Compression and federated learning : an approach to frugal machine learning

Louis Leconte

Sorbonne Université
Huawei Technologies France

Doctoral School **École Doctorale Informatique, Télécommunications et Electronique**

University Department **Laboratoire LISITE**

Thesis defended by **Leconte Louis**

Defended on **June 5, 2024**

In order to become Doctor from Sorbonne Université

Academic Field **Computer science**

Compression and Federated Learning: an approach to frugal machine learning

Thesis supervised by	Lionel Trojman	Supervisor
	Eric Moulines	Co-Supervisor
	Van Minh Nguyen	Co-Monitor

Committee members

Referees	Aurélien Bellet	Senior Researcher at Inria Montpellier	
	Karim Abed-Meraim	Professor at University of Orleans	
Examiner	Florence d'Alché-Buc	Professor at Télécom Paris	Committee President
Supervisors	Lionel Trojman	Professor at ISEP	
	Eric Moulines	Professor at Ecole Polytechnique	
	Van Minh Nguyen	Senior Researcher at Huawei Technologies France	

Sorbonne Université
Huawei Technologies France

Doctoral School **École Doctorale Informatique, Télécommunications et Electronique**

University Department **Laboratoire LISITE**

Thesis defended by **Leconte Louis**

Defended on **June 5, 2024**

In order to become Doctor from Sorbonne Université

Academic Field **Computer science**

Compression and Federated Learning: an approach to frugal machine learning

Thesis supervised by	Lionel Trojman	Supervisor
	Eric Moulines	Co-Supervisor
	Van Minh Nguyen	Co-Monitor

Committee members

Referees	Aurélien Bellet	Senior Researcher at Inria Montpellier	
	Karim Abed-Meraim	Professor at University of Orleans	
Examiner	Florence d'Alché-Buc	Professor at Télécom Paris	Committee President
Supervisors	Lionel Trojman	Professor at ISEP	
	Eric Moulines	Professor at Ecole Polytechnique	
	Van Minh Nguyen	Senior Researcher at Huawei Technologies France	

Sorbonne Université
Huawei Technologies France

École doctorale **École Doctorale Informatique, Télécommunications et Electronique**

Unité de recherche **Laboratoire LISITE**

Thèse présentée par **Leconte Louis**

Soutenue le **5 juin 2024**

En vue de l'obtention du grade de docteur de Sorbonne Université

Discipline **Informatique**

Compression et apprentissage Fédéré : une approche pour l'apprentissage machine frugal

Thèse dirigée par

Lionel Trojman	directeur
Eric Moulines	co-directeur
Van Minh Nguyen	co-encadrant

Composition du jury

Rapporteurs	Aurélien Bellet	directeur de recherche à l'Inria Montpellier	
	Karim Abed-Meraim	professeur à l'University of Orleans	
Examinatrice	Florence d'Alché-Buc	professeure au Télécom Paris	présidente du jury
Directeurs de thèse	Lionel Trojman	professeur à l'ISEP	
	Eric Moulines	professeur à l'Ecole Polytechnique	
	Van Minh Nguyen	chercheur au Huawei Technologies France	

Sorbonne Université and Huawei Technologies France neither endorse nor censure authors' opinions expressed in the theses: these opinions must be considered to be those of their authors.

Keywords: federated learning, quantized neural networks

Mots clés : apprentissage fédéré, réseaux de neurones quantifiés

The author warmly thank Lionel Trojman for his support and technical help regarding the organization of the mentoring. This thesis has been prepared at

Laboratoire LISITE
ISEP
Sorbonne Université

Web Site https://en.isep.fr/research_at_isep/

Apprendre n'est pas savoir; il y a les apprenants et les savants. La mémoire fait les uns, philosophe les autres.

Alexandre Dumas. "Le Comte de Monte-Cristo".

D'autres avant moi ont dit, d'autres après moi diront à quel point s'est élargi le fossé entre les peuples nantis et ceux qui n'aspirent qu'à manger à leur faim, boire à leur soif, survivre et conserver leur dignité. Mais nul n'imaginera à quel point le grain du pauvre a nourri chez nous la vache du riche.

Thomas Sankara.

Compression and Federated Learning: an approach to frugal machine learning

Abstract

"Intelligent" devices and tools are gradually becoming the standard, as the implementation of algorithms based on artificial neural networks is experiencing widespread development. Neural networks consist of non-linear machine learning models that manipulate high-dimensional objects and obtain state-of-the-art performances in various areas, such as image recognition, speech recognition, natural language processing, and recommendation systems.

However, training a neural network on a device with lower computing capacity can be challenging, as it can imply cutting back on memory, computing time or power. A natural approach to simplify this training is to use *quantized neural networks*, whose parameters and operations use efficient low-bit primitives. However, optimizing a function over a discrete set in high dimension is complex, and can still be prohibitively expensive in terms of computational power. For this reason, many modern applications use a network of devices to store individual data and share the computational load. A new approach, *federated learning*, considers a distributed environment: Data is stored on devices and a centralized server orchestrates the training process across multiple devices.

In this thesis, we investigate different aspects of (stochastic) optimization with the goal of reducing energy costs for potentially very heterogeneous devices. The first two contributions of this work are dedicated to the case of quantized neural networks. Our first idea is based on an annealing strategy: we formulate the discrete optimization problem as a constrained optimization problem (where the size of the constraint is reduced over iterations). We then focus on a heuristic for training binary deep neural networks. In this particular framework, the parameters of the neural networks can only have two values. The rest of the thesis is about efficient federated learning. Following our contributions developed for training quantized neural network, we integrate them into a federated environment. Then, we propose a novel unbiased compression technique that can be used in any gradient based distributed optimization framework. Our final contributions address the particular case of asynchronous federated learning, where devices have different computational speeds and/or access to bandwidth. We first propose a contribution that reweights the contributions of distributed devices. Then, in our final work, through a detailed queuing dynamics analysis, we propose a significant improvement to the complexity bounds provided in the literature on asynchronous federated learning.

In summary, this thesis presents novel contributions to the field of quantized neural networks and federated learning by addressing critical challenges and providing innovative solutions for efficient and sustainable learning in a distributed and heterogeneous environment. Although the potential benefits are promising, especially in terms of energy savings, caution is needed as a rebound effect could occur.

Keywords: federated learning, quantized neural networks

Laboratoire LISITE
ISEP – Sorbonne Université – – – –

Compression et apprentissage Fédéré : une approche pour l'apprentissage machine frugal

Résumé

Les appareils et outils "intelligents" deviennent progressivement la norme, la mise en œuvre d'algorithmes basés sur des réseaux neuronaux artificiels se développant largement. Les réseaux neuronaux sont des modèles non linéaires d'apprentissage automatique avec de nombreux paramètres qui manipulent des objets de haute dimension et obtiennent des performances de pointe dans divers domaines, tels que la reconnaissance d'images, la reconnaissance vocale, le traitement du langage naturel et les systèmes de recommandation.

Toutefois, l'entraînement d'un réseau neuronal sur un appareil à faible capacité de calcul est difficile en raison de problèmes de mémoire, de temps de calcul ou d'alimentation. Une approche naturelle pour simplifier cet entraînement consiste à utiliser des *réseaux neuronaux quantifiés*, dont les paramètres et les opérations utilisent des primitives efficaces à faible bit. Cependant, l'optimisation d'une fonction sur un ensemble discret en haute dimension est complexe et peut encore s'avérer prohibitive en termes de puissance de calcul. C'est pourquoi de nombreuses applications modernes utilisent un réseau d'appareils pour stocker des données individuelles et partager la charge de calcul. Une nouvelle approche a été proposée, l'*apprentissage fédéré*, qui prend en compte un environnement distribué : les données sont stockées sur des appareils différents et un serveur central orchestre le processus d'apprentissage sur les divers appareils.

Dans cette thèse, nous étudions différents aspects de l'optimisation (stochastique) dans le but de réduire les coûts énergétiques pour des appareils potentiellement très hétérogènes. Les deux premières contributions de ce travail sont consacrées au cas des réseaux neuronaux quantifiés. Notre première idée est basée sur une stratégie de recuit : nous formulons le problème d'optimisation discret comme un problème d'optimisation sous contraintes (où la taille de la contrainte est réduite au fil des itérations). Nous nous sommes ensuite concentrés sur une heuristique pour la formation de réseaux neuronaux profonds binaires. Dans ce cadre particulier, les paramètres des réseaux neuronaux ne peuvent avoir que deux valeurs. Le reste de la thèse s'est concentré sur l'apprentissage fédéré efficace. Suite à nos contributions développées pour l'apprentissage de réseaux neuronaux quantifiés, nous les avons intégrées dans un environnement fédéré. Ensuite, nous avons proposé une nouvelle technique de compression sans biais qui peut être utilisée dans n'importe quel cadre d'optimisation distribuée basé sur le gradient. Nos dernières contributions abordent le cas particulier de l'apprentissage fédéré asynchrone, où les appareils ont des vitesses de calcul et/ou un accès à la bande passante différents. Nous avons d'abord proposé une contribution qui repondère les contributions des dispositifs distribués. Dans notre travail final, à travers une analyse détaillée de la dynamique des files d'attente, nous proposons une amélioration significative des bornes de complexité fournies dans la littérature sur l'apprentissage fédéré asynchrone.

En résumé, cette thèse présente de nouvelles contributions au domaine des réseaux neuronaux quantifiés et de l'apprentissage fédéré en abordant des défis critiques et en fournissant des solutions innovantes pour un apprentissage efficace et durable dans un environnement distribué et hétérogène. Bien que les avantages potentiels soient prometteurs, notamment en termes d'économies d'énergie, il convient d'être prudent car un effet rebond pourrait se produire.

Mots clés : apprentissage fédéré, réseaux de neurones quantifiés

Remerciements

De ma petite expérience de chercheur je retiens une chose: bien souvent les étudiant·es, famille, et autres curieux·ses lisent en premier les Remerciements. Parfois la lecture s'arrête ici. Alors je vais oser en profiter pour dresser un portrait plus personnel de la recherche et de son (in)adéquation avec l'effondrement du vivant que nous vivons actuellement. Je présente par avance mes excuses à cell·eux qui s'attendaient à lire leurs noms ici.

Les quelques lignes qui suivent ne remettent en cause ni le travail, ni les collaborations réalisées pendant les trois dernières années, mais sont le résultat d'une réflexion à propos de l'effet rebond. En pratique c'est un phénomène très simple que l'on voit facilement dans notre vie de tous les jours. Avant de commencer ma thèse on m'avait présenté le sujet comme une nouvelle méthode qui permet d'entraîner des réseaux de neurones plus rapidement, et en consommnant moins d'énergie. Belle idée! Mais en pratique l'effet rebond est déjà là. Je prends souvent un exemple lorsque j'explique les limites de ma thèse: prenez un service d'écoute musical en ligne, il y a quelques années on écoutait juste de la musique en qualité standard, et on acceptait quelques images de publicités. Aujourd'hui la même application nous propose une qualité d'écoute profesionnelle, les publicités sont devenues des vidéos promotionnelles, et en plus on peut avoir accès en direct aux paroles de la chanson. En définitive, pour le même service, on consomme beaucoup plus d'énergie. Voilà une illustration de l'effet rebond: avant même que nos nouveaux algorithms soient mis en place (pour consommer un petit peu moins d'énergie), on se permet de consommer déjà beaucoup plus par nos usages. Au final, on consomme plus d'énergie.

Aujourd'hui l'écologie est un mot éminemment politique. Mais on se retrouve confronté avant tout à un problème simplement physique. On vit sur une planète finie. Qu'on le veuille ou non il y n'y aura bientôt plus de pétrole disponible car: (i) le pétrole ne pousse pas sur les arbres, (ii) on a déjà passé les pics de production des énergies fossiles (2008 et 2018 pour le pétrole conventionnel[1] et le gaz, respectivement). Cette consomation fossile, en plus de ses limites physiques, a évidemment des conséquences désastreuses sur notre environnement. La combustion des ressources fossiles accroît significativement la température du globe terrestre, et accompagne la destruction des milieux que l'on (sur)exploite: artificialisation des sols, éradication massive des insectes pollinisateurs, chute de la qualité de l'air... Non content de ce bilan, le mode de vie occidental (cf. citation en page xiii de ce manuscrit) pèse avant tout sur les populations et les pays les plus pauvres, au profit des populations/pays riches.

Alors que faire? On pourrait être tenté de ne rien faire. Et de continuer à prendre à l'avion sans réfléchir, de continuer à manger de la viande tous les jours, de continuer à acheter sans voir les alternatives (friperies, ateliers de réparation, etc.). Ce qui paraît être une lubie de bobo écolo n'est en fait qu'une inspiration de ce qui était la norme, même pour les personnes riches, il y seulement quelques dizaines d'années. Aujourd'hui, on peut ausi choisir, chacun·e à son échelle. On peut choisir politiquement (carte électorale), on peut choisir citoyennement (carte

[1] https://fr.wikipedia.org/wiki/Pic_pétrolier

bancaire). D'aucun·es diront que ce sont les grandes entreprises, les élus, les institutions qui doivent d'abord changer. Mais ces dernières ne sont bien souvent que le reflet de nos décisions (politiques *et* citoyennes, encore une fois). On ne peut pas forcer les agriculteur·ices français·es à produire en bio à leurs frais si les français·es continuent à consommer aveuglément du boeuf élevé dans des conditions douteuses à l'autre bout de la planète. On ne peut pas demander à une usine française de se fournir en énergies renouvelables si les français·es continuent à (sur)consommer des habits produits à l'autre bout du monde.

Et la recherche dans tout ça? On pourrait: (i) Interdir tous les déplacements en avion pour des conférences. La visio s'est bien implantée dans nos usages, et cela donnerait un coup d'accélérateur à toutes les initiatives accessibles en train. On a tendance à sous-estimer les distances que l'on peut parcourir sur voies ferrées[2]. (ii) Rendre obligatoire une estimation de l'impact environnemental des articles soumis, et la prendre en compte dans la notation. Certaines conférences prestigieuses (Aistats, Neurips) imposent déjà des "check-list" pour juger de la reproducibilité des résultats. (iii) Proposer un cours sur les limites physiques des ressources énergétiques à tou·tes les étudiant·es en master, futur·es chercheur·ses. (iv) Rendre accessible tous les enseignements pour les pays/populations pauvres. Vous trouverez des propositions, certes plus radicales, mais pour le moins étayées tant d'un point de vue scientifique que social, ici[3] et là[4].

