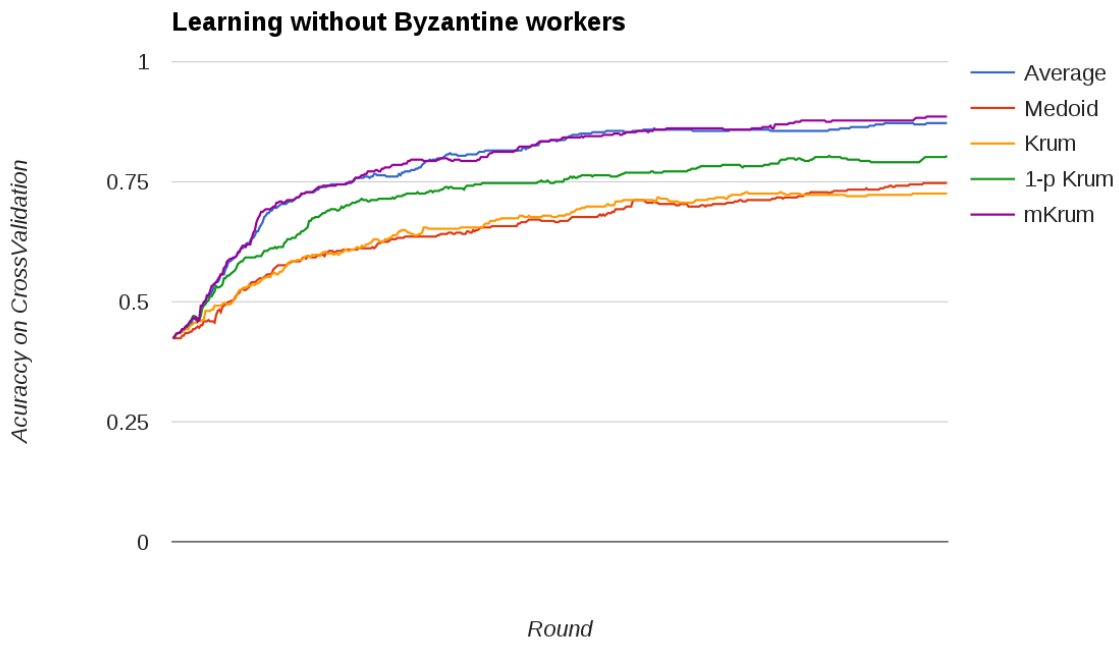


Figure A.3 – Evolution of cross-validation accuracy with rounds for the different aggregation rules in the absence of Byzantine workers. The model is the MLP and the task is spam filtering. The mini-batch size is 3. Averaging and mKrum are the fastest, 1-p Krum is second, Krum and the Medoid are the slowest.

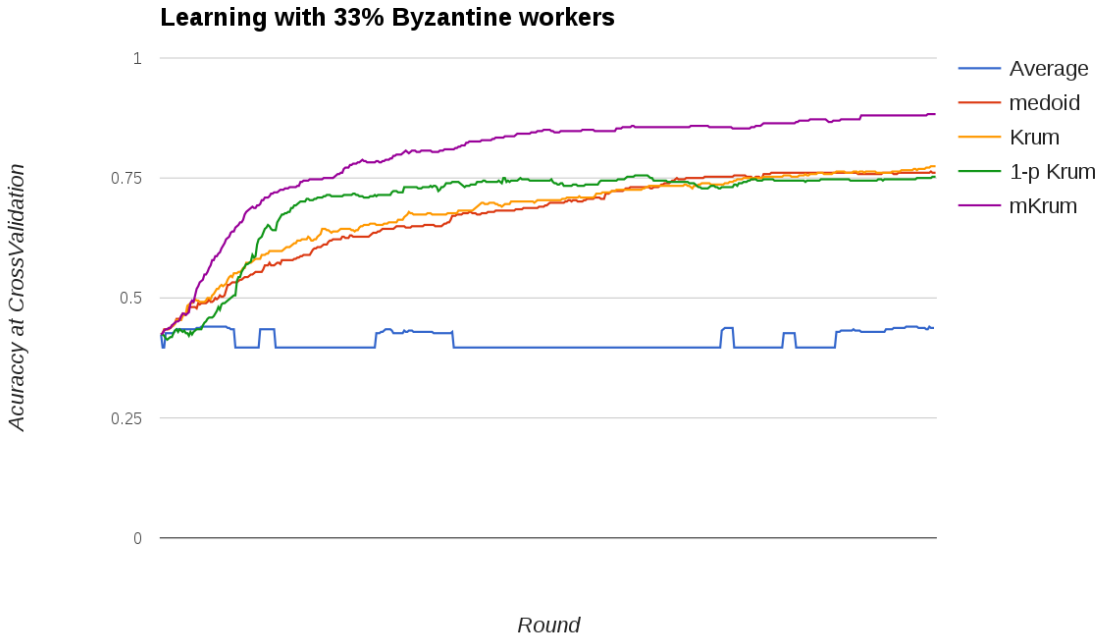


Figure A.4 – Evolution of cross-validation accuracy with rounds for the different aggregation rules in the presence of 33% Gaussian Byzantine workers. The model is the MLP and the task is spam filtering. The mini-batch size is 3. Multi-Krum (mKrum) outperforms all the tested aggregation rules.

B Bulyan

B.1 Brute's (α, f)–Byzantine–resilience proof

Background

Definition 10 ((α, f)–Byzantine–resilience)**.**
Let $(n, f) \in (\mathbb{N}^*)^2$ with $n > f$.
Let $(\alpha, f) \in \left[0, \frac{\pi}{2}\right] \times [0..n]$ be any angle and any integer.
Let $(V_1 \dots V_{n-f}) \in (\mathbb{R}^d)^{n-f}$ be independent, identically distributed random vectors, with $V_i \sim \mathscr{G}$ and $\mathbb{E}[\mathscr{G}] = G$.
Let $(B_1 \dots B_f) \in (\mathbb{R}^d)^f$ be random vectors, possibly dependent between them and the vectors $(V_1 \dots V_{n-f})$.
Then, an aggregation rule $\mathscr{F}$ is said to be (α, f)-Byzantine-resilient if, for any $1 \leq j_1 < \dots < j_f \leq n$, the vector:

$$F = \mathscr{F}\left(V_1, \dots, \underbrace{B_1}_{j_1}, \dots, \underbrace{B_f}_{j_f}, \dots, V_n\right)$$

satisfies:

1. $\langle \mathbb{E}[F], G \rangle \geq (1 - \sin\alpha) \cdot \|G\|^2 > 0$
2. $\forall r \in \{2, 3, 4\}, \mathbb{E}\|F\|^r$ *is bounded above by a linear combination of the terms* $\mathbb{E}\|\mathscr{G}\|^{r_1} \cdot \dots \cdot \mathbb{E}\|\mathscr{G}\|^{r_{n-1}}$,
   with $r_1 + \dots + r_{n-1} = r$.

DefinitionLet $(n, f) \in (\mathbb{N}^*)^2$ with $n \geq 2f + 1$.
Let $(V_1 \dots V_{n-f}) \in (\mathbb{R}^d)^{n-f}$ be independent, identically distributed random vectors, with $V_i \sim \mathscr{G}$ and $\mathbb{E}[\mathscr{G}] = G$.
Let $(B_1 \dots B_f) \in (\mathbb{R}^d)^f$ be random vectors, possibly dependent between them and the vectors

$\left(V_1 \dots V_{n-f}\right)$.
Let $\|\cdot\|_p$ be the ℓ_p–norm, with $p \in \mathbb{N}^* \cup \{+\infty\}$.

Let $\mathscr{Q} = \{V_1 \dots V_n\}$ be the set of submitted gradients.
Let $\mathscr{R} = \left\{\mathscr{X} \mid \mathscr{X} \subset \mathscr{Q}, |\mathscr{X}| = n-f\right\}$ be the set of all the subsets of $\mathscr{Q}$ with a cardinality of $n-f$.
Let $\mathscr{S} = \underset{\mathscr{X} \in \mathscr{R}}{\arg\min}\left(\max_{(V_i,V_j)\in\mathscr{X}^2}\left(\left\|V_i - V_j\right\|_p\right)\right)$.
Then, the aggregated gradient $F = \frac{1}{n-f}\sum_{V\in\mathscr{S}} V$.

ProofLet $\forall\left(i,j\right) \in \left[1..n-f\right]^2, i \neq j$ be $\bar{\sigma} \triangleq \mathbb{E}\left\|V_i - V_j\right\|_p$. Under the assumption that $2f\bar{\sigma} < \left(n-f\right)\|G\|_p$, we will prove that this rule is $\left(\alpha, f\right)$–Byzantine–resilient.

Trivial case: $\forall i \in \left[1..f\right], B_i \notin \mathscr{S}$.
As the aggregated gradient F is the arithmetic mean of unbiased vectors V_j, we have $\mathbb{E}[F] = G$, and points *1.* and *2.* of definition 10 are trivially satisfied.