[2] https://www.seat61.com/

[3] https://polaris.imag.fr/romain.couillet/index.html

[4] https://vous-netes-pas-seuls.org/

Glossary and symbols

NN Neural Network
DNN Deep Neural Network
BNN Binary Neural Network
QNN Quantized Neural Network
VQ Vector Quantization
FL Federated Learning
CS Central Server
OPs Operations (arithmetic)
FLOPs Floating point Operations
BOPs Binary Operations
w Parameters (weights and biases) of a NN
$:=$ Defined as
$\mathbb{1}$ Indicator/characteristic function
$\mathbb{R}$ Set of real number
$\mathbb{R}^d$ Set of d-dimensional real-valued vectors
$\langle x, y \rangle$ Inner product of vectors $x, y \in \mathbb{R}^d$
$x \wedge y$ Minimum of $x, y \in \mathbb{R}$
$\|x\|$ Euclidean norm of vector $x \in \mathbb{R}^d$
$\mathbb{E}[X]$ Expectation of a random variable X
∇f Gradient function of $f : \mathbb{R}^d \to \mathbb{R}^d$
n Number of nodes/clients
d Dimension (number of weights/biases)
T Number of optimization steps
$|\mathcal{D}|$ Number of data samples in a given dataset $\mathcal{D}$

Contents

List of Tables

List of Figures

Overview of the contributions

The chapters of this manuscript (except the introductory Chapter 1) are based on the contributions that were presented at scientific conferences during the doctoral program. The next 5 pages assume that the reader has a good understanding of optimization, federated learning, and quantization mechanisms. If necessary, the reader may choose to first read Chapter 1 before returning to this summary.

Context

Machine Learning (ML) is an expanding discipline that is revolutionizing our relationship with technology. Deep neural networks (DNNs) have demonstrated remarkable achievements in various fields, such as image recognition, speech recognition, natural language processing, recommendation systems, and more. ML algorithms employ statistical and optimization techniques to detect patterns in data, with the ultimate aim of transferring these patterns to new, unseen data. The effectiveness of ML algorithms is highly dependent on the quality and quantity of data used to train them.

Machine learning and frugality

Frugality is the characteristic of being mindful, cautious, judicious, and economical in the consumption of resources, while avoiding waste, extravagance, or excess. Historically, artificial intelligence (AI) has primarily existed in the form of cloud-based software installed on large servers. However, the future of AI is shifting towards localization, with AI being embedded directly on Internet of Things (IoT) devices such as sensors, and other equipment. This trend towards moving AI off the cloud and onto the edge is becoming increasingly prevalent. In order to be effective in critical systems with high safety, privacy, robustness, and reliability requirements, trust must be built into the design of edge AI and validated through testing. The deployment of edge AI in constrained, networked environments presents additional challenges, including the need for solutions with low data, computing, memory, and power budgets, as well as efficient, secure operation in such scenari.

It is trite to say that Deep Neural Networks (DNNs) are now present in every commercial automated tasks. But every signal, image, or video is now automatically processed by an artificial neural network. While low-tech equivalent models can sometimes deliver better performances (Couillet, Trystram, and Ménissier 2022), all smartphones and computers analyse a huge amount of data everyday through the inference phase (also known as forward phase) of their embedded DNNs. Fine-tuning a DNN is also possible (through the computation of the backward and updates phase) when a local dataset is available. However, training a neural network on a unit with a reduced computing capacity is difficult, due to problems of memory, computing time or energy requirements. A natural approach to make this training simpler is to use Quantized Neural Networks (QNNs), whose activations and operations use efficient low-bit primitives. Nevertheless, QNNs are by nature discrete, whereas a standard neural network is optimized using tools with almost everywhere differentiable functions. Moreover, decreasing the representation bits coding has a tremendous degrading effect in accuracy. For a given number of neurons, each neuron suffers from the quantization step introduced, and therefore inherent errors may appear.

Machine learning in a connected world

The application of machine learning (ML) algorithms presents challenges in many settings due to data protection concerns. Personal data, such as health records, financial information, and personal communications, can contain sensitive information that individuals may not wish to share publicly. Furthermore, regulations such as the General Data Protection Regulation (GDPR) restrict the ways in which companies and organizations can handle personal data.

A non-exhaustive list of scenari where machine learning can be used with sensitive data includes mobile devices (recommendation systems, speech recognition, next-word prediction), healthcare (effectiveness of different treatments, drug design, medical imaging), and autonomous

driving. In these scenari, data is gathered from user interactions with mobile devices, health institutions, and vehicles equipped with sensors. Federated learning (FL) is a distributed learning paradigm that allows a set of decentralized devices to collaboratively train a model without ever sharing their data. The training process is orchestrated by a central server, which does not have access to the data. This allows for the privacy of each client to be respected. FL enables the use of diverse and large datasets, which can improve the robustness and generalization of machine learning models. Additionally, FL distributes the computational load of training across a large pool of devices, making it practical and efficient.

In this work, we focus on the optimization aspect of FL and QNN. Our proposed methods are compatible with state-of-the-art privacy-preserving protocols (Huba et al. 2022), and with standard software. Our work enables more efficient federated and/or standard training with less energy consumption and less computational power.

Contributions of this Thesis

We introduce and summarize our contributions in the following. In all the document, we consider optimization problems in which the components of the objective function (i.e., the data for machine learning problems) are distributed over n clients, i.e.,

$$\min_{w \in \mathrm{Q}(d,M)} f(w);\ f(w) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}_{(x,y)\sim D_i^{\mathrm{data}}} [\ell(\mathrm{nn}(x,w),y)], \tag{1}$$

where d is the number of parameters (network weights and biases), n is the total number of clients, ℓ is the training loss (e.g., cross-entropy or quadratic loss), $\mathrm{nn}(x,w)$ is the DNN prediction function, $\mathrm{Q}(d,M) \subset \mathbb{R}^d$ is a set of constraints with finite cardinal M. We assume that each of the n clients can sample a local dataset $\mathcal{D}_i$ with distribution D_i^{data}. In FL, the distributions D_i^{data} are allowed to differ between clients (statistical heterogeneity). This general formulation takes into account the standard *non-federated* setting (by taking $n = 1$), and also encompasses the *unconstrained* learning case (by taking $\mathrm{Q}(d,M) = \mathbb{R}^d$).

An annealing process to obtain QNN

As a first step in the efficient deep learning domain, this thesis addresses the following question the following: is it possible to learn a neural network with quantized weights/activations, with guarantees? By guarantee we follow the "weak", yet standard criterion for DNN: is it possible to obtain a first-order stationary point on a quantized grid? If yes, we hope that this stationary point yields good performances. As such, in Chapter 2, we develop a new algorithm, Annealed Skewed SGD - `AskewSGD` - for training deep neural networks (DNNs) with quantized weights. First, we formulate the training of quantized neural networks (QNNs) as a smoothed sequence of interval-constrained optimization problems. Then, we propose a new first-order stochastic method, `AskewSGD`, to solve each constrained optimization sub-problem. Unlike algorithms with active sets and feasible directions, `AskewSGD` avoids projections or optimization under the entire feasible set and allows iterates that are infeasible. The numerical complexity of `AskewSGD` is comparable to existing approaches for training QNNs, such as the straight-through gradient estimator used in BinaryConnect, or other state of the art methods (ProxQuant, LUQ). We establish convergence guarantees for `AskewSGD` (under general assumptions for the objective function). Experimental results show that the `AskewSGD` algorithm performs better than or on par with state of the art methods in classical benchmarks. Overall, our contributions are to:

- Replace the discrete optimization (1) by an annealed sequence of differentiable inequality constraints that converges to (1) when the annealing parameter goes to 0.
- Use a novel first-order algorithm proposed in Muehlebach and Jordan 2021 to solve the relaxed sub-problems in the annealed sequence, leading to `AskewSGD`. Unlike classical constrained optimization algorithms, including the projection method or sequential quadratic programming (Gill and Wong 2012), this approach relies exclusively on local approximations of the feasible set. This local approximation includes only the active constraints, and is guaranteed to be a convex polyhedron even if the underlying constraint set is non convex. This makes the resulting algorithm easy to implement and also ensures that the descent is not stopped as soon as a new constraint is violated.
- Show how `AskewSGD` can be applied to train QNN. The complexity of the resulting algorithm is similar to that of BC or LUQ (Courbariaux, Bengio, and David 2015; Chmiel et al. 2021) and ProxQuant (Bai, Y.-X. Wang, and Liberty 2018). Our algorithm uses high precision latent weights and uses classical backpropagation to evaluate the gradients.
- Provide convergence guarantees for `AskewSGD`. We stress that, as opposed to Muehlebach and Jordan 2021, no convexity assumption on the objective function or the feasible set is made.
- Evaluate the performance of `AskewSGD` on classical computer vision datasets using ConvNets and ResNets. Our experiments show that QNNs trained with `AskewSGD` achieve accuracy very close to that of their floating-point counterparts, and outperform or are on par with comparable baselines.

The previously mentioned contributions are detailed in Chapter 2, and have led to a publication at the AISTATS2023 conference (Leconte, Schechtman, and Moulines 2023).

A heuristic to obtain BNN

The Quantized neural network (QNN) literature covers a very broad topic, and several signal processing techniques can be combined to improve the training and/or inference stage of a QNN. In Chapter 3, we focus on the restrictive, yet efficient use case of BNNs (Binary Neural Networks). By "efficient", we refer to the energy efficiency of the training steps. In particular, most of the research works have been dedicated to the reduction of the arithmetic complexity, while energy consumption is the most relevant bottleneck. In addition, the literature focus has been on inference whereas training is several times more intense. In Chapter 3, we make use of the Boolean neuron design (V. M. Nguyen 2023) to train deep models in the binary domain using Boolean logic instead of gradient descent and real arithmetic. We propose a detailed energy evaluation for both training and inference phases. Our method achieves the best results in standard image classification tasks and consumes almost less energy (compared to a 16 bits neural network). This energy efficiency paves the way for an edge device use, in particular for fine-tuning large models on a dedicated task. The contributions related to this heuristic to train BNNs are mainly technical and experimental:

- We adapt the initial Boolean strategy (V. M. Nguyen 2023) to the case of deep NNs by introducing a learning factor inspired from brain plasticity: "neurons that fire together wire together" (Fuchs, Flügge, et al. 2014; Hebb 2005).
- We show competitive or state-of-the-art results of this design in complex computer vision-related tasks, including image super-resolution.

- We show that employing pre-trained Boolean NNs for edge device fine-tuning tasks, such as classification and segmentation, yields very good performances at low energy cost.

Under an industrial mentoring, the Boolean idea was presented in a patent (V. M. Nguyen and Leconte 2022), and is currently under review at the CVPR2024 conference.

Efficient BNN training in the FL context

After focusing on efficient NN training on a single device, we have considered the challenging problem of training jointly a set of nodes (e.g. edge devices with low computational and communication capacity) in the synchronous Federated Learning (FL) context. In standard centralized FL, one assumes that (i) all nodes can compute gradients on their local dataset (i.e. energy/memory is not a bottleneck), and (ii) all nodes are synchronous with the central server (i.e. bandwidth and compute time are not bottlenecks). In Chapter 4 we demonstrate how one can integrate the method introduced in Chapter 3 into the FL framework. Despite its popularity, federated learning faces the increasingly difficult task of scaling communication over large wireless networks with limited bandwidth. Moreover, this distributed training paradigm requires clients to perform intensive computations for multiple iterations, which may exceed the capacity of a typical edge device with limited processing power, storage capacity, and energy budget. Therefore, practical deployment of FL requires a balance between energy efficiency due to resource constraints and latency due to bandwidth constraints. In this work, we overcome both constraints by integrating low-precision arithmetic on clients and exchanging only highly compressed vectors during training. Here we assume nodes are edge devices with constrained computational power, and we consider 2 options to tackle the bandwidth bottleneck: `FedBool` and `MajBool`. Experimental results show that the proposed algorithms `FedBool` and `MajBool` perform better than current methods on standard image classification tasks. In Chapter 4, we will give more details about:

- `FedBool`, a new algorithm for federated learning with a central server, using a Boolean neural network on clients that compute only backpropagated signals (no full accuracy updates required).

- `MajBool`, another novel federated learning method where only binary vectors are exchanged (we propose a new aggregation rule based on majority logic to update the binary weights of the central server).

- Experimental results to showcase that our approaches consistently perform better than other federated baselines for quantized neural networks.

These algorithms were presented in a patent, and at the FMEC/FLTA2023 conference (Leconte, Moulines, et al. 2023).

An unbiased quantization scheme for highly compressed FL communications

The algorithms introduced in the previous paragraph are well suited to train NNs on low capacity devices with a small bandwidth usage. However bandwidth consumption is also a difficulty regarding the training of full precision NNs. Indeed, the growing size of models and datasets have made distributed implementation of stochastic gradient descent (SGD) an active field of research. But the high bandwidth cost of communicating gradient updates between nodes

remains an impediment; lossy compression is a way to alleviate this problem. In Chapter 5, we present an *unbiased* vector quantization (VQ) method, named Stovoq, to deal with the high bandwidth cost of communicating gradient updates between nodes. This technique has the following characteristics:

- It relies on unitarily invariant random codebooks and on a straightforward bias compensation method.
- It presents a better distortion rate compared to existing methods in the FL literature.
- It allows significant reduction of bandwidth use while preserving performance on convex and non-convex deep learning problems.