Otherwise, without loss of generality, let $b \in \left[1..f\right]$ and $\mathscr{S} = \left\{V_1 \dots V_{n-f-b}, B_1 \dots B_b\right\}$, $\bar{\mathscr{R}} = \mathscr{R} \setminus \mathscr{S}$.
It holds:

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

We can also notice that: $\exists \mathscr{V} \in \bar{\mathscr{R}}, \forall i \in \left[1..f\right], B_i \notin \mathscr{V}$. Then, by combining this observation with the previous one:

$$\begin{aligned}
&\forall a \in [1..b], B_a \in \mathscr{S}\\
\Rightarrow\ &\exists\left(x,y\right) \in \left[1..n-f\right]^2, i \neq j,\\
&\forall k \in \left[1..n-f-b\right], \|B_a - V_k\|_p < \left\|V_x - V_y\right\|_p
\end{aligned}$$

This last observation will be reused in the following.

We can compute the aggregated gradient:

$$F = \frac{1}{n-f}\left(\sum_{i=1}^{n-f-b} V_i + \sum_{i=1}^{b} B_i\right)$$

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

$$
\begin{aligned}
\widehat{G} &= \frac{1}{n-f}\sum_{i=1}^{n-f} V_i \\
F - \widehat{G} &= \frac{1}{n-f}\left(\sum_{i=1}^{b} B_i - \sum_{i=n-f-b+1}^{n-f} V_i\right) \\
&= \frac{1}{n-f}\sum_{i=1}^{b} B_i - V_{i+n-f-b} \\
\left\|F - \widehat{G}\right\|_p &\leq \frac{1}{n-f}\sum_{i=1}^{b} \left\|B_i - V_{i+n-f-b}\right\|_p \\
&\leq \frac{1}{n-f}\sum_{i=1}^{b} \Big(\|B_i - V_k\|_p \\
&\qquad + \left\|V_k - V_{i+n-f-b}\right\|_p\Big) \\
&\leq \frac{1}{n-f}\sum_{i=1}^{b} \Big(\left\|V_x - V_y\right\|_p \\
&\qquad + \left\|V_k - V_{i+n-f-b}\right\|_p\Big)
\end{aligned}
$$

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

$$
\begin{aligned}
\|\mathbb{E}[F] - G\|_p &\leq \mathbb{E}\left\|F - \widehat{G}\right\|_p \\
&\leq \frac{1}{n-f}\sum_{i=1}^{b} \bar{\sigma} + \bar{\sigma} \\
&\leq \frac{2\,b\,\bar{\sigma}}{n-f} \leq \frac{2\,f\,\bar{\sigma}}{n-f}
\end{aligned}
$$

So, under the assumption that $2\,f\,\bar{\sigma} < \big(n-f\big)\,\|G\|_p$, we verify that $\|\mathbb{E}[F] - G\|_p < \|G\|_p$, and so: $\langle \mathbb{E}[F]\,, G\rangle > 0$.

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

$$
\mathbb{E}\,\|F\|_p^r \leq \frac{n-f-b}{n-f}\mathbb{E}\,\|\mathscr{G}\|_p^r + \frac{1}{n-f}\sum_{i=1}^{b}\mathbb{E}\,\|B_i\|_p^r
$$

Then, by using the binomial theorem twice:

$$\begin{aligned}
\|B_i\|_p^r &\leq \sum_{r_1+r_2=r} \binom{r}{r_1} \|B_i - V_k\|_p^{r_1} \|V_k\|_p^{r_2} \\
&\text{with } k \in [1..n-f-d] \\
\|B_i - V_k\|_p^{r_1} &\leq \left\|V_x - V_y\right\|_p^{r_1} \\
&\leq \sum_{r_3+r_4=r_1} \binom{r_1}{r_3} \|V_x\|_p^{r_3} \left\|V_y\right\|_p^{r_4}
\end{aligned}$$

Finally, as $(V_1 \dots V_{n-f})$ are *independent, identically distributed* random variables following the same distribution $\mathscr{G}$, we have that $\forall (i,j) \in [1..n-f]^2$, $i \neq j$, $\mathbb{E}\left[\|V_i\|_p^{r_1} \left\|V_j\right\|_p^{r_2}\right] = \mathbb{E}\|\mathscr{G}\|_p^{r_1} \cdot \mathbb{E}\|\mathscr{G}\|_p^{r_2}$, and so $\mathbb{E}\|B_i\|_p^r$ is bounded as described in point *2.* of definition 10.

B.2 Approximation of α_m, with $p \in \mathbb{N}^*$

Prior conventions and assumptions Let remind that $\forall i \in [1..n-f]$, $V_i = \left(v_1^{(i)} \dots v_d^{(i)}\right) \sim \mathscr{G}$.
We model each coordinate as a *normal distribution*:

$$\begin{aligned}
&\forall j \in [1..d], \exists (\mu_j, \sigma_j) \in \mathbb{R}^2, \\
&\forall i \in [1..n-f], v_j^{(i)} \sim \mathscr{N}(\mu_j, {\sigma_j}^2)
\end{aligned}$$

We assume $d \gg 1$, and we will write $\bar{\delta}$ for:

$$\begin{aligned}
\forall (i,j) \in [1..n-f]^2, i \neq j, \bar{\delta} &= \frac{1}{d} \sum_{k=1}^{d} \mathbb{E}\left|v_k^{(i)} - v_k^{(j)}\right| \\
&= \frac{2}{d\sqrt{\pi}} \sum_{k=1}^{d} \sigma_k \\
\text{and note that: } \frac{1}{d} \sum_{k=1}^{d} \mathbb{E}\left|v_k^{(i)} - \mu_k\right| &= \frac{\sqrt{2}}{d\sqrt{\pi}} \sum_{k=1}^{d} \sigma_k \\
&= \frac{\bar{\delta}}{\sqrt{2}}
\end{aligned}$$

Then, $\forall (i,j) \in [1..n-f]^2$, $i \neq j$, we can approximate:

$$\begin{aligned}
\left\|V_i - V_j\right\|_p &= \left(\sum_{k=1}^{d} \left|v_k^{(i)} - v_k^{(j)}\right|^p\right)^{\frac{1}{p}} \\
&\approx \left(d\bar{\delta}^p\right)^{\frac{1}{p}}
\end{aligned}$$

Let $E = (0 \dots 0, 1, 0 \dots 0) \in \mathbb{R}^d$ the attacked coordinate. Then, with $\alpha_m > 0$, $B = \overline{V} + \alpha_m E$, we can

approximate:

$$\begin{aligned}
\|B - V_i\|_p &= \left(\left(\sum_{k=1}^{d} \left|v_k^{(i)} - \bar{v}_k\right|^p\right)\right. \\
&\quad \left. - \left|v_e^{(i)} - \bar{v}_e\right|^p + \left|v_e^{(i)} - \bar{v}_e + \alpha_m\right|^p\right)^{\frac{1}{p}} \\
&\approx \left({\alpha_m}^p + \sum_{k=1}^{d} \left|v_k^{(i)} - \mu_k\right|^p\right)^{\frac{1}{p}} \\
&\approx \left({\alpha_m}^p + d\left(\frac{\bar{\delta}}{\sqrt{2}}\right)^p\right)^{\frac{1}{p}}
\end{aligned}$$

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

Assuming B is selected by Brute:

$$\begin{aligned}
& B \in \mathscr{S} \\
\Rightarrow\ & \exists (x, y) \in [1 .. n - f]^2,\ i \neq j, \\
& \forall k \in [1 .. n - f - b],\ \|B - V_k\|_p < \left\|V_x - V_y\right\|_p \\
\rightsquigarrow\ & \left({\alpha_m}^p + d\left(\frac{\bar{\delta}}{\sqrt{2}}\right)^p\right)^{\frac{1}{p}} < \left(d\,\bar{\delta}^p\right)^{\frac{1}{p}} \\
\rightsquigarrow\ & \alpha_m < \left(\left(1 - \frac{1}{\sqrt{2}^p}\right) d\right)^{\frac{1}{p}} \bar{\delta}
\end{aligned}$$

This is a *necessary*, approximated condition. It is only to give broad insights on the relation between some hyper–parameters and α_m: with p, q constants, $\alpha_m = \mathscr{O}\left(\bar{\delta}\ \sqrt[p]{d}\right)$.

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.

First, we approximate the Byzantine submission's score:

$$\begin{aligned}
s(B) &\approx 2\,\|B - V_i\|_p^q \\
&\approx 2\left({\alpha_m}^p + d\left(\frac{\bar{\delta}}{\sqrt{2}}\right)^p\right)^{\frac{q}{p}}
\end{aligned}$$

$\forall i \in [1 .. n - f]$, let $b \in [0 .. f]$ be how many B belongs to the $n - f - 2$ closest vectors to V_i. Then

the score of V_i is:

$$\begin{aligned} s(V_i) &\approx b\,\|B - V_i\|_p^q + \left(f+1-b\right)\left\|V_j - V_i\right\|_p^q \\ &\approx b\left(\alpha_m{}^p + d\left(\frac{\bar{\delta}}{\sqrt{2}}\right)^p\right)^{\frac{q}{p}} + \left(f+1-b\right)\left(d\,\bar{\delta}^p\right)^{\frac{q}{p}} \end{aligned}$$

Finally, B is selected $\Rightarrow \forall i \in \left[1..n-f\right], s(B) \lessapprox s(V_i)$

$$\begin{aligned} &\underset{\forall i}{\Rightarrow} \quad (2-b)\left(\alpha_m{}^p + d\left(\frac{\bar{\delta}}{\sqrt{2}}\right)^p\right)^{\frac{q}{p}} \lessapprox \left(f+1-b\right)\left(d\,\bar{\delta}^p\right)^{\frac{q}{p}} \\ &\underset{\exists i}{\Rightarrow} \quad \alpha_m \lessapprox \left(\left(\frac{f+1-b}{2-b}\right)^{\frac{p}{q}} - \frac{1}{\sqrt{2}^p}\right)^{\frac{1}{p}} d^{\frac{1}{p}}\,\bar{\delta} \end{aligned}$$

This last implication is always true: there *must* be at least one non–Byzantine vector V_j for with $b \in \{0,1\}$; else α_m could increase unbounded, which would be absurd.