FL with different node computation capacities, and a central clock

In the previous paragraphs, we have introduced solutions to work with limited nodes and a limited bandwidth in the FL context. In the final chapters, we consider the asynchronous FL setting (see Section 1.4 for further details): we *do not* assume anymore that nodes contribute to the CS at the same pace. In particular, nodes can have very different computational speeds and/or a very different access to the bandwidth. This creates a serious impediment: in practical scenari, one does not want to wait for the slowest node at every central server (CS) update. In Chapter 6, we detail a novel centralized Asynchronous Federated Learning (FL) framework, FAVANO, for training Deep Neural Networks (DNNs) in resource-constrained environments. Despite its popularity, "classical" federated learning faces the increasingly difficult task of scaling synchronous communication over large wireless networks. Moreover, clients typically have different computing resources and therefore computing speed, which can lead to a significant bias (in favor of "fast" clients) when the updates are asynchronous. Therefore, practical deployment of FL requires to handle users with strongly varying computing speed in communication/resource constrained setting. FAVANO is:

- an unbiased aggregation scheme for centralized federated learning with asynchronous communication. Our algorithm does not assume that clients computed the same number of epochs while being contacted.
- backed up with complexity bounds in the smooth non-convex setting. We emphasize that the dependence of the bounds on the total number of agents n is improved compared to Zakerinia et al. 2022 and does not depend on a maximum delay.
- Experimental results show that our approach consistently outperforms other asynchronous baselines on the challenging TinyImageNet dataset (Y. Le and X. Yang 2015).

FAVANO (Leconte, V. M. Nguyen, and Moulines 2023) has been accepted for publication at the ICASSP2024 conference.

Queuing dynamics in the context of FL with different node computation capacities

For our final contribution, we have developed another novel method to deal with asynchronous FL. This scheme is orthogonal to the algorithm FAVANO presented in Chapter 6. Indeed, as detailed in Chapter 7, nodes are not interrupted by the central server (CS). However, each node is allowed to work on models with potential delays and contribute to updates to the CS at its own pace. We study asynchronous federated learning mechanisms with nodes having potentially

different computational speeds. Existing analyses of such algorithms typically depend on intractable quantities such as the maximum node delay, and do not consider the underlying queuing dynamics of the system. In this paper, we propose a non-uniform sampling scheme for the central server that allows for lower delays with better complexity, taking into account the closed Jackson network structure of the associated computational graph. Our experiments clearly show a significant improvement of our method over current state-of-the-art asynchronous algorithms. This improvement is the results of the following steps:

- We identified key variables that affect the performance of the optimization procedure and depend on the queuing dynamics.
- Building on the findings of our analysis, we introduced a new algorithm called `Generalized AsyncSGD`. This algorithm exploits non-uniform agent selection and offers two notable advantages: First, it guarantees unbiased gradient updates, and second, it improves convergence bounds.
- To gain deeper insights, we delved into the limit regimes characterized by large concurrency. In these contexts, our analysis showed that heterogeneity in server speeds can be balanced by the strategic use of non-uniform sampling among agents.
- Experimental results finally showed that our approach outperforms other asynchronous baselines on deep learning experiments.

Chapter 7 is based on the work Leconte, Jonckheere, et al. 2024.

Chapter 1

Introduction

In this chapter, we lay the foundation for the subsequent chapters by introducing and contextualizing the primary concepts and ideas that will be explored throughout the document. This chapter serves as an overview of the critical areas that are essential for understanding the contents of this thesis. We begin by providing a historical overview of statistical learning, machine learning, and its mathematical formulation. We then move on to the quantized world, which forms the main constraint basis of this thesis. Following that, we discuss the case of federated learning, and specifically the *efficient* setting, which takes into account bandwidth and/or device computational power limitations.

1.1 Statistical Learning

1.1.1 A brief historical introduction

The field of statistical learning focuses on developing and analyzing methods for making predictions or decisions based on data. Its roots can be traced back to the 19^{th} century, with early pioneers such as Legendre, Gauss, and Laplace introducing independent methods for regression (Legendre 1806; Gauss 1809), and conditional probability (Laplace 1814). In the mid-20^{th} century, the field experienced a surge of interest in machine learning and artificial intelligence, leading to the development of methods such as the perceptron algorithm (McCulloch and Pitts 1943) and decision trees (Hunt, Marin, and Stone 1966). However, progress in statistical learning was hindered by limitations in computing power and availability of large datasets in the 1970s and 1980s. The 1990s saw a resurgence of interest, with the development of methods that could handle large datasets and complex models, such as support vector machines (Cortes and Vapnik 1995), boosting (Freund and Schapire 1996), random forests (Breiman 2001), and neural networks (LeCun 1985; Schmidhuber 1989; LeCun, Bottou, et al. 1998). The growth of the internet and availability of massive amounts of data contributed to the field's expansion, leading to a dynamic and ever-evolving field of statistical learning. Today, statistical learning is applied in various domains, including climate studies, finance, healthcare, robotics, social media, among others. Researchers continue to develop methods for analyzing data and automatizing human tasks, including deep learning (LeCun, Bengio, and G. Hinton 2015), reinforcement learning (Sutton and Barto 2018), distributed learning (Konečný et al. 2016; B. McMahan et al. 2017), and generative AI (A. Vaswani et al. 2017).

This thesis focuses on supervised learning, which involves predicting an output $y \in \mathcal{Y}$ based on a new input $x \in \mathcal{X}$, with $\mathcal{X}$ and $\mathcal{Y}$ being measurable spaces. The main tasks of supervised learning are regression, which predicts a quantitative outcome, and classification, which predicts a categorical outcome. The goal of supervised machine learning is to find a predictor h, a measurable function, that predicts an output $y \in \mathcal{Y}$ for any new input $x \in \mathcal{X}$.

1.1.2 Loss function

To measure the quality of a predictor, we select a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ that is measurable and intuitively, for any $(y, y') \in \mathcal{Y}^2$, $\ell(y, y')$ is small if y and y' are similar, and large otherwise. For regression tasks, the squared loss $\ell(y, y') = \frac{1}{2}(y - y')^2$ is typically used, whereas for classification, the logistic loss $\ell(y, y') = \log(1 + \exp(-yy'))$ (for logistic regression) or the hinge loss $\ell(y, y') = \max\{0, 1 - yy'\}$ are employed. We define the risk f of a predictor h as the expected loss under the distribution D^{data} of the observations:

$$f(\mathrm{h}) := \mathbb{E}_{(x,y)\sim D^{\text{data}}}[\ell(\mathrm{h}(x), y)]. \tag{1.1}$$

The learning process aims to find the best predictor h_* that minimizes the risk f. In most cases, the quality of a predictor h is not measured based on its loss, but rather on the excess risk $f(\mathrm{h}) - f(\mathrm{h}_*)$, which measures how far h is from h_* based on the chosen loss ℓ and the distribution D^{data}. To approximate h_*, the learning process involves selecting a parametric family $\mathcal{H}$ of predictors and minimizing the risk over it, resulting in $h_{\mathcal{H}} = \arg\min_{h \in \mathcal{H}} f(h)$.

Empirical Risk Minimization (ERM) is a technique used in machine learning when the true distribution of observations D^{data} is unknown and we only have access to a dataset $\mathcal{D}$ of cardinal

$|\mathcal{D}|$, composed of several pairs $(x_k, y_k)_{k\in\{1,\dots,|\mathcal{D}|\}} \in (\mathcal{X}\times\mathcal{Y})^{|\mathcal{D}|}$. The empirical risk error is defined as

$$f_{\mathcal{D}}(\mathrm{h}) := \frac{1}{|\mathcal{D}|}\sum_{k=1}^{|\mathcal{D}|} \ell(\mathrm{h}(x_k), y_k). \tag{1.2}$$

One approach is to minimize it instead of the true risk f:

$$\min_{\mathrm{h}\in\mathcal{H}} f_{\mathcal{D}}(\mathrm{h}). \tag{1.3}$$

However, this approach has a major pitfall: over-fitting. It occurs when the empirical risk error is very low, but the excess risk is large. In this case, the true risk is also called the generalization error as it measures how accurately the predictor $\mathrm{h}_{\mathcal{D}}$ trained on a dataset $\mathcal{D}$ is able to predict output values for unseen data. In practice, the learning process is the same as if one could access the true distribution of observations D^{data}: from a predefined family of predictors $\mathcal{H}$, one seek to find $\mathrm{h}_{\mathcal{D}}$ that minimizes the empirical risk error $f_{\mathcal{D}}$ (instead of f). But this additional approximation induces another term in the excess risk:

$$f(\mathrm{h}_{\mathcal{D}}) - f(\mathrm{h}_*) = \underbrace{f(\mathrm{h}_{\mathcal{D}}) - f(\mathrm{h}_{\mathcal{H}})}_{\text{estimation error}} + \underbrace{f(\mathrm{h}_{\mathcal{H}}) - f(\mathrm{h}_*)}_{\text{approximation error}}. \tag{1.4}$$

The first term represents the estimation error, which measures how well the true risk $f(\mathrm{h}_*)$ is estimated by the empirical risk $f_{\mathcal{D}}$. The second term represents the approximation error, which measures how well the family $\mathcal{H}$ approximates the set of all possible predictors.

1.2 Machine Learning

The goal of machine learning is to find a solution (from a given family of predictors) to the empirical risk minimization problem in a suitable amount of time. When the chosen family of predictors $\mathcal{H}$ is small (and finite), one can solve the empirical risk minimization problem with a brute-force method: testing all possible predictors, and selecting the one that yields the smallest error. We give an example of such "simple" family in Section 2.4.1 and Figure A.3. But small families of functions have high approximation errors (bias), and results in poor performances on complex tasks. As a consequence, current state of the art family of predictors $\mathcal{H}$ are very large (often infinite), and as such we rely on the Gradient Descent (GD) method to solve the empirical risk minimization problem.

1.2.1 Gradient descent

A common practice is to parameterized the family of predictors $\mathcal{H}$ with some parameters $w \in \mathbb{R}^d$ for a chosen dimension d: $\mathrm{h} := \mathrm{h}(w)$. Hence, the goal of optimization (Nesterov 2013) is to find an optimal point (not necessarily unique) that minimizes the (true or empirical) risk f:

$$w_* := \arg\min_{w\in\mathbb{R}^d} f(\mathrm{h}(w)) \tag{1.5}$$

With a slight abuse of notation, in the subsequent chapters, we will note $f(w) = f(\mathrm{h}(w))$ for any parameters $w \in \mathbb{R}^d$.

Gradient descent (GD) is a widely used optimization algorithm for training machine learning

models. It is an iterative optimization algorithm that updates the model parameters by taking small steps in the direction of the negative gradient of the loss function with respect to the model parameters. The key advantage of GD over other optimization algorithms is its ability to handle large datasets and high-dimensional models. Solving the risk minimization problem with an accuracy $\epsilon > 0$ means finding an approximate solution w_T after $T \in \mathbb{N}$ iterations, such that the error $f(w_T) - f(w_*) < \epsilon$.

Gradient descent was first introduced by Cauchy 1847, and aims at finding an optimal point w_* minimizing the (empirical) risk, by employing (first-order) regularity information about the function $w \to f(w)$. The gradient descent algorithm updates the model parameters by taking steps in the direction of the negative gradient of the loss function f. The update rule for gradient descent is:

$$w_{new} = w_{old} - \eta * \nabla f(w_{old}), \tag{1.6}$$

where w_{new} is the new value of the model (h(w)) parameters, w_{old} is the old value of the model parameters, η is the learning rate (or step size), and $\nabla f(w_{old})$ is the gradient of the loss function with respect to the model parameters. The step size controls the update's magnitude. The choice of the step size rate is fundamental and has been one of the most studied questions: taking η "too small" slows down convergence, and η "too big" yields divergence.

The choice for η, and all related convergence guarantees are tightly related to the geometry of the landscape of the function $w \to f(w)$. A non-regular function may have discontinuities, singularities, or oscillations that can cause optimization algorithms to get stuck or to converge slowly. This is why, regularity is a desirable property for objective functions in optimization. A standard, yet limited, regularity assumption is convexity:

A1. *Convexity (in $\mathbb{R}$). For all $r \in [0,1]$, and for all $w, v \in \mathbb{R} \times \mathbb{R}$:*

$$f(rw + (1-r)v) \leq rf(w) + (1-r)f(v). \tag{1.7}$$

If in addition f is also differentiable, we have the following useful equivalent characterization of convexity:

A2. *Convexity (+ diff. in $\mathbb{R}$). For all $w, v \in \mathbb{R} \times \mathbb{R}$:*

$$f(v) \geq f(w) + (v-w)^\top f(w). \tag{1.8}$$

These property is of prime importance to study standard machine learning algorithms such as the Least Squares Regression (LSR) problem. Every first-order stationary points are (without uniqueness) minima for convex functions. Strong convexity assumption can additionally ensure that the minimum is unique. However the models we consider in this thesis are highly non-convex. Therefore we will mainly focus on finding first-order stationary points, under smoothness assumptions:

A3. *Smooth gradients (in $\mathbb{R}$). The gradient ∇f of a function f is L-Lipschitz continuous for some $L > 0$, i.e. for all $w, v \in \mathbb{R}$:*

$$\|\nabla f(w) - \nabla f(v)\| \leq L\|w - v\|. \tag{1.9}$$

In particular we will use, and study a stochastic (yet unbiased) version of the gradient descent method: stochastic gradient descent (SGD). It is particularly useful when the size of the dataset we consider for the empirical risk $f_{\mathcal{D}}$ is large. The iteration procedure is close to the one of GD: the model parameters takes small steps in the direction of "an unbiased approximation" of the negative gradient of the loss function with respect to the model parameters. The update rule for

SGD (Robbins and Monro 1951) is:

$$w_{new} = w_{old} - \eta * \nabla f(w_{old}, \zeta), \tag{1.10}$$

where w_{new} is the new value of the model parameters, w_{old} is the old value of the model parameters, η is the learning rate, and $\nabla f(w_{old}, \zeta)$ is an approximation of the gradient of the loss function with respect to the model parameters. This approximation step complexifies the method and its analysis (F. Bach and Moulines 2013). In the following chapters, we will assume that the variance of the stochastic gradients $\nabla f(w, \zeta)$ is bounded:

A4. *Bounded noise (in $\mathbb{R}$). There exists some $\sigma^2 > 0$, such that for all $w \in \mathbb{R}$:*

$$\mathbb{E}[\|\nabla f(w, \zeta) - \nabla f(w)\|^2] \leq \sigma^2. \tag{1.11}$$

There are many variants of SGD that have been developed to address the limitations of the standard SGD algorithm. Some of these variants include mini-batch gradient descent, stochastic gradient descent with momentum, stochastic gradient descent with weight decay, and stochastic gradient descent with adaptive learning rate methods.