In conclusion, with p, q constants: $\alpha_m = \mathcal{O}\left(\bar{\delta}\,\sqrt[q]{f}\,\sqrt[p]{d}\right)$.

B.3 Supplementary experiments

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

Krum/GeoMed	30 non–Byzantines + 27 Byzantines
Brute	6 non–Byzantines + 5 Byzantines
Average	30 non–Byzantines + 0 Byzantines

On CIFAR–10, we use $\eta_0 = 0.5$, $r_\eta = 2000$, a batch size of 128 images (256 for Brute), and for the worker counts:

Krum/GeoMed	21 non–Byzantines + 18 Byzantines
Brute	6 non–Byzantines + 5 Byzantines
Average	21 non–Byzantines + 0 Byzantines

In Figure B.1, the attack is maintained only up to 50 epochs. The attack variant for ℓ_∞ norm–based gradient aggregation rules exhibited a very strong impact. None of the presented gradient aggregation rules prevented the stochastic gradient descent from being *pushed* and remaining in a sub–space of *ineffective* models, and for at least 1000 epochs.

In Figure B.2, the attack is never stopped. Again, none of the presented gradient aggregation rules prevented the stochastic gradient descent from being *pushed* and remaining in a sub–space of

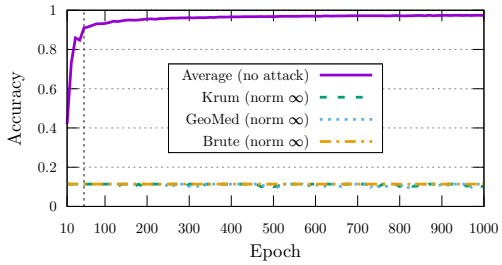


Figure B.1 – MNIST: accuracy on the testing set up to epoch 1000**, comparing the presented aggregation rules under our attack. The attack was maintained only up to epoch** 50 **(dotted line). The *average* is the reference: it is the accuracy the model would have shown if only non–Byzantine gradients had been selected.**

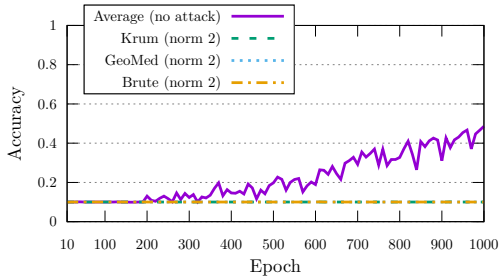


Figure B.2 – CIFAR–10: accuracy on the testing set up to epoch 1000**, comparing the presented aggregation rules under our attack. The *average* is the reference: it is the accuracy the model would have shown if only non–Byzantine gradients had been selected.**

ineffective models, for at least 1000 epochs.

C Kardam

C.1 Experiments

In this section, we report on our empirical evaluation of our distributed implementation of Kardam. Experiments on Byzantine attacks are mostly illustrative for (1) the importance of the dampening component and (2) the overhead of the filtering component. Due to the intractability of testing all possible attacks, the only option is to prove Byzantine resilience mathematically and focus in the empirical part on the performance overhead of Kardam.

We employ the convolutional neural network (CNN) described in Table C.1 for image classification on CIFAR-100. The chosen base learning rate is $15 * 10^{-4}$ and the mini-batch size is 100 examples [135]. If not stated otherwise, we employ a setup with no actual Byzantine behavior and deploy Kardam with $f = 3$ on 10 workers.

Parameters	Input	Conv1	Pool1	Conv2	Pool2	FC1	FC2	FC3
Kernel size	32×32×3	3×3×16	3×3	3×3×64	4×4	384	192	100
Strides		1×1	3×3	1×1	4×4			

Table C.1 – CNN parameters for CIFAR-100.

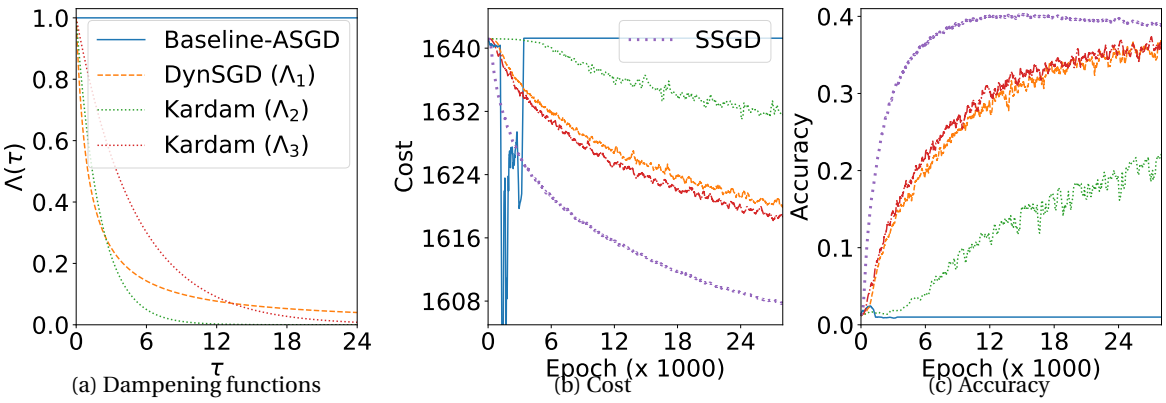
Staleness-aware learning. We simulate a Gaussian staleness distribution [183] and evaluate Kardam with different dampening functions $\Lambda(\tau)$ (Definition 5) shown in Figure C.1(a). We compare with the performance of Kardam without the Byzantine resilience component (BASELINE-ASGD) by using the constant function ($\Lambda_1 = 1$). Additionally, we compare with a state-of-the-art staleness-aware learning algorithm (DYNSGD [85]) that employs an inverse dampening function ($\Lambda_2(\tau) = \frac{1}{1+\tau}$). Finally, we use two exponential functions ($\Lambda = exp(-\alpha * \tau)$) which, to the best of our knowledge, only Kardam enables.

Figure C.1 depicts the very fact that the staleness-aware component of Kardam is crucial in asynchronous environments. We simulate a Gaussian distribution (Figure C.1(b)) (similar to [183]) and show that SSGD has the faster convergence whereas BASELINE-ASGD diverges (Figures C.1(b) and C.1(c)).

Figure C.1 also highlights the need for an adjustable smoothness on the dampening function. A

very steep function (Λ_2) almost ignores many of the updates (weighted by a very small value) and thus suffers a slower convergence. A tuned exponential function (Λ_3) accelerates the convergence in comparison with the inverse function of DYNSGD. Moreover, Kardam (Λ_3) assigns larger weights to the less stale updates ($\tau < 13$) compared to DYNSGD and vice versa.

The dampening function selection is the outcome of adjusting the trade-off between the robustness and the magnitude of each update. We observe similar results for the EMNIST dataset (in our supplementary material) and thus highlight that the dampening function can be selected based on the expected staleness distribution, and not necessarily adjusted for each different application.



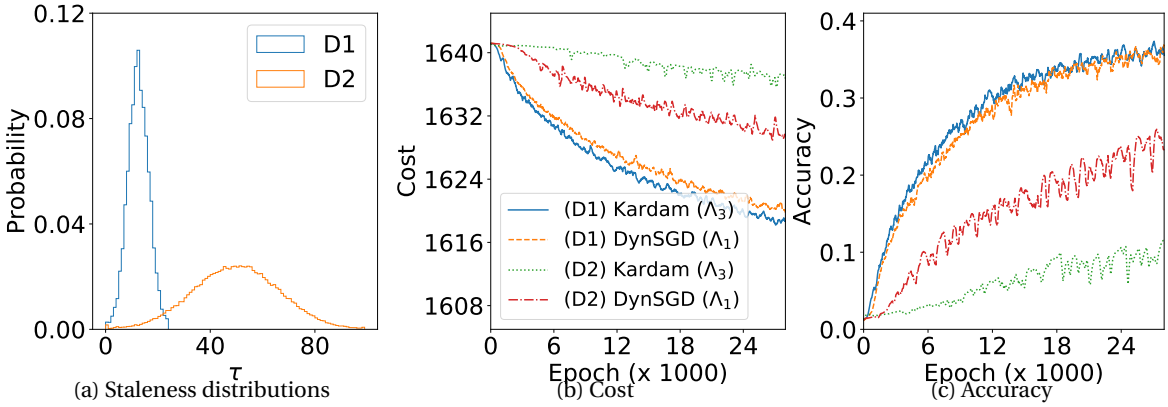
(a) Dampening functions (b) Cost (c) Accuracy

Figure C.1 – Staleness-aware learning for CIFAR-100. BASELINE-ASGD denotes Kardam without the dampening component and SSGD the ideal (synchronous) SGD execution. The staleness follows a Gaussian distribution ($mean = 12$, $\sigma = 4$) and the dampening functions are $\Lambda_1 = \frac{1}{\tau+1}$, $\Lambda_2 = e^{0.5\tau}$, $\Lambda_3 = e^{0.2\tau}$.