1.2.2 Deep Learning

Deep Neural Networks (DNNs) are complex machine learning models that use non-linear functions with a large number of parameters to process high-dimensional data and achieve state-of-the-art performance. In deep learning methods, DNNs are often viewed as a composition of functions. From the perspective of efficient inference, a DNN can be defined as a set of computational blocks that form a directed graph where each node represents a mathematical operation and each vertex corresponds to an actual composition. Among the most common neural network architectures, we can classify the following nodes: convolutions, normalization layers, activation layers, and merge layers. In this manuscript, we will focus on the optimization and/or the quantization of convolutions and activations layers: where the computational cost is the most important.

Convolutions also encompass fully-connected layers, which take as inputs a tensor and perform an operation equivalent to a matrix (weights) and vector (inputs) multiplication. Normalization layers, such as batch-normalization (Ioffe and Szegedy 2015) and layer normalization (J. L. Ba, Kiros, and G. E. Hinton 2016), were introduced to improve the stability of training. The defined operation is a sequence of affine transformations of the features that are approximations of the expectation and standard deviation of the features. Activation layers add non-linear transformations to the graph called activation functions, such as the sigmoid (Menon et al. 1996), softmax (Banerjee et al. 2020), ReLU (Agarap 2018), and GELU (A. Nguyen et al. 2021). These functions enable DNNs to approximate any continuous function on a compact set to any given precision (Cybenko 1989). Merge layers are the final set of nodes that we consider and are extensions of skip connections (He et al. 2016). When several intermediate outputs are considered, they should be merged using either concatenation (Ronneberger, Fischer, and Brox 2015), addition (Melekhov et al. 2017), or multiplication (A. Vaswani et al. 2017).

More formally, we will consider the training of an artificial neural network model: for an input $x \in \mathcal{X}$, the predictor h is denoted and parameterized as $\mathrm{h}(x) := \mathrm{nn}(x, w)$. The model is parameterized by w (the so-called weights and biases), and predicts an output $y \in \mathcal{Y}$ for each input $x \in \mathcal{X}$. In this work, we will focus on supervised federated learning, although all ideas explored here can be easily extended to unsupervised or semi-supervised settings.

1.3 Quantization

1.3.1 Compression and Quantization

Compression refers to the process of reducing the size of a signal without significantly losing its information content. This is achieved by identifying and removing redundancies in the signal. Compression is used in a variety of applications, including digital audio and video compression, image compression, and data compression.

In signal processing, compression is used to reduce the amount of data that needs to be transmitted or stored. This can be achieved through techniques such as wavelet transforms, Fourier transforms, and other signal processing techniques (Shannon 1948). In wireless communications, compression is used to reduce the amount of data that needs to be transmitted over a wireless channel. This can help to increase the efficiency of the communication system and reduce the amount of time required for data transmission. By reducing the size of a signal without losing its information content, compression can help to improve the performance of communication systems and enable the development of new technologies. Depending on the input to be compressed, a vast amount of methods have been developed including matrix factorization (e.g. low rank decomposition), sparsification, spectral analyses, or quantization. In this work, we will focus on the latter.

Quantization is the process of discretizing continuous variables into discrete ones. It is a fundamental technique in physics and mathematics that has played a crucial role in the development of quantum mechanics. Quantization is also a fundamental process in signal processing and wireless communications. It involves reducing the precision of a signal representation to a more efficient and compact form. The goal of quantization is to maintain the essential information of the signal while minimizing the number of bits required to represent it. There are several types of quantization, including scalar quantization, vector quantization, deterministic quantization, and stochastic quantization.

Deterministic quantization is a quantization technique where the quantization function is defined by a deterministic rule. This means that the same quantization function is used for all input signals. Deterministic quantization is simple and efficient, but it may not be optimal in terms of the number of bits required to represent the signal. Stochastic quantization is a more advanced form of quantization that uses probability distributions to determine the quantization function.

Scalar quantization is the simplest form of quantization, where a continuous-time signal is divided into discrete-time samples and each sample is quantized to a finite number of bits. This process is used in many practical applications, such as digital audio and video compression. A random scalar quantizer is a random map from the real line to a (scalar) codebook $\mathscr{O}_Q = \{o_1, \ldots, o_M\} \subset \mathbb{R}$ where $M \geq 2$. It is assumed that $-\infty < o_1 < \cdots < o_M < \infty$. The resolution (or code rate) is $P = \log_2(M)$ is the number of bits needed to uniquely specify a codeword. A scalar quantizer is said to be *uniform* if for all $i \in [M-1]$, $o_{i+1} - o_i = \delta$, for some $\delta > 0$. For $x \in \mathbb{R}$ and $u \in [0,1]$, consider a function $\mathrm{SQ}(x, \mathscr{O}_Q, u) \in \mathscr{O}_Q$. If $U \sim \mathrm{Unif}([0,1])$, then $\mathrm{SQ}(x, \mathscr{O}_Q, U)$ is a random scalar quantizer. A random scalar quantizer is said to be *unbiased* if for all $x \in [o_1, \ldots, o_M]$, $\mathbb{E}_{U \sim \mathrm{Unif}([0,1])}[\mathrm{SQ}(x, \mathscr{O}_Q, U)] = x$. A simple way to construct an unbiased scalar quantizer goes a follows. We first compute the index $j(x) \in [M]$ such that $x \in \left[o_{j(x)}, o_{j(x)+1}\right)$. Note that $x = \lambda^*_{j(x)}(x) o_{j(x)} + (1 - \lambda^*_{j(x)}(x)) o_{j(x)+1}$ where

$$\lambda^*_{j(x)}(x) = (x - o_{j(x)}) / (o_{j(x)+1} - o_{j(x)}) \in (0,1] .$$

For $u \in (0,1]$, we set

$$\mathrm{SQ}(x, \mathscr{O}_Q, u) = \mathbb{1}_{\{u \le \lambda^*_{j(x)}(x)\}} o_{j(x)} + \mathbb{1}_{\{u > \lambda^*_{j(x)}(x)\}} o_{j(x)+1}\,.$$

Since $\mathbb{E}_{U\sim\mathrm{Unif}([0,1])}(U \le \lambda_j(x)^*(x)) = \lambda^*_{j(x)}$ the unbiasedness follows. It is easily seen that the distortion of a scalar quantizer is directly related to the diameter of the quantizer:

Proposition 1. *For all $x \in [o_1, o_M]$, it holds that*

$$\mathbb{E}_{U\sim\mathrm{Unif}([0,1])}[\{\mathrm{SQ}(x, \mathscr{O}_Q, U) - x\}^2] \le (1/4) \sup_{i\in[Q-1]} \{o_{i+1} - o_i\}^2\,.$$

If the scalar quantizer is uniform,

$$\mathbb{E}_{U\sim\mathrm{Unif}([0,1])}[\{\mathrm{SQ}(x, \mathscr{O}_Q, U) - x\}^2] \le (1/4)(Q-1)^{-2}\{o_Q - o_1\}^2\,.$$

Proof of this proposition, and more details about scalar quantization can be found in Section 5.3 and Appendix B.3.1.

Vector quantization is a more advanced form of quantization that is used to represent signals in higher dimensions. It involves dividing the signal into a set of vectors and quantizing each vector to a finite number of bits. Vector quantization is used in many applications, such as wireless communications and image compression. The previously defined scalar quantizer can be applied element wise to a given vector $x \in \mathbb{R}^d$ to build a basic vector quantizer:

Definition 2 (s-quantization operator)**.** *Let $s \ge 1$ and $p \ge 1$. Given $x \in \mathbb{R}^d$, the s-quantization operator $\mathcal{C}_s$ is defined by:*

$$\mathcal{C}_s(x) := \|x\|_p \times \sum_{i=1}^{d} \mathrm{sign}(x_i)\Big\{s^{-1}\lfloor s|x_j|/\|x\|_p \rfloor + \mathbb{1}_{\{U_i \le s|x_j|/\|x\|_p - \lfloor s|x_j|/\|x\|_p \rfloor\}}\Big\} e_i\,. \tag{1.12}$$

where $\{U_i\}_{i=1}^d$ are d-independent uniform random variables on $[0,1]$.

A large body of literature has covered and extended the topic of vector quantization with several strategies like gain-shape (Gersho and Gray 2012), or product quantization (Matsui et al. 2018; Stock et al. 2020). We present here the (stochastic) Delaunay quantization (also known as dual quantization). The principle of Delaunay quantization is to map an $\mathbb{R}^d$-valued vector x onto a codebook $\mathscr{C}_M$ using a random splitting operator Dual-VQ$(x, \mathscr{C}_M, U)$ such that, for all $x \in \mathrm{ConvHull}(\mathscr{C}_M)$,

$$\mathbb{E}_{U\sim\mathrm{Unif}([0,1])}[\text{Dual-VQ}(x, \mathscr{C}_M, U)] = x\,. \tag{1.13}$$

In practice, a dual quantizer procedure amounts to define a probability distribution of $\mathscr{C}_M$, with weights $(\lambda_1^*(x), \dots, \lambda_M^*(x))$, $\lambda_i^*(x) \ge 0$, $\sum_{j=1}^M \lambda_j^*(x) = 1$. Set $\Lambda_0^*(x) = 0$ and for $i \in [M]$, $\Lambda_i^*(x) = \sum_{j=1}^i \lambda_j^*(x)$. Note that $\Lambda_M^*(x) = 1$. If $u \in \left(\Lambda_{j-1}^*(x), \Lambda_j^*(x)\right]$, $j \in [M]$, we set Dual-VQ$(x, \mathscr{C}_M, u) = c_j$. In such that, for all $x \in \mathrm{ConvHull}(\mathscr{C}_M)$, we get

$$\mathbb{E}_{U\sim\mathrm{Unif}([0,1])}[\text{Dual-VQ}(x, \mathscr{C}_M, U)] = \sum_{i=1}^{M} \lambda_i^*(x) c_i = x\,.$$

More details will be given in Appendix B.3.2.

1.3.2 QNN

Deep neural networks (presented in Section 1.2.2) are currently used in many IoT devices with limited computational and memory resources. This has led to a growing research area focused on reducing the size and inference time of DNNs while maintaining accuracy. The methods used in this area include model pruning, neural architecture search, efficient architecture design, and low-rank decomposition. In this work, we focus on network quantization, where weights and/or activations are quantized to lower bit widths, allowing for efficient fixed-point inference and reduced memory bandwidth usage. Network quantization has been studied extensively, with examples such as Courbariaux, Bengio, and David 2015; Jacob et al. 2018; Darabi et al. 2018; Choukroun et al. 2019; Lei Deng et al. 2020; Qin et al. 2020; Bhalgat et al. 2020; Chmiel et al. 2021. These works demonstrate the effectiveness of network quantization in reducing the size and inference time of DNNs while maintaining accuracy.

Quantized neural networks (QNNs) have attracted many research efforts. The task of learning a quantized neural network (QNN) can be formulated as minimizing the training loss with quantization constraints on the weights, i.e.,

$$\min_{w \in \mathcal{Q}} f(w),\ f(w) = \mathbb{E}_{(x,y)\sim D^{\text{data}}}[\ell(\text{nn}(x,w),y)]. \tag{1.14}$$

The set of quantization levels, $\mathcal{Q}$, is a subset of the set of all possible values for the DNN's parameters. The objective of this optimization problem is to minimize the training loss ℓ, which can be a cross-entropy or square loss function, subject to certain constraints that ensure the quantization process is feasible. The DNN's prediction function, $\text{nn}(x,w)$, takes in an input x and a set of parameters w and produces an output. The training distribution, D^{data}, is used to evaluate the performance of the DNN and to determine the optimal parameters. The optimization problem in this program is extremely challenging due to several factors. Firstly, the underlying function being optimized is non-convex and non-differentiable, making it difficult to find a globally optimal solution. Secondly, the optimization problem is combinatorial in nature, meaning that it involves selecting a subset of the quantization levels from a larger set. Finally, the optimization problem is also constrained by the need to ensure that the quantization process is feasible, which adds an additional layer of complexity to the problem. Given these challenges, it is important to develop algorithms that can produce a manageable and accurate solution to this optimization problem. While finding a globally optimal solution may not always be possible, it is possible to develop approximation algorithms that can find a solution that is close to the optimal solution while requiring a manageable amount of computational effort. One of the advantages of QNNs is their computational efficiency. This is achieved by using a lower bit-width for weight values and intermediate features, which reduces their memory footprint and the number of individual bit operations required. While floating-point operations may be better supported by some hardware (e.g. CPUs), fixed-point operations are more suitable for efficiency on specific devices, such as GPUs, which can leverage them more effectively (Hettiarachchi, Davuluru, and Balster 2020).

Floating point vs fixed point In the deep learning community, the default scalar value representation is typically floating point on 32 bits. A floating point value is defined by three sets of bits: one first bit to encode the sign, m bits for the significant or mantissa, and e bits for the exponent. The size of the mantissa and exponent determines the trade-off between the level of precision and the amount of memory used. The IEEE 754 format uses $m = 8$ bits for the mantissa and $e = 23$ bits for the exponent, totaling 32 bits. Other formats, such as half format, have been introduced to increase speed and decrease memory usage. However, some operations, such as

loss computation and gradients, are still performed on 32 bits. In computer science, fixed point representation defines a real value as a first integer multiplied by an implicit scaling factor. However, in the quantization community, fixed point representation is assumed to be an integer representation, often with uniform quantization (S. Zhou et al. 2016). In practice, most weight values follow a mono-modal, almost symmetric distribution that can be modeled by a Gaussian. Non-uniform quantizers have been proposed in Sun, N. Wang, C.-Y. Chen, Ni, Agrawal, Cui, Venkataramani, El Maghraoui, et al. 2020; Chmiel et al. 2021 to provide better coverage of the entire span of values. The fixed point representation must use a specific scale suited for each tensor, which can be learned or defined layerwise. Quantization is currently difficult to work with, especially in current deep learning frameworks. In practice, the compression community simulates the quantization process to leverage existing optimization libraries such as PyTorch (Paszke et al. 2019), TensorFlow (Martín Abadi et al. 2015), or Jax (Bradbury et al. 2018).

Simulated Quantization In order to simulate the quantization process using only the best supported formats (e.g. float32 and float16), the quantization (Q()) and de-quantization ($Q^{-1}()$) processes are performed before the tensor products. In practice, the standard dot-product operation $w \cdot x$, for any weight w and any input x is replaced by $Q^{-1}(Q(w)) \cdot Q^{-1}(Q(x))$ in the code. The resulting quantized network is characterized by zero derivatives almost everywhere, which makes it challenging to perform stochastic gradient descent optimization. To address this limitation, various methods have been proposed, with post-training quantization being a popular approach, especially for large language models (Frantar et al. 2022; J. Lin et al. 2023; J. Kim et al. 2023; H. Guo et al. 2023). Quantization-aware training (S. Gupta et al. 2015; D. Zhang et al. 2018; Jin et al. 2021; Yamamoto 2021; C.-W. Huang, T.-W. Chen, and J.-D. Huang 2021; Umuroglu et al. 2017) and quantized training (J. Chen et al. 2020; Sun, N. Wang, C.-Y. Chen, Ni, Agrawal, Cui, Venkataramani, Maghraoui, et al. 2020; Yukuan Yang et al. 2022; Chmiel et al. 2021) are also commonly used techniques to obtain better quantized models. The majority of these methods employ two main solutions to tackle the zero derivative challenge. One approach is to simply omit the rounding step in the backward pass (straight-through estimation (Courbariaux, Bengio, and David 2015) also known as STE), while the other is to use a soft function to approximate the rounding step (Ajanthan, K. Gupta, et al. 2021).

A brief state of the art A special case of quantization-aware training is binarized neural networks (BNNs) which were first proposed by Courbariaux, Bengio, and David 2015; Courbariaux, Hubara, et al. 2016 and have been followed by a huge amount of subsequent contributions (Gholami et al. 2021; W. Zhao et al. 2020; Y. Guo 2018; Nagel et al. 2021). This design usually binarizes weights and activations to obtain principal forward computation blocks in binary. It learns binarized weights via full-precision latent ones, which are updated by the classical gradient descent backpropagation. Many concurrent approaches (Bai, Y.-X. Wang, and Liberty 2018; Ajanthan, Dokania, et al. 2019; Ajanthan, K. Gupta, et al. 2021; M. Lin et al. 2020; Leconte, Schechtman, and Moulines 2023) formulated the BNN learning task as a constrained optimization problem and discussed different methods to generate binary weights from real-valued latent ones.

Remark 3. *Note that the first deep neural network considered in LeCun 1985 is an artificial neural network with weights $w \in \{-1, +1\}$. We will develop and give more details about BNNs in Chapter 3.*

In the context of Quantized Neural Networks (QNN), the choice of the quantizer and the normalization of the weights play a crucial role. Several studies have focused on the development of non-uniform or distribution-dependent quantizers, such as Banner et al. 2018; Hou and Kwok 2018; Bhalgat et al. 2020; Liang et al. 2021; Fournarakis and Nagel 2021; A. Zhou et al. 2017; Y.

Zhou et al. 2018. While statistical quantizers are often more efficient, they are more complex to implement and require fine-tuning (Zhaoyang Zhang et al. 2021). Several works have formulated the quantization problem as an optimization problem (H. Li et al. 2017; F. Li, B. Zhang, and B. Liu 2016; C. Zhu et al. 2016; Carreira-Perpinán and Idelbayev 2017; Leng et al. 2018; Polino, Pascanu, and Alistarh 2018). However, these methods rely on certain assumptions that may not hold for deep neural networks (Y. Guo 2018). In Moons et al. 2017; T.-J. Yang, Y.-H. Chen, and Sze 2017; Esser et al. 2015, the QNN training is tackled as an energy efficiency problem, whereas Gong et al. 2019 propose a Differentiable Soft Quantization (DSQ) to efficiently train QNN. For BNN, M. Kim and Smaragdis 2016; Hubara et al. 2016; Rastegari et al. 2016 proposed to use sign$(\cdot)$ function for quantizing activation functions, but this approach significantly affects the performance. More complex quantization schemes have been considered in Choi et al. 2018 alleviating performance degradation. Hybrid formats FP8 (N. Wang et al. 2018) or INT8 (Wiedemann et al. 2020; Banner et al. 2018) were successfully employed to achieve a low precision training. Recent works have proposed to jointly optimize the quantization parameters (of weights and activations) and the weights parameters. This task can be done by modifying the learning loss or by minimizing the quantization error (C. Zhu et al. 2016; D. Zhang et al. 2018; Y. Li, Dong, and W. Wang 2019). More details will be given in Chapter 2 and Chapter 3.

1.4 Federated Learning

Federated learning is a distributed machine learning technique that allows n devices to collaboratively train a model without sharing their data. This approach has gained popularity due to its potential to address privacy concerns and the limitations of centralized data storage and processing.

We assume that each client i has a local model w_i, can sample from a local dataset, and seeks to optimize a local loss function $f_i(w_i)$:

$$f_i(w_i) = \mathbb{E}_{(x,y)\sim D_i^{\text{data}}}[\ell(\text{nn}(x, w_i), y)]. \tag{1.15}$$

Without loss of generality, the goal of FL is to produce a model that optimizes the average of loss functions (this average can be weighted based on the size of local datasets and/or based on user preferences).

1.4.1 Synchronous Federated Learning

There are several challenges associated with federated learning, including but not limited to communication overhead, model convergence, and computational imbalance. Communication overhead is related to the amount of data that needs to be transmitted between devices during the training process. Modern models (e.g. LLMs) have hundreds of billions of parameters, and a direct transmission would not fit the bandwidth. Model convergence is the challenge of ensuring that the global model accurately represents the local models. The implicit assumption of FL systems is that a server can benefit from learning from multiple sources of data. Therefore, it is assumed that the data present in each device, implicitly encoded in the loss function f_i, can be used to train a model CS that is collectively useful. Naturally, if the loss functions f_i are completely unrelated to each other (i.e. there is data heterogeneity), then there is no point in learning a shared model. Computational imbalance occurs when devices does not share the same computational power and/or when some devices have more data than others. These differences can lead to unequal representation in the global model and/or to an increase of the time spent between two consecutive CS updates.

Remark 4. *Federated learning (FL) is an approach to machine learning that utilizes data from multiple clients in a manner that protects their privacy. To this end, ensuring provable privacy guarantees for FL algorithms is of utmost importance. Modern FL algorithms operate by the transmission of gradients between the server and clients (B. McMahan et al. 2017; J. Nguyen et al. 2022). A recent study (Geiping et al. 2020) has shown how to invert gradients and reconstruct images from Convolutional Neural Networks (CNNs) gradients. In order to protect privacy, FL protocols employ additional security measures to prevent the recovery of sensitive data from gradients. Two main methods have been developed to tackle this issue: Secure Aggregation (Keith Bonawitz, Ivanov, et al. 2016) and Differential Privacy (Kairouz, B. McMahan, et al. 2021). These techniques provide mathematical guarantees of privacy. Although our study does not make a contribution to the field of FL privacy, all the methods we examine in this work are compatible with privacy-preserving techniques.*

Federated Averaging (FedAvg) is the most widely used algorithm in the field of Federated Learning (B. McMahan et al. 2017). FedAvg has two significant contributions to the FL field: client sub-sampling and the importance of local steps. At each step t of training, the central server selects a subset $\mathcal{S}_t$ of s clients for a training task. The population of devices may contain hundreds of millions of devices. However, in B. McMahan et al. 2017, the authors observed that choosing around $s = 100$ clients at every training step produces results comparable to choosing all n clients all the time. FedAvg shows that we can gain communication efficiency and reduce the number of training rounds if each client performs multiple local steps of Stochastic Gradient Descent (SGD). Specifically, each client will receive a model weight from the server and update it K times (perform K local steps):

$$\begin{cases} w_{t,0} = w_t \\ w_{t,q} = w_{t,q-1} - \eta \widetilde{\nabla} f(w_{t,q-1}) \quad \forall q \in [1, K], \end{cases} \tag{1.16}$$

and then transmits the total update $w_{t,K} - w_{t,0}$ to the server. More details can be found in Algorithm 1. The update $z^i_{t,K} = w_t - w^i_{t,K}$ is no longer a gradient. However, the intuition behind FedAvg suggests that $z^i_{t,K}$ points in a direction that will reduce the loss f_i. To incorporate information from higher-order derivatives of f_i, multiple local steps are taken. For this reason, $z^i_{t,K}$ is called a pseudo-gradient. Several papers have validated the empirical improvement obtained by using multiple local steps (Karimireddy, Kale, et al. 2020; B. McMahan et al. 2017; J. Nguyen et al. 2022). Moreover, some works have shown theoretically that adding local steps can reduce the number of communication rounds (Karimireddy, Kale, et al. 2020; Woodworth et al. 2020). However, the same work also demonstrates settings in which local steps actually harm performance (Woodworth et al. 2020). The problem of client-drift arises when clients perform local steps. Each client optimizes its own loss function f_i, which may differ significantly from the global loss f. The problem can be illustrated by an example where each client performs a very large number K of local steps. In this case, for adequate learning rates, the individual models would converge locally to the minimizers w^i_*. However, the landscapes of the loss functions (f_i) can be non-convex, and the average of optima $(\frac{1}{s}\sum_i w^i_*)$ is not necessarily the optimum of the average. In fact, such average of optima may correspond to a weight vector w that results in a very bad value for the loss function f. To overcome this issue, the learning rates η that allow FedAvg to converge to a global optimum must be tuned together with the number of local steps K. The theoretical analysis in several works (Karimireddy, Kale, et al. 2020; J. Nguyen et al. 2022) requires that the learning rate η decreases on the same speed as $\frac{1}{K}$.

Algorithme 1 : FedAvg over T iterations.

Input : Number of steps T, LR η, Selection Size s, Maximum local steps K ;

/* At the Central Server */

Initialize

  Initialize parameters w_0;

end

for $t = 1, \dots, T$ **do**

  Generate set $\mathcal{S}_t$ of s clients ;

  for *all clients* $i \in \mathcal{S}_t$ **do**

    Server sends w_t to client i;

    Server receives update $z^i_{t,K}$ from client i;

  end

  Update central server model $w_t \leftarrow w_{t-1} - \frac{\eta}{|\mathcal{S}_t|} \sum_{i \in \mathcal{S}_t} z^i_{t,K}$;

end

/* At Client i */

Initialize

  Client receives w_t and K from the Server;

  Set $w^i_{t,0} = w_t; q = 0$;

end

for $q = 1, \dots, K$ **do**

  Compute local stochastic gradient $\widetilde{g}^i$ at $w^i_{t,q-1}$;

  Update local model $w^i_{t,q} \leftarrow w^i_{t,q} - \eta \widetilde{g}^i$;

end

Send $z^i_{t,K} := w_t - w^i_{t,K}$ to the server;

1.4.2 Asynchronous Federated Learning

A significant advantage of FL is the substantial number of devices collaborating to train a shared model. As such, the development of scalable FL algorithms is a critical open issue. When the number of devices increases, several problems arise, including synchronization issues and device heterogeneity. In Section 1.4.1, client updates are synchronized, meaning that all clients commence their local iterations on the same model and collectively contribute to the update of the central server (CS). However, it is crucial to ensure that the FL training process does not negatively impact the users' experience when utilizing their devices. Asynchronous FL is necessary to address device availability and heterogeneity concerns, allowing for a more seamless and efficient training process.

Federated learning (FL) algorithms require significant computational power and memory, and the transmission of parameters between clients and server necessitates network stability and a substantial amount of data exchange. To address these issues, the FL protocol must adhere to certain requirements when conducting training on a client device. The device must be idle to avoid negatively impacting the user experience, connected to a power source due to the energy consumption of training, and connected to an unmetered network. However, any of these conditions may change while the client is performing a training task, which could cause the task to be abandoned. Consequently, it is imperative to develop fault-tolerant FL protocols that can account for potential changes in these conditions. In large-scale applications of FL, devices with varying computing capacities will participate in the training process, and faster

clients will finish their training tasks in a shorter time-frame than slower clients. In synchronous settings, the time taken for each training round will be determined by the slowest client, leading to waiting times that are undesirable. This waiting can slow training by a factor of 10: also know as the *straggler* problem in the FL literature. One common solution to this issue is over-sampling, where the server selects a larger number of clients for training and subsequently drops any clients that take too long to respond. However, this approach has a drawback in that slow clients will be less relevant in the training process. To address this issue, an asynchronous FL protocol was proposed in Xie, Koyejo, and I. Gupta 2019 that does not require the waiting times of synchronous protocols.

In Section 1.4.1, we discussed the effects of data heterogeneity on the learning process. However, a significant challenge arises when data and device heterogeneity occur simultaneously. Devices that are more frequently available or update quickly will contribute more to the training process. The capabilities of a device can be complexly correlated with the data present in the device. For instance, network reliability may vary in different geographical regions, which can affect the availability of data. This correlation between data availability and device capability can lead to bias in the trained model towards clients who contributed more to the training process. A study by Huba et al. 2022 demonstrated that the shared model performs better for clients who contributed more to training. However, in fairness-critical applications, it is unacceptable for the trained model to favor one subset of clients over another. Therefore, it is essential to develop FL algorithms that consider fairness and are resilient to non-iid data and device heterogeneity.