Impact of staleness. An increase in the amount of staleness leads to a slower convergence according to Theorem 5 (i.e., larger χ in Equation 5.4). Figure C.2 depicts the impact of the amount of staleness on Kardam and DYNSGD for two different staleness distributions (D_1 and D_2). We observe that the smaller the mean of the distribution, the faster the convergence. We verify that Kardam outperforms DYNSGD for D_1 and vice versa for D_2, as the values of the exponential dampening function become very small for the larger staleness values (D_2). We highlight that our experimental setup includes significantly higher staleness (D2) than the competitors [183, 85].

Byzantine resilience. We observe that the overhead of the Byzantine resilience in the setup with no actual Byzantine behavior is only in terms of filtered (i.e., wasted) gradients and not in terms of convergence speed (in terms of epochs). Moreover, the drop ratio under the staleness distribution D_1 is 27.9% and 19.6% for Kardam employing Λ_1 and Λ_3 respectively, thus aligned with our theoretical bound (Theorem 1). The slowdown would decrease accordingly by decreasing f, i.e., being more optimistic about the number of Byzantine workers.

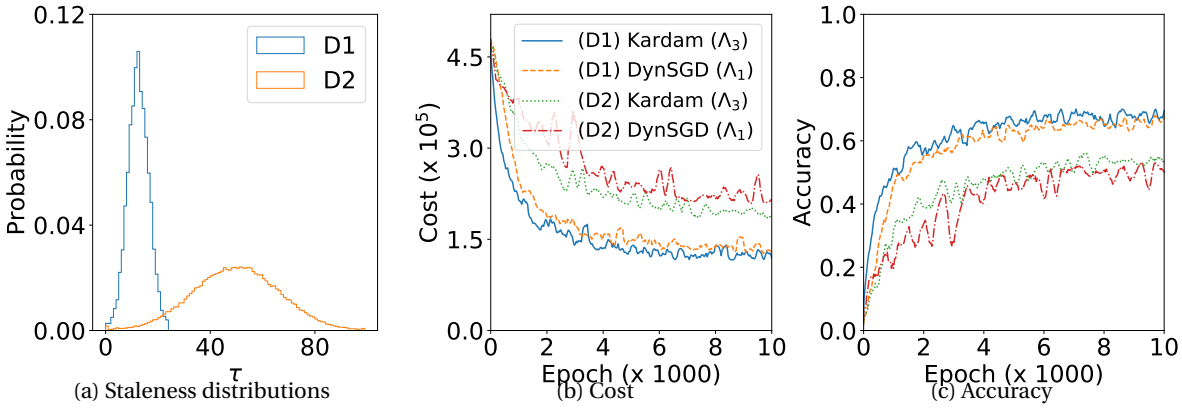
We test Kardam against a *baseline* Byzantine behavior (3 out of 10 workers send $g_p^{byz} = -10g_p$) and observe that Kardam successfully filters 100% of the Byzantine gradients (an empirical confirmation of the theoretically proven Byzantine resilience of Kardam).



(a) Staleness distributions (b) Cost (c) Accuracy

Figure C.2 – Impact of staleness for CIFAR-100.

Experimental Results. We also evaluate the performance of Kardam for image classification on the EMNIST dataset[1] consisting 814,255 examples of handwritten characters and digits (62 classes). We perform min-max scaling normalization as a pre-processing step resulting in 784 normalized input. We split the dataset into 697,932 training and 116,323 test examples and employ a base learning rate of $8 * 10^{-4}$ alongside a mini-batch of 100 examples if not stated otherwise.



(a) Staleness distributions (b) Cost (c) Accuracy

Figure C.3 – Impact of staleness for EMNIST.

[1]https://www.nist.gov/itl/iad/image-group/emnist-dataset

Bibliography

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI). Savannah, Georgia, USA*, 2016.

[2] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318. ACM, 2016.

[3] G. Alain, A. Lamb, C. Sankar, A. Courville, and Y. Bengio. Variance reduction in sgd by distributed importance sampling. *arXiv preprint arXiv:1511.06481*, 2015.

[4] D. Alistarh, Z. Allen-Zhu, and J. Li. Byzantine stochastic gradient descent. In *Neural Information Processing Systems, to appear*, 2018.

[5] D. Alistarh, J. Li, R. Tomioka, and M. Vojnovic. Qsgd: Randomized quantization for communication-optimal stochastic gradient descent. *arXiv preprint arXiv:1610.02132*, 2016.

[6] C. Allen and C. F. Stevens. An evaluation of causes for unreliability of synaptic transmission. *Proceedings of the National Academy of Sciences*, 91(22):10380–10383, 1994.

[7] D. Amodei and D. Hernandez. AI and compute. *Downloaded from https://blog.openai.com/ai-and-compute*, 2018.

[8] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

[9] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

[10] A. P. Arechiga and A. J. Michaels. The robustness of modern deep learning architectures against single event upset errors. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pages 1–6. IEEE, 2018.

[11] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov. How to backdoor federated learning. *arXiv preprint arXiv:1807.00459*, 2018.

[12] A. R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945, 1993.

[13] P. L. Bartlett, D. J. Foster, and M. J. Telgarsky. Spectrally-normalized margin bounds for neural networks. In *Neural Information Processing Systems*, pages 6241–6250, 2017.

[14] M. Baruch, G. Baruch, and Y. Goldberg. A little is enough: Circumventing defenses for distributed learning. *NeurIPS (to appear)*, 2019.

[15] Y. Bengio and R. De Mori. Use of neural networks for the recognition of place of articulation. In *Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference on*, pages 103–106. IEEE, 1988.

[16] J. Bernstein, J. Zhao, K. Azizzadenesheli, and A. Anandkumar. signSGD with majority vote is communication efficient and fault tolerant. In *International Conference on Learning Representations*, 2019.

[17] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo. Analyzing federated learning through an adversarial lens. *arXiv preprint arXiv:1811.12470*, 2018.

[18] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 387–402. Springer, 2013.

[19] B. Biggio and P. Laskov. Poisoning attacks against support vector machines. In *In International Conference on Machine Learning (ICML*. Citeseer, 2012.

[20] B. Biggio and F. Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *arXiv preprint arXiv:1712.03141*, 2017.

[21] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Advances in Neural Information Processing Systems (**NeurIPS**)*. 2017.

[22] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[23] L. Bottou. Online learning and stochastic approximations. *Online learning in neural networks*, 17(9):142, 1998.

[24] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[25] L. Bottou. *Stochastic Gradient Descent Tricks*, pages 421–436. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[26] O. Bousquet and L. Bottou. The tradeoffs of large scale learning. In *Neural Information Processing Systems*, pages 161–168, 2008.

[27] C. Cachin, R. Guerraoui, and L. Rodrigues. *Introduction to reliable and secure distributed programming.* 2011.

[28] J. Chen, R. Monga, S. Bengio, and R. Jozefowicz. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, 2016.

[29] L. Chen, H. Wang, and D. Papailiopoulos. Draco: Robust distributed training against adversaries, 2018.

[30] X. Chen, T. Chen, H. Sun, Z. S. Wu, and M. Hong. Distributed training with heterogeneous data: Bridging median and mean based algorithms. *arXiv preprint arXiv:1906.01736*, 2019.

[31] Y. Chen, L. Su, and J. Xu. Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. *arXiv preprint arXiv:1705.05491*, 2017.

[32] C.-T. Chiu et al. Robustness of feedforward neural networks. In *IEEE International Conference on Neural Networks*, pages 783–788. IEEE, 1993.

[33] L. Chizat, E. Oyallon, and F. Bach. On lazy training in differentiable programming. 2019.

[34] A. Choromanska, M. Henaff, M. Mathieu, G. Ben Arous, and Y. LeCun. The loss surfaces of multilayer networks. In *Artificial Intelligence and Statistics*, pages 192–204, 2015.

[35] A. Choromanska, Y. LeCun, and G. Ben Arous. Open problem: The landscape of the loss surfaces of multilayer networks. In *Conference on Learning Theory*, pages 1756–1760, 2015.

[36] G. Chowdhary and E. Johnson. Adaptive neural network flight control using both current and recorded data. In *AIAA Guidance, Navigation, and Control Conference, AIAA-2007-6505. Hilton Head*, 2007.

[37] M. Cisse, P. Bojanowski, E. Grave, Y. Dauphin, and N. Usunier. Parseval networks: Improving robustness to adversarial examples. In *International Conference on Machine Learning*, pages 854–863, 2017.

[38] A. Clement, E. L. Wong, L. Alvisi, M. Dahlin, and M. Marchetti. Making byzantine fault tolerant systems tolerate byzantine faults. In *NSDI*, volume 9, pages 153–168, 2009.

[39] M. B. Cohen, Y. T. Lee, G. Miller, J. Pachocki, and A. Sidford. Geometric median in nearly linear time. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 9–21. ACM, 2016.