Chapter 2

An annealing process to obtain QNN

We present in this chapter our first contribution, which deals with training artificial neural networks with weights represented with few bits. This led to the following publication (Leconte, Schechtman, and Moulines 2023) in an international conference:

Leconte Louis, Sholom Schechtman, and Eric Moulines (2023). "AskewSGD: an annealed interval-constrained optimisation method to train quantized neural networks". In: International Conference on Artificial Intelligence and Statistics. PMLR, pp. 3644–3663.

In this chapter, the weights w of the NN are represented with few bits only at the end of the training process, i.e. for the inference phase. During the training steps (see Section 1.2.1 for gradient descent references), the weights are represented in full-precision, and gently pushed towards quantization values.

2.1 Introduction

The use of deep neural networks (DNNs) on computing hardware such as mobile and IoT devices with limited computational and memory resources is becoming increasingly important. This has led to a growing area of research focused on reducing the model size and inference time of DNNs; in this area, the overall goal is to keep the loss of accuracy below an acceptable level compared to "full-precision" implementations (i.e.16 or 32 bits, and no reduction on the architecture). These methods include, for example, model pruning, neural architecture search, novel efficient architecture design, and low-rank decomposition. In this work, we focus on network quantization, where weights and activations are quantized to lower bit widths, allowing for efficient fixed-point inference and reduced memory bandwidth usage; see, for example, Courbariaux, Bengio, and David 2015; Jacob et al. 2018; Darabi et al. 2018; Choukroun et al. 2019; Lei Deng et al. 2020; Qin et al. 2020; Bhalgat et al. 2020; Chmiel et al. 2021 and references therein. Quantized neural networks (QNNs) have attracted many research efforts. Nevertheless, the challenge of closing the accuracy gap between full-precision and quantized networks remains open, especially for extremely low-precision arithmetics (e.g. binary). The task of learning a quantized neural network (QNN) can be formulated as minimising the training loss with quantization constraints on the weights, i.e.,

$$\min_{w \in \mathcal{Q}} f(w),\ f(w) = \mathbb{E}_{(x,y)\sim D^{\text{data}}}[\ell(\text{nn}(x,w),y)], \tag{2.1}$$

where $\mathcal{Q} \subset \mathbb{R}^d$ is the set of quantization levels, d is the number of parameters (network weights and biases), ℓ is the training loss (e.g. the cross-entropy or square loss), $\text{nn}(x,w)$ is the DNN prediction function, D^{data} is the training distribution. The quantization constraints in the above program make it an extremely difficult task: the underlying optimization problem is non-convex, non-differentiable, and combinatorial in nature. Optimization of smooth functions of integer valued variables (and even quadratic ones like the max-cut problem in graph theory) is known to be NP-hard (Garey and D. S. Johnson 1980). The challenge is to find algorithms that can produce a sensible approximate solution with a manageable computational effort. Inspired by mixed-integer nonlinear programming (MINLP) problems, several approaches using geometric, analytic, and algebraic techniques have been proposed to transform the discrete problem into a continuous problem. Examples include the use of global or concave optimization formulations, semidefinite programming, and spectral theory (see e.g. Mitchell, Pardalos, and Resende 1998; Bussieck, Drud, and Meeraus 2003; Horst and Tuy 2013; Beck and Teboulle 2000; Murray and Ng 2010). However, these types of approaches are doomed to fail in the NN context because the number of parameters is several orders of magnitude larger than for classical MINLP problems.

For large training data sets and number of variables d, stochastic gradient-based (first-order) methods for finding minimizers of (2.1) are often the only manageable option (see Section 1.2.1). Several methods have been proposed which transform the loss function (2.1) into a differentiable surrogate (with possibly an additional penalty term) to "favor" quantized solutions. The general approach is to introduce real-valued "latent" weights $w \in \mathbb{R}^d$ from which the quantized weights are generated; in the binary case, it is classical to use the $\text{sign}(\cdot)$ function or a differentiable surrogate thereof. The simplest method, called BinaryConnect (BC) (Courbariaux, Bengio, and David 2015), is based on straight-through estimators (STE) that ignores the sign conversion in computing the gradient with respect to the latent weights w. BC reaches state-of the art performance on elementary classification tasks and is still a competitive baseline method for more sophisticated problems. Extensions of STE has also been used for more general QNN by Chmiel et al. 2021; Sun, N. Wang, C.-Y. Chen, Ni, Agrawal, Cui, Venkataramani, El Maghraoui, et al. 2020; Choi et al. 2018; Z. Wang et al. 2019.

However, despite its success in NN inference, the STE method does not rely on solid theoretical groundings and may be shown to fail on simple low-dimensional benchmarks - even with convex objective functions; see Bai, Y.-X. Wang, and Liberty 2018. We discuss this method and its recent improvements in the paragraph on related works.

2.2 Related works

We focus on BNN and QNN that replace floating-point multiplication and addition operations with efficient fixed-point arithmetic. We do not consider algorithms that use low-bit computations at the learning stage; see Sakr and Shanbhag 2018; J. Chen et al. 2020. Given the abundance of works, we focus mostly on methods used in our benchmarks.

Binary NN: The first attempt to train BNN is BinaryConnect (BC) (Courbariaux, Bengio, and David 2015; Hubara et al. 2016) which is the first algorithm to implement Quantization Aware Training (QAT); see (Gholami et al. 2021; W. Zhao et al. 2020; Y. Guo 2018; Nagel et al. 2021) and the references therein. BC uses full precision latent weights. On the forward path, the latent weights are binarized. On the backward path, classical backpropagation is applied to update the latent weights, using a differentiable proxy of the binarization function in the gradient calculation. The most common implementation uses the identity proxy, resulting in the straight-through estimator (STE). Although the neural network parameters are highly compressed (and quantization errors can be large), the BC-STE estimator and its numerous recent improvements perform satisfactorily in many benchmarks and have become a de facto standard; see Hu, P. Wang, and Cheng 2018; Faraone et al. 2018; H. Le et al. 2021; Anderson and Berg 2018. Some original methods bypass the use of latent weights. Helwegen et al. 2019 updates binary weights directly with a flipping rule based on an exponential moving average of gradients computed by backpropagation. Such methods have recently been successfully used in Laydevant et al. 2021 with equilibrium propagation instead of backpropagation. Although these methods give reasonable results on standard benchmarks, they do not have strong theoretical guarantees. It is easy to find counterexamples where BinaryConnect or BinaryRelax (phase 2 in Yin et al. 2018) do not converge.

ProxQuant (PQ) (Bai, Y.-X. Wang, and Liberty 2018), Proximal Mean-Field (PMF) (Ajanthan, Dokania, et al. 2019), Mirror Descent (MD) (Ajanthan, K. Gupta, et al. 2021), and Rotated Binary Neural Networks (RBNN) (M. Lin et al. 2020) formulate the task of training BNNs as a constrained optimization problem and discuss different methods to generate binary weights from real-valued latent weights. All of these methods have in common that they use gradual annealing of the conversion mapping, in the sense that, unlike BC and its variants, the latent weights are not projected onto a finite set of quantization values in the forward path. Instead, a force is applied to gradually push the latent weights to the quantization constraints, in a manner reminiscent of homotopy methods for solving nonlinear systems or penalty barrier in nonlinear optimization.

BNN as Variational Inference (VI): Training binary neural networks can also be approached with VI; see among others Raiko et al. 2015; Peters and Welling 2018; Roth et al. 2019. Instead of optimizing binary weights, the parameters of Bernoulli distributions are learned using the VI Bayesian learning rule; see e.g. Khan and Rue 2021. Even if unbiased estimators of the ELBO are available, classical methods like MuProp (Gu et al. 2016) or REINFORCE with variance-reduction baselines (Andriy Mnih and Gregor 2014) have a prohibitively high variance. The use of Gumbel-Softmax (GS) trick (Jang, Gu, and Poole 2016; Maddison, Mnih, and Teh 2017)

has been advocated in Meng, Bachmann, and Khan 2020, but as noted in Shekhovtsov 2021, Section 4 there is an issue in the implementation which paradoxically enables the training. The connections between STE algorithms and their many variants - including MD - and VI methods are further discussed in Shekhovtsov and Yanush 2021.

Quantized NN: The STE estimator is easily adapted to QNN by adding a projection step onto the set of quantization levels in the forward pass (S. Zhou et al. 2016); see (Choi et al. 2018; Sun, N. Wang, C.-Y. Chen, Ni, Agrawal, Cui, Venkataramani, El Maghraoui, et al. 2020; Chmiel et al. 2021) and the references therein. To mitigate performance loss reported in early work from S. Zhou et al. 2016, a number of attempts has been proposed. One possible way is to increase the NN size (Zagoruyko and Komodakis 2016), or the number of channel for convolution layers (A. Mishra, Nurvitadhi, et al. 2017; McDonnell 2018). Knowledge distillation has also been considered with some success (A. Mishra and Marr 2017). A teacher network (typically very large (Z. Liu, Shen, et al. 2020) and trained in full-precision) is employed to help the QNN training (the student network).

In QNN, the choice of the quantizer and the normalization of the weights (at each layer) play a key role. Many works have been devoted to the design of non-uniform or statistical (distribution dependent) quantizers; see (Banner et al. 2018; Hou and Kwok 2018; Bhalgat et al. 2020; Liang et al. 2021; Fournarakis and Nagel 2021; A. Zhou et al. 2017; Y. Zhou et al. 2018) and the references therein. Statistical quantizers are often more efficient, but they are more complex to implement and often require fine tuning (Zhaoyang Zhang et al. 2021).

A number of works have considered formulating the quantization problem as an optimization problem (H. Li et al. 2017; F. Li, B. Zhang, and B. Liu 2016; C. Zhu et al. 2016; Carreira-Perpinán and Idelbayev 2017; Leng et al. 2018; Polino, Pascanu, and Alistarh 2018), but the proposed methods rely on assumptions which may not hold for deep neural networks (Y. Guo 2018). In Moons et al. 2017; T.-J. Yang, Y.-H. Chen, and Sze 2017; Esser et al. 2015, the QNN training is tackled as an energy efficiency problem, whereas Gong et al. 2019 propose a Differentiable Soft Quantization (DSQ) to efficiently train QNN.

Activation function Quantization: We have so far described the quantization of the network weights. But an efficient implementation also requires the quantization of the activation functions. For BNN, (M. Kim and Smaragdis 2016; Hubara et al. 2016; Rastegari et al. 2016) proposed to use sign$(\cdot)$ function, but this approach significantly affects the performance. More complex quantization schemes have been considered in Choi et al. 2018 alleviating performance degradation. Hybrid formats FP8 (N. Wang et al. 2018) or INT8 (Wiedemann et al. 2020; Banner et al. 2018) were successfully employed to achieve a low precision training. Recent works have proposed to jointly optimize the quantization parameters (of weights and activations) and the weights parameters. This task can be done by modifying the learning loss or by minimizing the quantization error (C. Zhu et al. 2016; D. Zhang et al. 2018; Y. Li, Dong, and W. Wang 2019).

2.3 Algorithm derivation

In this section we first introduce the Muehlebach and Jordan 2021 (MJ) algorithm for smooth constrained optimization, initially proposed in a convex setting. We describe the algorithm in full generality and then show how to adapt the MJ algorithm to the QNN setting.

The MJ algorithm Consider the following optimization problem:

$$\min_{w \in C} f(w), \quad C = \{w \in \mathbb{R}^d : h(w) \geq 0\}, \tag{2.2}$$

where $f : \mathbb{R}^d \to \mathbb{R}$ denotes the objective function, $h : \mathbb{R}^d \to \mathbb{R}^{n_h}$ define the inequality constraints. We assume that the feasible set C is non-empty and compact and that the functions f and h are continuously differentiable. We stress that neither f nor C are assumed to be convex. Standard solutions to find a local minimizer of (2.2) use either a projected gradient descent algorithm or "non-linear" projection like mirror descent. However, C might have a complicated form, in which case computing the projection on C might require to solve a non-trivial optimization algorithm in itself (and may fail to be properly defined). The basic idea behind Muehlebach and Jordan 2021's proposal is to "skew" the search direction in order to force the algorithm to find a minimizer of (2.2) without constraining the sequence $(w_k)_{k \in \mathbb{N}}$ to the feasible set. For any $w \in \mathbb{R}^d$, define by $\mathrm{I}(w)$ the set of active constraints

$$\mathrm{I}(w) = \{i \in \{1, \dots, n_h\}, h_i(w) \leq 0\}. \tag{2.3}$$

Under mild assumptions (basically, Muehlebach and Jordan 2021 assume that Mangasarian Fromowitz constraint qualification conditions hold everywhere and not on the feasible set only) the tangent and normal cones of C at $w \in C$ are given by:

$$T_C(w) = \{v \in \mathbb{R}^d, \nabla h_i(w)^\top v \geq 0, \text{for all } i \in \mathrm{I}(w)\}, \tag{2.4}$$

$$N_C(w) = \{-\sum\nolimits_{i \in \mathrm{I}(w)} \lambda_i \nabla h_i(w), \lambda_i \in \mathbb{R}_+\}. \tag{2.5}$$

Moreover, the Karush-Kuhn-Tucker (KKT) conditions hold Borwein and Lewis 2006, Theorem 7.2.9: if w^* is a local minimizer of (2.2), then $w^* \in \mathcal{Z}$, where

$$\mathcal{Z} := \{w \in C : 0 \in -\nabla f(w) - N_C(w)\}. \tag{2.6}$$

The MJ algorithm (Muehlebach and Jordan 2021) generates iterates in $\mathbb{R}^d$ as follows:

$$\begin{cases} w_{t+1} = w_t + \eta_t v_t \\ v_t = \arg\min_{v \in V_\alpha(w_t)} (1/2)\|v + \nabla f(w_t)\|^2 \end{cases}, \tag{2.7}$$

where $(\eta_t)_{t>0}$ is a non-increasing sequence of positive step sizes, $\alpha > 0$ is an hyper-parameter, and the sets $V_\alpha(w)$ are defined as:

$$V_\alpha(w) = \{v \in \mathbb{R}^d : \nabla h_i(w)^\top v \geq -\alpha h_i(w) \text{ for all } i \in \mathrm{I}(w)\}. \tag{2.8}$$

If $w \in C$ and $i \in \mathrm{I}(w)$, then $h_i(w) = 0$ and thus $V_\alpha(w)$ reduces to $T_C(w)$. The set $V_\alpha(w)$ can be considered as an extension of the tangent cone "outside" of the feasible set. Note also that $V_\alpha(w)$, for all $w \in \mathbb{R}^d$, is a convex polyhedron whose construction includes only the active constraints.