[40] M. Costanzo, B. VanderSluis, E. N. Koch, A. Baryshnikova, C. Pons, G. Tan, W. Wang, M. Usaj, J. Hanchard, S. D. Lee, et al. A global genetic interaction network maps a wiring diagram of cellular function. *Science*, 353(6306):aaf1420, 2016.

[41] H. Cui, J. Cipar, Q. Ho, J. K. Kim, S. Lee, A. Kumar, J. Wei, W. Dai, G. R. Ganger, P. B. Gibbons, et al. Exploiting bounded staleness to speed up big data analytics. In *USENIX ATC*, pages 37–48, 2014.

[42] A. R. da Silva, L. M. Rodrigues, L. de Oliveira Rech, and A. F. Luiz. Rdfm: Resilient distributed factorization machines. In *International Conference on Artificial Intelligence and Soft Computing*, pages 585–594. Springer, 2019.

[43] D. Data, L. Song, and S. Diggavi. Data encoding for byzantine-resilient distributed gradient descent. In *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 863–870. IEEE, 2018.

[44] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[45] I. Diakonikolas, G. Kamath, D. M. Kane, J. Li, A. Moitra, and A. Stewart. Robustly learning a gaussian: Getting optimal error, efficiently. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2683–2702. Society for Industrial and Applied Mathematics, 2018.

[46] A. Diaz Alvarez et al. Modeling the driving behavior of electric vehicles using smartphones and neural networks. *Intelligent Transportation Systems Magazine, IEEE*, 6(3):44–53, 2014.

[47] D. L. Donoho and P. J. Huber. The notion of breakdown point. *A festschrift for Erich L. Lehmann*, 157184, 1983.

[48] M. Dredze, D. A. Broniatowski, M. C. Smith, and K. M. Hilyard. Understanding vaccine refusal: why we need social media now. *American journal of preventive medicine*, 50(4):550–552, 2016.

[49] E. El Mhamdi and R. Guerraoui. When neurons fail - technical report. page 19, 2016. Biological Distributed Algorithms Workshop, Chicago.

[50] E. M. El Mhamdi and R. Guerraoui. When neurons fail. Technical report, EPFL, 2016.

[51] E. M. El Mhamdi and R. Guerraoui. When neurons fail. In *IEEE International Parallel and Distributed Processing Symposium (**IPDPS**)*. 2017.

[52] E. M. El Mhamdi, R. Guerraoui, and A. Guirguis. Fast Byzantine machine learning with unreliable servers. *arXiv preprint arXiv:1911.07537*, 2019.

[53] E. M. El Mhamdi, R. Guerraoui, A. Guirguis, and S. Rouault. Sgd: Decentralized Byzantine resilience. *arXiv preprint arXiv:1905.03853*, 2019.

[54] E. M. El Mhamdi, R. Guerraoui, A. Kucharavy, and S. Volodin. The probabilistic fault tolerance of neural networks in the continuous limit. *arXiv preprint arXiv:1902.01686*, 2019.

[55] E. M. El Mhamdi, R. Guerraoui, and S. Rouault. On the robustness of a neural network. In *IEEE Symposium on Reliable Distributed Systems (**SRDS**)*. 2017.

[56] E. M. El Mhamdi, R. Guerraoui, and S. Rouault. The hidden vulnerability of distributed learning in byzantium. *arXiv preprint arXiv:1802.07927*, 2018.

[57] E. M. El Mhamdi, R. Guerraoui, and S. Rouault. The hidden vulnerability of distributed learning in Byzantium. In *International Conference on Machine Learning (**ICML**)*. 2018.

[58] E. M. El Mhamdi, A. Kucharavy, R. Guerraoui, and R. Li. Predicting complex genetic phenotypes using error propagation in weighted networks. *bioRxiv, 487348 (under review for a biology journal)*, 2018.

[59] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha. Backpropagation for energy-efficient neuromorphic computing. In *Advances in Neural Information Processing Systems*, pages 1117–1125, 2015.

[60] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha. Backpropagation for energy-efficient neuromorphic computing. In *Advances in Neural Information Processing Systems*, pages 1117–1125, 2015.

[61] S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch, et al. Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the National Academy of Sciences*, page 201604850, 2016.

[62] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115, 2017.

[63] M. E. Farmer, C. S. Jacobs, and S. Cong. Neural network radar processor, Apr. 2 2002. US Patent 6,366,236.

[64] A. Fawzi, S.-M. Moosavi-Dezfooli, and P. Frossard. Robustness of classifiers: from adversarial to random noise. In *Advances in Neural Information Processing Systems*, pages 1624–1632, 2016.

[65] J. Feng, H. Xu, and S. Mannor. Outlier robust online learning. *arXiv preprint arXiv:1701.00251*, 2017.

[66] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[67] C. Fung, C. J. Yoon, and I. Beschastnikh. Mitigating sybils in federated learning poisoning. *arXiv preprint arXiv:1808.04866*, 2018.

[68] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

[69] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 69–77. ACM, 2011.

[70] J. Gilmer, L. Metz, F. Faghri, S. S. Schoenholz, M. Raghu, M. Wattenberg, and I. Goodfellow. Adversarial spheres. *arXiv preprint arXiv:1801.02774*, 2018.

[71] J. I. Glass, N. Assad-Garcia, N. Alperovich, S. Yooseph, M. R. Lewis, M. Maruf, C. A. Hutchison, H. O. Smith, and J. C. Venter. Essential genes of a minimal bacterium. *Proceedings of the National Academy of Sciences*, 103(2):425–430, 2006.

[72] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[73] N. Gupta and N. H. Vaidya. Byzantine fault tolerant distributed linear regression. *arXiv preprint arXiv:1903.08752*, 2019.

[74] N. C. Hammadi and H. Ito. A learning algorithm for fault tolerant feedforward neural networks. *IEICE TRANSACTIONS on Information and Systems*, 80(1):21–27, 1997.

[75] C. Hardy. *Contribution au développement de l'apprentissage profond dans les systèmes distribués*. PhD thesis, Rennes 1, 2019.

[76] C. Hardy, E. Le Merrer, and B. Sericola. Md-gan: Multi-discriminator generative adversarial networks for distributed datasets. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 866–877. IEEE, 2019.

[77] S. S. Haykin. *Neural networks and learning machines*, volume 3. Pearson Upper Saddle River, NJ, USA:, 2009.

[78] M. Herlihy, S. Rajsbaum, M. Raynal, and J. Stainer. Computing in the presence of concurrent solo executions. In *Latin American Symposium on Theoretical Informatics*, pages 214–225. Springer, 2014.

[79] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing. More effective distributed ml via a stale synchronous parallel parameter server. In *Advances in neural information processing systems*, pages 1223–1231, 2013.

[80] A. Holzinger. Interactive machine learning for health informatics: when do we need the human-in-the-loop? *Brain Informatics*, 3(2):119–131, 2016.

[81] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[82] G. Indiveri et al. Neuromorphic silicon neuron circuits. *Frontiers in neuroscience*, Vol. 5, 2011.

[83] A. Jacot, F. Gabriel, and C. Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 8580–8589. Curran Associates, Inc., 2018.

[84] J. Ji, X. Chen, Q. Wang, L. Yu, and P. Li. Learning to learn gradient aggregation by gradient descent.

[85] J. Jiang, B. Cui, C. Zhang, and L. Yu. Heterogeneity-aware distributed parameter servers. In *SIGMOD*, pages 463–478, 2017.

[86] R. Jiang, S. Chiappa, T. Lattimore, A. Agyorgy, and P. Kohli. Degenerate feedback loops in recommender systems. *arXiv preprint arXiv:1902.10730*, 2019.

[87] R. Jin, X. He, and H. Dai. Distributed byzantine tolerant stochastic gradient descent in the era of big data. *arXiv preprint arXiv:1902.10336*, 2019.

[88] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, N. E. Jerger, and A. Moshovos. Proteus: Exploiting numerical precision variability in deep neural networks. In *Proceedings of the 2016 International Conference on Supercomputing*, page 23. ACM, 2016.

[89] Z. Kaoudi, J.-A. Quiané-Ruiz, S. Thirumuruganathan, S. Chawla, and D. Agrawal. A cost-based optimizer for gradient descent optimization. In *SIGMOD*, pages 977–992, 2017.

[90] P. Kerlirzin. *Etude de la robustesse des réseaux multicouches*. PhD thesis, Paris 11, 1994.

[91] P. Kerlirzin and F. Vallet. Robustness in multilayer perceptrons. *Neural computation*, 5(3):473–482, 1993.

[92] P. Kerlirzin and F. Vallet. Robustness in multilayer perceptrons. *Neural computation*, 5(3):473–482, 1993.

[93] L. Kim. How many ads does google serve in a day? http://goo.gl/oIidXO, November 2012.

[94] P. W. Koh and P. Liang. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning*, pages 1689–1698, 2017.

[95] J. Konečnỳ, B. McMahan, and D. Ramage. Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575*, 2015.

[96] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[97] N. Konstantinov and C. Lampert. Robust learning from untrusted sources. *arXiv preprint arXiv:1901.10310*, 2019.

[98] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[99] A. Kumar, S. Mehta, and D. Vijaykeerthy. An introduction to adversarial machine learning. In *International Conference on Big Data Analytics*, pages 293–299. Springer, 2017.