By construction, whenever $h_i(w_t) \leq 0$, $\nabla h_i(w_t)^\top v_t \geq -\alpha h_i(w_t)$. Thus, in Equation (2.7), the velocity v_t is chosen to match the unconstrained gradient flow $-\nabla f(w_t)$ as closely as possible, subject to the velocity constraint $v_t \in V_\alpha(w_t)$ (this is illustrated on a simple example in Figure 2.1, for different values of $\alpha > 0$). A striking difference from the classical projected gradient algorithm is that the MJ approach is based on a local approximation of the feasible set. This local approximation includes only the active constraints and is guaranteed to be a convex polyhedron even if the underlying feasible set is not convex. In "classical" constrained optimization algorithms,

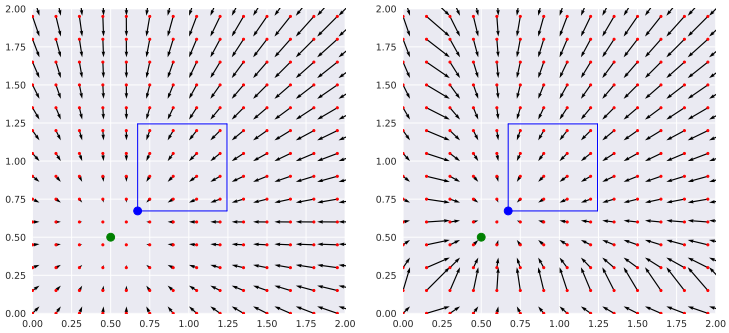


Figure 2.1: The vector field of velocities for $\epsilon = 0.3$, and $\alpha = 0.1$ (left panel) or $\alpha = 1.0$ (right panel). Here, $f(w^1, w^2) = ((w^1 - 0.5)^2 + (w^2 - 0.5)^2)/3$ and $h(w^1, w^2)^\top = (\epsilon - ((w^1)^2 - 1)^2, \epsilon - ((w^2)^2 - 1)^2)$. The border of the set of constraints is shown in blue, and the minimizer of the constrained and unconstrained optimization problems are shown with blue and green dots, respectively.

constraints are typically handled by direct reference to positions, meaning that the iterates w_t, for all $t \geq 0$, must lie in the constraint set C.

`AskewSGD` description In Muehlebach and Jordan 2021 the convergence of the MJ algorithm was proven under the condition that the function f and the set C are convex. We now adapt this algorithm to the QNN problem, removing the requirement that f, C are convex and, furthermore, replacing ∇f by a mini-batch stochastic gradient $\widetilde{\nabla} f$.

We consider f, the training loss, written as $f(w) = \frac{1}{|\mathcal{D}|} \sum_{j=1}^{|\mathcal{D}|} f_j(w)$, where $|\mathcal{D}|$ is the size of the training set, and f_j is the loss associated with the j-th observation.

We relax the quantization constraints $w^i \in \mathcal{Q}$, $i \in \{1, \ldots, d\}$, to a sequence of "smoothed" interval constraints. The set of quantization values $\mathcal{Q}$ is defined coordinate wise: $\{c_1^i, \ldots, c_{M^i}^i\}$, where M^i is the number of quantization values for a given coordinate i. We assume for full generality a different scalar quantizer for each coefficient; we typically use different scalar quantizers for each layer of the NN (but the same quantizer for the coefficients in the same layer). For $\omega \in \mathbb{R}$ such that $\omega \in [c_1^i, c_{M^i}^i]$, we define

$$\phi^i(\omega) = (\omega - c^i_{Q^i(\omega)})^2 (\omega - c^i_{Q^i(\omega)+1})^2,$$

where $Q^i(\omega)$ is the unique index satisfying $c^i_{Q^i(\omega)} \leq \omega < c^i_{Q^i(\omega)+1}$. If $\omega < c_1^i$ we define $\phi^i(\omega) = (\omega - c_1^i)^2$, and if $\omega > c^i_{M^i}$ we define $\phi^i(\omega) = (\omega - c^i_{M^i})^2$. For any $\epsilon \in [0,1]$ and $\omega \in \mathbb{R}$, define $\psi^i_\epsilon(\omega) := \epsilon - \phi^i(\omega)$ and consider the feasible set

$$C_\epsilon = \{w \in \mathbb{R}^d : \forall\, i \in \{1, \ldots, d\}, h_{\epsilon,i}(w) := \psi^i_\epsilon(w^i) \geq 0\}. \tag{2.9}$$

For each $\epsilon \in (0,1)$, we consider the constrained optimization problem $\mathcal{P}_\epsilon : \min_{w \in C_\epsilon} f(w)$. It is easily seen that $\bigcap_{\epsilon > 0} C_\epsilon = \mathcal{Q}$, recovering the constraints of the QNN problem. We therefore define a decreasing sequence $(\epsilon_n)_{n \geq 0}$ of numbers in $[0,1]$ such that $\lim_{n \to \infty} \epsilon_n = 0$ and solve (approximately) the sequence of problems $(\mathcal{P}_{\epsilon_n})_{n \in \mathbb{N}}$.

Here we must notice that the set $V_\alpha(w)$ is empty if and only if there is $1 \leq i \leq d$ such that $w^i = (c_{Q^i(\omega)} + c_{Q^i(\omega)+1})/2$. For such a point, there is no "best" direction, so we chose it arbitrarily

by specifying that the i-th coordinate must go to the right (see the following clipping convention). A symmetric choice prescribing a left direction is also possible. Moreover, since the set of such w is of Lebesgue-measure zero, we can hope that we will never stumble upon such a point (this is further guaranteed by the fact that the iterates converge to C_ϵ, implying that such points asymptotically never occur).

We denote by $\mathcal{Z}_\epsilon := \{w \in C_\epsilon : 0 \in -\nabla f(w) - N_{C_\epsilon}(w)\}$, the set of KKT points of $\mathcal{P}_\epsilon$. Notice that any element of $N_{C_\epsilon}(w)$ can be written as $(-\lambda^1 \psi'_\epsilon(w^1), \dots, -\lambda^d \psi'_\epsilon(w^d))$, with $\lambda^i \geq 0$ and $\lambda^i \neq 0$ only if $\psi_\epsilon(w^i) = 0$. Therefore, $w \in \mathcal{Z}_\epsilon$ if and only if for every $i \in \{1, \dots, d\}$,

$$\begin{cases} \nabla_i f(w) = 0 & \text{if } \psi_\epsilon(w^i) > 0 \\ \operatorname{sign}(\nabla_i f(w)) = \operatorname{sign}(\psi'_\epsilon(w^i)) & \text{if } \psi_\epsilon(w^i) = 0, \end{cases} \tag{2.10}$$

where for $i \in \{1, \dots, d\}$, $\nabla_i f(w)$ is the partial derivative of f w.r.t. w^i. In this setting, the set of active constraints and V_α can be written down as:

$$\begin{cases} \mathrm{I}_\epsilon(w) = \{i \in \{1, \dots, d\} : \psi_\epsilon(w^i) \leq 0\}, \\ V_{\epsilon,\alpha}(w) = \{v \in \mathbb{R}^d : v^i \psi'_\epsilon(w^i) \geq -\alpha \psi_\epsilon(w^i) \text{ for } i \in \mathrm{I}_\epsilon(w)\}. \end{cases} \tag{2.11}$$

Let w be such that $w^i \neq (c_{Q^i(\omega)} + c_{Q^i(\omega)+1})/2$ for all $i \in \{1, \dots, d\}$. For $u \in \mathbb{R}^d$, denote by

$$s_{\epsilon,\alpha}(u, w) = \operatorname*{arg\,min}_{v \in V_{\epsilon,\alpha}(w)} 1/2 \, \|v + u\|^2. \tag{2.12}$$

This problem admits an explicit solution: $[s_{\epsilon,\alpha}(u,w)]^i = -u^i$ if $\psi_\epsilon(w^i) > 0$ or $-\psi'_\epsilon(w^i)u^i \geq -\alpha\psi_\epsilon(w^i) \geq 0$ and $[s_{\epsilon,\alpha}(u,w)]^i = -\alpha\psi_\epsilon(w^i)/\psi'_\epsilon(w^i)$, otherwise. Note that when $w^i \to (c_{Q^i(\omega)} + c_{Q^i(\omega)+1})/2$, the quantity $\psi'_\epsilon(w^i)$ converges to zero, and thus $[s_{\epsilon,\alpha}(u,w)]^i$ might diverge to infinity. To alleviate this problem, we furthermore clip the update. For $(a, b) \in \mathbb{R} \times \mathbb{R}_+$, define $\operatorname{clip}(a, b)$ equal to a if $|a| \leq b$ and to $b \operatorname{sign}(a)$ otherwise. Choose $V_{\epsilon,c} > 0$ and let $s^c_{\epsilon,\alpha}$ be defined for $i \in \{1, \dots, d\}$, $w^i \neq (c_{Q^i(\omega)} + c_{Q^i(\omega)+1})/2$, by:

$$[s^c_{\epsilon,\alpha}(u,w)]^i = \begin{cases} -u^i & \text{if } \psi_\epsilon(w^i) > 0 \text{ or } -\psi'_\epsilon(w^i)u^i \geq -\alpha\psi_\epsilon(w^i) \geq 0; \\ \operatorname{clip}(-\alpha\psi_\epsilon(w^i)/\psi'_\epsilon(w^i), V_{\epsilon,c}) & \text{otherwise.} \end{cases} \tag{2.13}$$

We set by convention $[s^c_{\epsilon,\alpha}(u,w)]^i = V_{\epsilon,c}$ if $w^i = (c_{Q^i(\omega)} + c_{Q^i(\omega)+1})/2$. For given α, ϵ, `AskewSGD` is summarized in Algorithm 2. Under mild assumptions, we establish the convergence of `AskewSGD`.

Algorithme 2 : `AskewSGD` algorithm

Data : sequence of step sizes (η_t); size of the mini-batch $b \leq |\mathcal{D}|$; $w_0 \in \mathbb{R}^d$

for t=1, . . . , T **do**
- Sample a minibatch of b observations $\{j_1, \dots j_b\}$ in $\{1, \dots, |\mathcal{D}|\}$;
- Compute the Stochastic Gradient $\widetilde{\nabla} f(w_t) = 1/b \sum_{i=1}^b \nabla f_{j_i}(w_t)$;
- Compute the update direction $v_t = s^c_{\epsilon,\alpha}(\widetilde{\nabla} f(w_t), w_t)$;
- Update the parameter $w_{t+1} = w_t + \eta_t v_t$.

end

Consider the following assumptions.

A5. *For $j \in \{1,\dots,|\mathcal{D}|\}$, the function f_j is d-times continuously differentiable and has L_{f_j}-Lipschitz continuous gradients.*

A6. *The stepsizes $(\eta_t)_{t\geq 0}$ are positive, $\sum_{j=0}^{\infty}\eta_t = \infty$ and $\sum_{j=0}^{\infty}\eta_t^2 < \infty$.*

Notice that **A**6 holds for (η_t) of the form $(1/t^\delta)$, with $\delta \in (1/2, 1]$. **A**5 will ensure the stability of AskewSGD (i.e. the iterates are bounded with probability one). Moreover, **A**5 implies that $f(\mathcal{Z}_\epsilon)$ is of empty interior, as a consequence of the Sard's theorem (see Lemma 22).

Theorem 5. *Assume **A**5-**A**6 and $0 < \epsilon \leq \inf_{1\leq i\leq d}\inf_{1\leq j\leq M^i}|c_j^i - c_{j+1}^i|^4/16$, where $\{c_j^i\}$ are the quantization levels. Then, $f(w_t)$ converges and $\lim_{t\to\infty}\mathrm{d}(w_t, \mathcal{Z}_\epsilon) = 0$ almost surely.*

Note that the condition on ϵ ensures that the projection of C_ϵ onto the i-th coordinate is a disconnected set of M^i intervals. The proof is based on a general convergence result of Davis et al. 2020, on asymptotic behavior of stochastic approximation of differential inclusion (DI). In our particular case, the corresponding DI is $\dot{\mathrm{y}}(t) \in -\nabla f(\mathrm{y}(t)) - N_{C_\epsilon}(\mathrm{y}(t))$ (we might notice here that this DI is also the continuous-time limit of the projected gradient method). Definitions and important results on DIs and their stochastic approximations can be found in Appendix A.1.1.

The proof of Theorem 5 is done in several steps (see Appendix A.1 for complete derivations). First we prove that almost surely, the sequence of iterates (w_t) converges to C_ϵ (see Lemma 23). Then we show that an update step of AskewSGD can be written as $w_{t+1} = w_t - \eta_\nabla f(w_t) + \eta_t\rho_{t+1} - \eta_t u_t$, where $\rho_{t+1} = \nabla f(w_t) - \widetilde{\nabla} f(w_t)$ and u_t approximates an element of $N_C(w_t)$. We show the convergence of $\sum_{j=1}^{t}\eta_j\rho_{j+1}$ in Lemma 24, and complete the proof by applying Theorem 20, which is adapted from Davis et al. 2020, Theorem 3.2.