[100] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[101] S. Lawrence, C. L. Giles, and A. C. Tsoi. What size neural network gives optimal generalization? Convergence properties of backpropagation. Pennsylvania State University, 1998.

[102] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10), 1995.

[103] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[104] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler. Understanding error propagation in deep learning neural network (dnn) accelerators and applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 8. ACM, 2017.

[105] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su. Scaling distributed machine learning with the parameter server. In *OSDI*, volume 1, page 3, 2014.

[106] M. Li, D. G. Andersen, A. J. Smola, and K. Yu. Communication efficient distributed machine learning with the parameter server. In *Advances in Neural Information Processing Systems*, pages 19–27, 2014.

[107] M. Li, L. Zhou, Z. Yang, A. Li, F. Xia, D. G. Andersen, and A. Smola. Parameter server for distributed machine learning. In *Big Learning NIPS Workshop*, volume 6, page 2, 2013.

[108] X. Lian, Y. Huang, Y. Li, and J. Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pages 2737–2745, 2015.

[109] X. Lian, H. Zhang, C.-J. Hsieh, Y. Huang, and J. Liu. A comprehensive linear speedup analysis for asynchronous stochastic parallel optimization from zeroth-order to first-order. In *NIPS*, 2016.

[110] M. Lichman. UCI machine learning repository, 2013.

[111] F. Lin, Q. Ling, and Z. Xiong. Byzantine-resilient distributed large-scale matrix completion. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8167–8171. IEEE, 2019.

[112] J. Liu, Stephen, and J. Wright. Asynchronous stochastic coordinate descent: Parallelism and convergence properties. Technical report, SIAM Journal on Optimization, 2015.

[113] N. A. Lynch. *Distributed algorithms*. Morgan Kaufmann, 1996.

[114] D. M. MacKay and W. S. McCulloch. The limiting information capacity of a neuronal link. *The bulletin of mathematical biophysics*, 14(2):127–135, 1952.

[115] S. Mahloujifar, M. Mahmoody, and A. Mohammed. Multi-party poisoning through generalized p-tampering. *arXiv preprint arXiv:1809.03474*, 2018.

[116] J. Markoff. How many computers to identify a cat? 16,000. *New York Times*, pages 06–25, 2012.

[117] H. Markram, E. Muller, S. Ramaswamy, et al. Reconstruction and simulation of neocortical microcircuitry. *Cell*, 163(2):456–492, 2015.

[118] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282, 2017.

[119] K. Mehrotra, C. K. Mohan, S. Ranka, and C.-t. Chiu. Fault tolerance of neural networks. Technical report, DTIC Document, 1994.

[120] K. Mehrotra, C. K. Mohan, S. Ranka, and C.-t. Chiu. Fault tolerance of neural networks. Technical report, DTIC Document, 1994.

[121] H. Mendes and M. Herlihy. Multidimensional approximate agreement in byzantine asynchronous systems. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 391–400. ACM, 2013.

[122] H. Mendes, M. Herlihy, N. Vaidya, and V. K. Garg. Multidimensional agreement in byzantine systems. *Distributed Computing*, 28(6):423–441, 2015.

[123] J. Misra and I. Saha. Artificial neural networks in hardware: A survey of two decades of progress. *Neurocomputing*, 74(1):239–255, 2010.

[124] T. Mitra, S. Counts, and J. W. Pennebaker. Understanding anti-vaccination attitudes in social media. In *ICWSM*, pages 269–278, 2016.

[125] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1765–1773, 2017.

[126] A. S. Morcos, D. G. Barrett, N. C. Rabinowitz, and M. Botvinick. On the importance of single directions for generalization. In *International Conference on Learning Representations*, 2018.

[127] L. Muñoz-González, K. T. Co, and E. C. Lupu. Byzantine-robust federated machine learning through adaptive model averaging. *arXiv preprint arXiv:1909.05125*, 2019.

[128] E. Munsing and S. Moura. Cybersecurity in distributed and fully-decentralized optimization: Distortions, noise injection, and admm. *arXiv preprint arXiv:1805.11194*, 2018.

[129] A. F. Murray and P. J. Edwards. Enhanced mlp performance and fault tolerance resulting from synaptic weight noise during training. *IEEE Transactions on neural networks*, 5(5):792–802, 1994.

[130] M. Métivier. *Semi-Martingales*. Walter de Gruyter, 1983.

[131] S. Navlakha and Z. Bar-Joseph. Distributed information processing in biological and computational systems. *Communications of the ACM*, 58(1):94–102, 2015.

[132] C. Neti, M. H. Schneider, and E. D. Young. Maximally fault tolerant neural networks. *IEEE Transactions on Neural Networks*, 3(1):14–23, 1992.

[133] B. Neyshabur, S. Bhojanapalli, D. McAllester, and N. Srebro. Exploring Generalization in Deep Learning. *coRR*, 2017.

[134] B. Neyshabur, S. Bhojanapalli, and N. Srebro. A pac-bayesian approach to spectrally-normalized margin bounds for neural networks. *arXiv preprint arXiv:1707.09564*, 2017.

[135] B. Neyshabur, R. R. Salakhutdinov, and N. Srebro. Path-sgd: Path-normalized optimization in deep neural networks. In *Neural Information Processing Systems*, pages 2422–2430, 2015.

[136] W. H. Organization and R. Akbar. Ten threats to global health in 2019. https://www.who.int/emergencies/ten-threats-to-global-health-in-2019, 2019. [Online; accessed 21-January-2019].

[137] D. S. Phatak and I. Koren. Complete and partial fault tolerance of feedforward neural nets. *IEEE Transactions on Neural Networks*, 6(2):446–456, 1995.

[138] V. Piuri. Analysis of fault tolerance in artificial neural networks. *Journal of Parallel and Distributed Computing*, 61(1):18–48, 2001.

[139] V. Piuri. Analysis of fault tolerance in artificial neural networks. *Journal of Parallel and Distributed Computing*, 61(1):18–48, 2001.

[140] B. T. Polyak and A. B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

[141] A. Qiao, B. Aragam, B. Zhang, and E. P. Xing. Fault tolerance in iterative-convergent machine learning. *arXiv preprint arXiv:1810.07354*, 2018.

[142] S. Rajput, H. Wang, Z. Charles, and D. Papailiopoulos. Detox: A redundancy-based framework for faster and more robust gradient aggregation. *Neural Information Processing Systems*, 2019.

[143] B. Recht, C. Re, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Neural Information Processing Systems*, pages 693–701, 2011.

[144] S. J. Reddi, M. Zaheer, S. Sra, B. Poczos, F. Bach, R. Salakhutdinov, and A. J. Smola. A generic approach for escaping saddle points. *arXiv preprint arXiv:1709.01434*, 2017.

[145] P. J. Rousseeuw. Multivariate estimation with high breakdown point. *Mathematical statistics and applications*, 8:283–297, 1985.

[146] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319, 1990.

[147] H. T. Siegelmann and E. D. Sontag. Analog computation via neural networks. *Theoretical Computer Science*, 131(2):331–360, 1994.

[148] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[149] N. Smith and T. Graham. Mapping the anti-vaccination movement on facebook. *Information, Communication & Society*, pages 1–18, 2017.

[150] M. Y.-J. Song and A. Gruzd. Examining sentiments and popularity of pro-and anti-vaccination videos on youtube. In *Proceedings of the 8th International Conference on Social Media & Society*, page 17. ACM, 2017.

[151] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[152] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[153] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. In *Advances in neural information processing systems*, pages 2377–2385, 2015.

[154] J. Stilgoe. Machine learning, social learning and the governance of self-driving cars. *Social studies of science*, 48(1):25–56, 2018.

[155] L. Su. *Defending distributed systems against adversarial attacks: consensus, consensus-based learning, and statistical learning*. PhD thesis, University of Illinois at Urbana-Champaign, 2017.

[156] L. Su and S. Shahrampour. Finite-time guarantees for byzantine-resilient distributed state estimation with noisy measurements. *arXiv preprint arXiv:1810.10086*, 2018.

[157] L. Su and N. H. Vaidya. Fault-tolerant multi-agent optimization: optimal iterative distributed algorithms. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, pages 425–434. ACM, 2016.

[158] L. Su and N. H. Vaidya. Non-bayesian learning in the presence of byzantine agents. In *International Symposium on Distributed Computing*, pages 414–427. Springer, 2016.

[159] Y. Tan and T. Nanya. Fault-tolerant back-propagation model and its generalization ability. In *Neural Networks, 1993. Proceedings of 1993 International Joint Conference on*, volume 3, pages 2516–2519. IEEE, 1993.

[160] W. TianXiang, Z. ZHENG, T. ChangBing, and P. Hao. Aggregation rules based on stochastic gradient descent in byzantine consensus. In *2019 IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, pages 317–324. IEEE, 2019.

[161] R. Tomsett, K. Chan, and S. Chakraborty. Model poisoning attacks against distributed machine learning systems. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, volume 11006, page 110061D. International Society for Optics and Photonics, 2019.

[162] C. Torres-Huitzil and B. Girau. Fault and error tolerance in neural networks: A review. *IEEE Access*, 5:17322–17341, 2017.

[163] A. Trask, D. Gilmore, and M. Russell. Modeling order in neural word embeddings at scale. In *ICML*, pages 2266–2275, 2015.

[164] J. Tsitsiklis, D. Bertsekas, and M. Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE transactions on automatic control*, 31(9):803–812, 1986.

[165] P. Vyavahare, L. Su, and N. H. Vaidya. Distributed learning with adversarial agents under relaxed network condition. *arXiv preprint arXiv:1901.01943*, 2019.

[166] B. Wang, J. Gao, and Y. Qi. A theoretical framework for robustness of (deep) classifiers under adversarial noise. *arXiv preprint arXiv:1612.00334*, 2016.

[167] T.-W. Weng, P.-Y. Chen, L. M. Nguyen, M. S. Squillante, I. Oseledets, and L. Daniel. Proven: Certifying robustness of neural networks with a probabilistic approach. *arXiv preprint arXiv:1812.08329*, 2018.

[168] P. Werbos. *Beyond regression: New tools for prediction and analysis in the behavioral sciences*. Harvard University, 1974.

[169] Q. Xia, Z. Tao, Z. Hao, and Q. Li. Faba: An algorithm for fast aggregation against byzantine attacks in distributed neural networks.

[170] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli. Is feature selection secure against training data poisoning? In *International Conference on Machine Learning*, pages 1689–1698, 2015.

[171] C. Xie. Zeno++: robust asynchronous sgd with arbitrary number of byzantine workers. *arXiv preprint arXiv:1903.07020*, 2019.

[172] C. Xie, O. Koyejo, and I. Gupta. Generalized byzantine-tolerant sgd. *arXiv preprint arXiv:1802.10116*, 2018.

[173] C. Xie, O. Koyejo, and I. Gupta. Phocas: dimensional byzantine-resilient stochastic gradient descent. *arXiv preprint arXiv:1805.09682*, 2018.

[174] C. Xie, O. Koyejo, and I. Gupta. Zeno: Distributed stochastic gradient descent with suspicion-based fault-tolerance. *arXiv preprint arXiv:1805.10032*, 2018.

[175] C. Xie, S. Koyejo, and I. Gupta. Fall of empires: Breaking byzantine-tolerant sgd by inner product manipulation. *arXiv preprint arXiv:1903.03936*, 2019.

[176] H. Yang, X. Zhang, M. Fang, and J. Liu. Byzantine-resilient stochastic gradient descent for distributed learning: A lipschitz-inspired coordinate-wise median approach. *arXiv preprint arXiv:1909.04532*, 2019.

[177] Z. Yang and W. U. Bajwa. Bridge: Byzantine-resilient decentralized gradient descent. *arXiv preprint arXiv:1908.08098*, 2019.

[178] C. Yu, H. Tang, C. Renggli, S. Kassing, A. Singla, D. Alistarh, C. Zhang, and J. Liu. Distributed learning over unreliable networks. In *International Conference on Machine Learning*, pages 7202–7212, 2019.

[179] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. Avestimehr. Lagrange coded computing: Optimal design for resiliency, security and privacy. *arXiv preprint arXiv:1806.00939*, 2018.

[180] R. Zhang, S. Zheng, and J. T. Kwok. Asynchronous distributed semi-stochastic gradient optimization. In *AAAI*, pages 2323–2329, 2016.

[181] S. Zhang, A. E. Choromanska, and Y. LeCun. Deep learning with elastic averaging sgd. In *Advances in Neural Information Processing Systems*, pages 685–693, 2015.

[182] T. Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, page 116. ACM, 2004.

[183] W. Zhang, S. Gupta, X. Lian, and J. Liu. Staleness-aware async-sgd for distributed deep learning. In *IJCAI*, pages 2350–2356, 2016.

[184] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola. Parallelized stochastic gradient descent. In *Neural Information Processing Systems*, pages 2595–2603, 2010.

El Mahdi EL MHAMDI
Telephone : (+41) 76 587 53 84 -- Email : elmahdi@elmhamdi.com Webpage: www.elmahdielmhamdi.com

About:

My current research is mostly on the robustness of (distributed) learning systems, where I focus on algorithmic design and formal proofs, while closely collaborating with systems engineering colleagues. My contributions led to the first provably Byzantine-resilient algorithm for Gradient Descent and a series of follow-ups. I also have a keen interest in theoretical biology on which I collaborated with colleagues from the Johns Hopkins School of Medicine. Before my PhD, I worked as a Research Engineer in applied physics, co-founded a University-level e-Learning platform in Switzerland and an award winning News portal in my home country, Morocco.

Publications: (authors order is alphabetic in most papers, except when indicated with *, details in my website.)

Peer reviewed conferences and journals:

E.M. El Mhamdi, R. Guerraoui, A. Maurer, V. Tempez. *"Exploring the Borderlands of the Gathering Problem"*. Bulletin of the European Association of Theoretical Computer Science (**EATCS**). 2019. (I presented an earlier version of this work at the 5th Biological Distributed Algorithms, Washington D.C, July 2017).

G. Damaskinos, E.M. El Mhamdi, R. Guerraoui, A. Guirguis, S. Rouault. *"AggregaThor: Byzantine Machine Learning via Robust Gradient Aggregation"*. Conference on Machine Learning and Systems **MLsys** 2019. *Regular Talk (Given by G.D).*

E.M. El Mhamdi, R. Guerraoui, S. Rouault. *"The Hidden Vulnerability of Distributed Learning in Byzantium"*. International Conference on Machine Learning (**ICML**), Sweden. July 2018. ***Long Talk*** *(Given by myself).*

G. Damaskinos, E.M. El Mhamdi, R. Guerraoui, R. Patra, M. Taziki. *"Asynchronous Byzantine Machine Learning (the case of SGD)"*. (**ICML**), Sweden. July 2018. ***Long Talk*** *(Given by myself).*

El Mahdi El Mhamdi*, Andrei Kucharavy*, Rachid Guerraoui, Rong Li. *"Predicting complex genetic phenotypes using error propagation in weighted networks"*. Conferences of the European Molecular Biology Laboratory (**EMBL**) Heidelberg, Germany Nov 2018. *Short Talk (given by A.K.). (*Currently under review for a Biology journal, preprint: https://www.biorxiv.org/content/10.1101/487348v1*)*

E.M. El Mhamdi, R. Guerraoui, H. Hendrikx, A. Maurer. *"Dynamic Safe Interruptibility for Decentralized Multi-Agent Reinforcement Learning"*. Neural Information Processing Systems (**NeurIPS**), California. Dec 2017. ***Spotlight*** *(by H.H).*

P. Blanchard, E.M. El Mhamdi, R. Guerraoui, J.Stainer. *"Machine Learning in the Presence of Attackers: Byzantine-Tolerant Gradient Descent"*. Neural Information Processing Systems (**NeurIPS**), California. Dec 2017.

E.M. El Mhamdi, R. Guerraoui, S. Rouault. *"On The Robustness of a Neural Network"*. IEEE 38th Symposium on Reliable Distributed Computing (**SRDS**), Hong Kong, China. Sept. 2017. *Regular Talk (Given by myself).*

P. Blanchard, E.M. El Mhamdi, R. Guerraoui, J. Stainer. *"Byzantine-Tolerant Machine Learning"*. ACM Symposium on Principles of Distributed Computing (**PODC**), Washington D.C, USA. July 2017. *Short Talk (Given by myself).*

E.M. El Mhamdi and R. Guerraoui. *"When Neurons Fail"*. IEEE 31st International Parallel and Distributed Processing Symposium (**IPDPS**), Orlando, Florida, USA. June 2017. *Regular Talk (Given by myself).*

E.M. El Mhamdi*, J. Holovsky, B. Demaurex, C. Ballif and S. De Wolf. "*Is Light Induced Degradation of a-Si:H*/c-Si interfaces reversible?" **Applied Physics Letters**, June 2014. (APL is the leading journal in applied physics, condensed matter and semiconductors).

E.M. El Mhamdi*, S. De Wolf, J. Holovsky, B. Demaurex and C. Ballif. "*Metastability versus Irreversibility in a-Si:H Degradation - Understanding Passivation in Silicon Heterojunction Solar Cells* " 3rd International Conference on Crystalline Silicon Photovoltaics. March 2013, Hamelin, Germany. **Plenary Oral** (*Given by myself*).

D. Boullier and E.M. El Mhamdi. *"Machine learning and social sciences in the face of computational complexity"*. Accepted to the **Revue d'Anthropologie des Connaissances**, to appear, 2020.

R. Rößler*, L. Korte, C. Leendertz, N. Mingirulli, E.M El Mhamdi, B Rech. "*ZnO: Al/(p) a-Si: H Contact Formation and Its Influence on Charge Carrier Lifetime Measurements*". **EuPVSEC**, Sept. 2012, Frankfurt, Germany. *Regular Talk (by R.R).*

In preparation / workshops:

E.M. El Mhamdi and R. Guerraoui. *"Asynchronous Learning in the Optimal Window of Staleness". Under review*. In this work we look at asynchrony (in the absence of malicious nodes) as a "mild" adversary for the convergence of SGD.

E.M. El Mhamdi, R. Guerraoui, A. Guirguis. *"Fast Machine Learning with Byzantine Workers and Servers ". 2019*. This is a follow-up on my research on Byzantine-resilient workers-servers learning, now the servers could also be faulty. (synchronous setting). Preprint: https://arxiv.org/abs/1911.07537

E.M. El Mhamdi, R. Guerraoui, A. Guirguis, S. Rouault. *"SGD: Decentralized Byzantine Resilience". 2019*. This is a follow-up on my research on Byzantine-resilient workers-servers learning, now the servers could also be faulty. (asynchronous setting). Preprint: https://arxiv.org/abs/1905.03853

E.M. El Mhamdi, R. Guerraoui, A. Kucharavy, S. Volodin. (submitted). *"The Probabilistic Fault Tolerance of Neural Networks in the Continuous Limit". 2019*. Here we generalize the results of my IPDPS and SRDS 2017 papers on probabilistic error propagation (and fault tolerance) in neural networks. Preprint: https://arxiv.org/abs/1902.01686

E.M. El Mhamdi*, A. Kucharavy*, R. Guerraoui, R. Li. *"Essential genes as evolutionary dead-ends in biomolecular networks"*. 5th Biological Distributed Algorithms, Washington D.C, July 2017.

E.M. El Mhamdi, R. Guerraoui, L.N Hoang, A. Maurer. *"Removing Algorithmic Discrimination (with minimal individual error)". Preprint:* https://arxiv.org/abs/1806.02510.

H. Aslund, E.M. El Mhamdi, R. Guerraoui, A. Maurer. *"Virtuously Safe Reinforcement Learning". Preprint: https://arxiv.org/abs/1805.11447*. (Here we look back at our NeurIPS 2017 work on safe interruptibility, this time introducing an unreliable perception mechanism, we prove inevitable trade-offs).

E.M. El Mhamdi, R. Guerraoui, A. Maurer, V. Tempez. *"Learning to Gather Without Communication"* 5th Biological Distributed Algorithms, Washington D.C, July 2017. Preprint: https://arxiv.org/abs/1802.07834.

Other Academic activities:

Reviewing: NeurIPS 2018, 2019 (among top reviewers). ICML 2019, 2020. ICLR 2019, 2020. AAAI 2020. UAI 2019, 2020. DISC 2017 (external reviewer).

Grants: Wrote the research proposal part of two successful grants for a total 1.2 Million $ funding (PI: Prof. Rachid Guerraoui): (1) on Biological Distributed Algorithms (400k) and (2) on Robust Distributed Machine Learning (800k).

Patents:

E.M. El Mhamdi, R. Guerraoui, S. Rouault, M. Taziki. *"Strong Byzantine Tolerant Gradient Descent with Asynchrony for Distributed Machine Learning"*. Filed by EPFL on July. 10th 2018 under: PCT/EP2018/080812.

P. Blanchard, E.M. El Mhamdi, R. Guerraoui, J. Stainer. *"Byzantine Tolerant Gradient Descent for Distributed Machine Learning with Adversaries"*. Filed by EPFL on Nov. 29th 2017 under: PCT/EP2017/080806.

Book: "*The Fabulous Endeavor: Making Artificial Intelligence Robustly Beneficial*" written with Lê Nguyên Hoang.
A French version was published in November 2019 by academic publisher *EDP Sciences* under "*Le fabuleux chantier: rendre l'intelligence artificielle robustement bénéfique*". The English version is due for mid 2020.

Talks:

Academic talks:

During my PhD: UC Berkeley (invited by members of the CHAI group, 2019). ICML 2018 (**two long talks**), IPDPS 2017, SRDS 2017, PODC 2017, BDA 2016, Netys-Metis 2017 (invited). TDAH-Lausanne 2018 (invited). Beneficial AGI 2019 (invited).
Before my PhD:
- Silicon Photovoltaics 2013, Hamelin, Germany (**Plenary oral session**, to present my work in Physics).
- The association of learning technology (ALT) conference, University of Manchester, 2015

Industrial talks:

Google Brain Seattle, 2019 (invited by the Federated Learning team). **IBM Zurich**, 2019 (Distributed Computing & Machine Learning groups). **Ecocloud** 2019. **Applied Machine Learning Days** 2019 in the AI & Trust track. The **AI Governance Forum**, Geneva, 2019.

Awards:

PhD Thesis: My thesis is currently nominated for a best thesis award.

EPFL IC Research Day 2018: First runner-up presentation. Finalists were preselected among 60 PhD talks.

Google & Global Voices Online: Breaking Borders Award 2012 to *Mamfakinch*, a citizen-media I co-founded with fellow Moroccan bloggers following the 2011 so called "Arab Spring". (Google press release: http://goo.gl/dRcNvA).

EPFL: Excellence Scholarship for Masters studies, 2010-2012.

Military dean of the Ecole Polytechnique: "*Outstanding student for his leadership skills and capacity to draw people together in a collaborative project*"(OL) & "*Outstanding student who has distinguished himself through his dedication and commitment to the student body*"(OI). **Only 3%** of the students got a double award (OL & OI).

Procter and Gamble North Africa: Youngest participant and winner (team of 6) of the May 2008 edition of the business case-studies competition organized by Procter and Gamble (Talent'ger).

French Ministry of Foreign Affairs: Academic excellence scholarship, 2007-2010.

Professional Experience:

Research and Teaching Assistant
Sep. 2015 – Present: **Distributed Computing Laboratory** – Lausanne, Switzerland

Teaching assistant for three graduate courses in Distributed Computing and Optimization for Machine Learning. Currently developing and teaching a full graduate-level introductory course on Machine Learning in Morocco.

Technical/Scientific reviewer
Dec. 2016 – Jan. 2017: **The African Innovation Foundation** – Zurich, Switzerland / Accra, Ghana

Reviewed and wrote reports on ~100 technical applications for the annual grant share-prize of USD 185.000 with competitors from 42 different african countries http://innovationprizeforafrica.org/news06-14-17.html

In charge of the Wandida online education project
Jun. 2013 – Sept. 2015: **Swiss Federal Institute of Technology** (EPFL) – Lausanne, Switzerland

Co-founded the online learning project **Wandida** with Prof. R. Guerraoui.
Wrote and obtained grants from third-party organizations, including **Google.**
Involved high profiles such as French Academy of Sciences members Serge Abiteboul and Gérard Berry.

Research Engineer
2012 – 2013: **Photovoltaics lab.**, Neuchâtel, Switzerland

Implemented a model for photoluminescence reaction of solar cells to laser pulses in a physical lab-setup.
Investigated light induced degradation (LID). Established the first-evidence of a non-reversible component of the Staebler-Wronsky effect. Published in the **Applied Physics Letters of the American Institute of Physics.** http://scitation.aip.org/content/aip/journal/apl/104/25/10.1063/1.4885501

Research Intern
April - August 2010: **Helmholtz Zentrum Berlin für Materialien und Energie**, Berlin, Germany.

Education:

PhD in Computer and Communication Sciences
Sep. 2015- Nov. 2019: **Swiss Federal Institute of Technology**, EPFL, Lausanne, Switzerland
Currently nominated for a best thesis award. Jury: Francis Bach (ENS Ulm), Martin Jaggi (EPFL), Maurice Herlihy (Brown University), Rachid Guerraoui (EPFL). President: Babak Falsafi (EPFL).

MSc in Material Sciences
2010- 2012: **Swiss Federal Institute of Technology**, EPFL, Lausanne, Switzerland
Focused on condensed-matter and semiconductors physics.

Polytechnicien
2007 - 2010: **Ecole Polytechnique,** Paris, France
Double major in Physics and Mathematical Economics.
Co-Organizer of the 2009 Caroline Aigle Triathlon.
Treasurer of the Ecole Polytechnique Diving Club, managing a 30 000 € annual budget.

Mathematics, Physics and Philosophy Bachelor-level program
2004 - 2007: **Mohammed V**, Casablanca, Morocco.
Admitted to the Ecole Polytechnique of Paris (the only admitted out of ~250 candidates from Casablanca).

Languages: Fluent in Arabic, French and English; Basics of Spanish and Maltese.

Extra Curricular / MISC:

Sports : Winner (single) of the Lausanne 24 hours swimming challenge in 2011 (**36 km non-stop, 1096 participants**, supported by the International Olympic Committee). (http://goo.gl/9mZbTW).
Five times Finisher (2011 to 2016) of the "Tour du Léman" world's longest rowing race in a closed area (**160 km** non-stop in about 16 hours). Participants of the Tour include Olympic gold medallist Tim Grohmann and world Champion Patrik Stöcker. In 2013, I was vice-champion of Switzerland for Masters-A mixed-boats in a team of 4.

Jury member / Coach for December 2013' HackXplor hackathon, organized by the Wallonia Export and Investment Agency, Belgium. Helped organizing following editions of hackXplor (December 2013 in Liège, **Belgium**, April 2014 in Meknes, **Morocco**, November 2014 in Dakar, **Senegal**) (https://goo.gl/t93u2W).

Author in *Futurechallenges.org*, Germany: Web-based project gathering more than a hundred bloggers, FutureChallenges.org won the 2012 German Online Communication Award (https://goo.gl/jCkgzQ).

Coach and Educator at the Fondation d'Auteuil, Château des Vaux, France : Ecole Polytechnique's Human & Military Training in an institution for teenagers in social difficulty.

Columnist (sporadic) : When time allowed (up to ~2016), I used to write for some media outlets, such as here for ***Le Monde*** (French) https://goo.gl/T9z9By or the moroccan **Medias24** (Arabic and French) https://goo.gl/kq2GRC.