Forward pass quantization For completeness, we finally describe the quantization of the activation function when AskewSGD is used to train a deep NN. During the forward pass, we employ a round-to-nearest approach INT4 quantization methods for the activations, taken from Chmiel et al. 2021. We make use of Statistics Aware Weight Binning (SAWB) of (Choi et al. 2018), which finds the optimal scaling factor that minimizes the quantization error based on the statistical characteristics of activation distribution. As emphasized by (Chmiel et al. 2021; Choi et al. 2018), non-linearities of loss and activation functions make unnecessary the use of an unbiased scalar quantizer. After scaling, we use a uniform quantization (e.g., INT4): the set of quantization values $\mathcal{Q}$ is defined coordinate wise: $\{c_1^i,\dots,c_{M^i}^i\}$. The quantization values $\{c_1^i,\dots,c_{M^i}^i\}$ are the integer from the quantization interval (e.g., $\{-8,-7,\dots,8\}$ for INT4). After quantization, both the weights and the activation are rescaled using the scaling factor calculated layerwise.

2.4 Experiments

We evaluate the performance of AskewSGD with weights quantized with 1, 2, and 4 bits. While BNN performs well on some simple benchmarks, it lags significantly behind full precision NN on more demanding tasks. QNN with higher precision and quantization of the activations offers a trade-off between performance and computation efficiency. For simplicity, we refer to [Wx/Ay] as a neural architecture with x-bit precision weights and y-bit precision activations. Details of the implementations and complementary experiments are reported in Appendix A.2. In all experiments, ϵ is annealed throughout the training process during successive episodes. Our experiments show that the initial value for ϵ is not critical. We use a logarithmic schedule. Given a fixed ϵ, we run the algorithm until the test error does not improve, and then reduce it by using

the last iterates as the starting point for the next round. For example, in the experiments of Table 2.2 and Table 2.1, the initial value for ϵ is 1, and we reduce it as K^t with $K = 0.88$. We can set K to different values ($\frac{1}{2}$, 0.8 were tested) as long as $K < 1$.

2.4.1 1-bit quantization

We evaluate the performance of `AskewSGD` [W1/A32] on four tasks: a convex problem, a 2D toy example and two classical image classification benchmarks.

Convex toy example We compare `AskewSGD` , BinaryConnect (Courbariaux, Bengio, and David 2015) and AdaSTE (H. Le et al. 2021) in a logistic regression problem. We generate $|\mathcal{D}| = 6000$ feature vectors $\{x_k\}_{k=1}^{|\mathcal{D}|}$ of dimension $d = 10$, drawn independently from the uniform distribution in $[-1, 1]$. We randomly choose an optimal vector w_* on the vertices of the hypercube and generate the labels as follows: $y_k \sim \text{Bernoulli}(\{1 + e^{-x_k^\top w_*}\}^{-1})$. For completeness, we study how a SGD converges with full precision to the optimal point w_* of this convex problem. All methods are trained for 25 epochs using the SGD optimizer. The learning rate is set to 1 and the gradients are computed on random batches of 1000 samples. For AdaSTE, we have used the code [1] with the hyperparameters specified in the package for annealing. `AskewSGD` performance is on par with

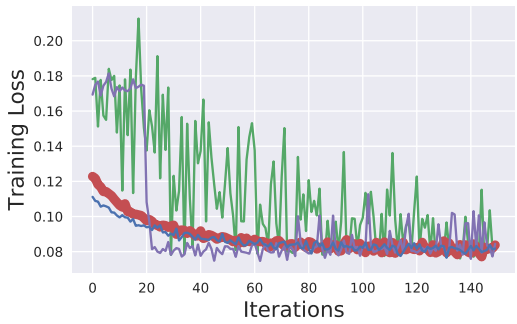


Figure 2.2: Training losses for the logistic regression problem with batches of size 1000. BinaryConnect (green), `AskewSGD` (blue), full Precision (red), AdaSTE (purple) methods. The x-axis represents the iteration index. Red points are made artificially bigger to help visualization.

full precision method, while the STE variants all suffer from strong oscillations (see Shekhovtsov 2021; Shekhovtsov and Yanush 2021; Bai, Y.-X. Wang, and Liberty 2018). Figure 2.2 illustrates the effects of such oscillations on the convergence . In all settings, AdaSTE converges faster than BC, but still all STE variants exhibit a larger loss compared to other methods. Additional results are reported in Appendix A.2.

Non-convex toy example We consider the binary classification problem on "2 moons dataset" presented in Meng, Bachmann, and Khan 2020. The training dataset consists of 2000 samples (split into 2 moon-like clusters in 2 dimensions) and 200 test samples; see Appendix A.2. We train a BNN with 9 neurons. In this low-dimensional environment, we can enumerate all $2^9 = 512$ possible binary configurations and select the best one(s). Our method is compared with 4 different approaches: a full precision NN, BinaryConnect (Courbariaux, Bengio, and David 2015), AdaSTE (H. Le et al. 2021), and exhaustive search. All methods are trained for 50

[1] https://github.com/intellhave/AdaSTE

epochs with logistic loss. The full precision NN is trained using the Adam optimizer (Kingma and J. Ba 2014) with default hyperparameters, a learning rate of 0.1, and a batch of size 100. The BinaryConnect approach is trained using the Adam optimizer with default hyperparameters, a learning rate of 1, and a batch of size 100. The AdaSTE method is implemented using a learning rate of 1. Our method uses the same parameters as the STE method, and we set α to 4. For a single run, we plotted the training loss in Figure 2.3. For a fair comparison, in

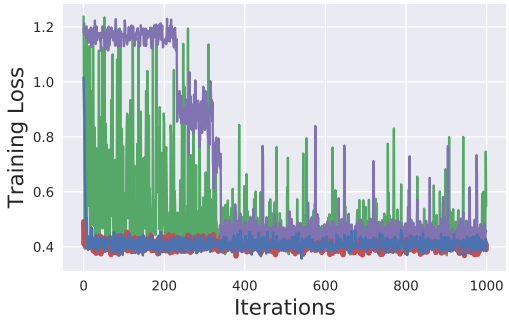


Figure 2.3: Training losses for the toy non-convex problem with batches of size 100. BinaryConnect (green), AskewSGD (blue), full Precision (red), AdaSTE (purple) methods. The x-axis represents the iteration index. Red points are made artificially bigger to help visualization

Table A.1 in Appendix A.2 we report the performance averaged on 50 random experiments of the various methods on the test set (full precision reaches a 2.045 ± 0.005 loss, when exhaustive search presents a 2.1 loss, AskewSGD reaches 2.11 ± 0.01, AdaSTE and STE reach 2.24 ± 0.10 and 2.32 ± 0.11 respectively).

The exhaustive search shows that different configurations lead to near-optimal performance (see Figure A.3 in Appendix A.2). Here we chose the configuration that achieves the lowest loss on the test set. AskewSGD outperforms AdaSTE and BC.

Computer vision tasks In this section, we benchmark AskewSGD with BC (Courbariaux, Bengio, and David 2015; Hubara et al. 2016), Mirror Descent (Ajanthan, K. Gupta, et al. 2021), and ProxQuant (Bai, Y.-X. Wang, and Liberty 2018) on classical computer vision datasets. To avoid overloading the figures, the AdaSTE results are reported separately in Appendix A.2. We also report performance with a standard full precision NN and a full precision NN projected onto the hypersphere. We compare the different methods using the same NN architecture. We do not add bias on any neuron. We introduce batch normalisation (without learning scale and bias parameters) after each layer. We emphasise that our method is generic and not specific to the classical ConvNet architecture. We have also obtained SOTA results for large ResNet architectures (see Table 2.3).

We use the standard data augmentations and normalizations for all the methods. AskewSGD is implemented in Pytorch, and the experiments are run on a NVIDIA Tesla-P100 GPU. Standard multiclass cross-entropy loss is used for all experiments unless otherwise stated. We perform cross-validation of the hyperparameters, such as the learning rate, the tradeoff between constraints α, the rate of increase of the annealing hyperparameter, and their respective schedules. The search space for tuning the hyperparameters and the final hyperparameters can be found in Appendix A.2. All models are fine-tuned for 100 epochs using the Adam (Kingma and J. Ba 2014) optimizer with dynamics of 0.9 and 0.999, and batch of size 100.

The NN with full precision is trained with an initial learning rate of 0.08. The projected full precision NN uses a projected gradient algorithm. The same hyperparameters as the "plain" algorithm are used, except that a deterministic projection onto the hypersphere is performed for each iteration $w_{k+1} = \Pi(w_k - \eta_k \widetilde{\nabla} f(w_k))$. For BinaryConnect, we use the method described in Courbariaux, Bengio, and David 2015. For Mirror Descent (MD), we use the code[2] from Ajanthan, K. Gupta, et al. 2021 and implement the version $\tanh(\cdot)$ (without annealing and with $\alpha = 0.01$ and $\mu = 100$ when training). ProxQuant was run with the parameters specified in Bai, Y.-X. Wang, and Liberty 2018. Note ProxQuant does not initially quantize the fully-connected layer, and add full precision biases. For fair comparison we have tested ProxQuant with all layers binarized. The AskewSGD method is described in Algorithm 2. Multiple values for α in $[0.1, 5]$ are considered. The precision threshold ϵ is decreased from epoch to epoch: it is set to 1 at the beginning and then exponentially annealed to $.88^t$ in the last 50 epochs, where t is the epoch. After the last step, all weights are within an interval of length $\epsilon_{\text{final}} = 0.01$ of $\{-1, +1\}$.

For AskewSGD we apply the function $\text{sign}(\cdot)$ to our NN before evaluating it on the test set. For a fair comparison, each method was randomly initialized and independently executed 5 times. An intensive learning rate search was also performed independently for each method. The learning rate at epochs $[20, 40]$ is divided by 2 for all methods.

Most neural networks use the inference accuracy of image classification as an evaluation metric. We first compared the training/testing accuracy with the CIFAR-10 dataset (Krizhevsky, G. Hinton, et al. 2009), which consists of 50000 training images and 10000 test images (in 10 classes). Figure 2.4 illustrates the distribution of the weights of the first convolutional layer (the behavior is similar for other layers) at epochs 20, 39, 55, and 99. We have also tested

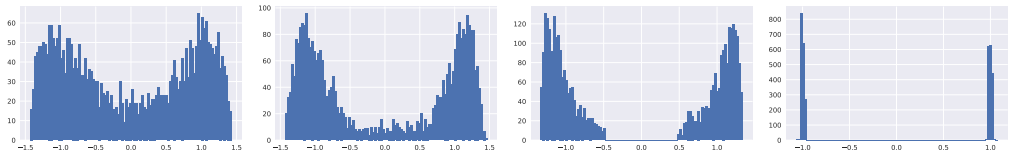


Figure 2.4: Histogram of weights during the training phase of our AskewSGD [W1/A32] on CIFAR-10.

Table 2.1: Test accuracy (average over 5 random experiments) for AskewSGD [W1/A32] at several epochs.

Epochs	ϵ	CIFAR-10	TinyImageNet
50	0.88	75.77	8.74
65	0.15	88.37	31.97
90	0.006	88.84	46.96

AskewSGD [W1/A32] on the TinyImageNet dataset (Y. Le and X. Yang 2015) with a ResNet-18. TinyImageNet has 200 classes and each class has 500 (RGB) training images, 50 validation images, and 50 test images. To train ResNet-18 we follow the common practices used for training NNs: we resize the input images to 64×64 and then randomly flip them horizontally during training. During testing we center-crop them to the corresponding sizes. In Figure 2.5, the loss increases slightly in the final steps as the constraints become more stringent. However, this

[2] https://github.com/kartikgupta-at-anu/md-bnn

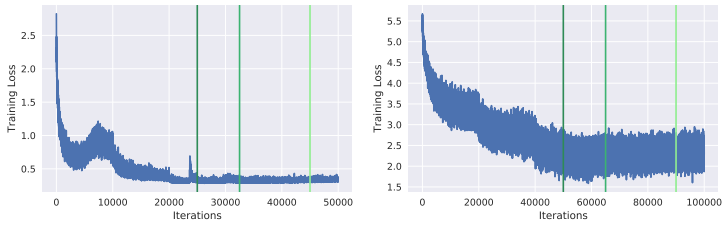


Figure 2.5: Training Loss of AskewSGD [W1/A32] on CIFAR-10 (left) and TinyImageNet (right). The x-axis represents the batch iterations and green vertical lines correspond to epochs [50, 65, 90].

increase in training loss remains moderate and the final performance in both the training set and the test set is the best among all methods. Some test accuracies are presented in Table 2.1 at several epochs (identified with green lines in Figure 2.5) with the corresponding precision ϵ. The best test classification accuracies of the binary networks obtained with each method are listed in Table2.2. For reproducibility none of the concurrent results are reported from existing papers, but each approach has been independently rerun from the available codes. Compared to other binarization algorithms, our method consistently yields better or equivalent results, while narrowing the performance gap between binary networks and floating-point counterparts on multiple datasets to an acceptable level. The performance of the projected gradient method highlights the strength of our method: we do not simply project the iterates on the nearest constraint set, but progressively push the iterates towards a smoothed version of the constraints (see Section 2.3), which leads to better results.

Table 2.2: Best Test accuracy (average and variance over 5 random experiments) after 100 training epochs.

Method	CIFAR-10	TinyImageNet
Full-precision [W32/A32]	89.46 ± 0.07	56.46 ± 0.46
BinaryConnect [W1/A32]	88.33 ± 0.29	42.35 ± 0.33
MD [W1/A32]	88.13 ± 0.25	34.89 ± 0.36
ProxQuant [W1/A32]	88.22 ± 0.28	48.79 ± 0.32
Projected gradient [W1/A32]	71.34 ± 0.46	11.78 ± 0.67
AskewSGD [W1/A32]	**88.98** ± 0.35	**50.23** ± 0.37

2.4.2 Low-bit quantization

We consider now low-bit weight quantization and activation quantization. To fully benefit from low precision arithmetic, one should also tackle the problem of gradient quantization (Chmiel et al. 2021; Sun, N. Wang, C.-Y. Chen, Ni, Agrawal, Cui, Venkataramani, El Maghraoui, et al. 2020) and accumulation. We keep the last fully connected layer in full-precision, following Z. Liu, Shen, et al. 2020; Chmiel et al. 2021. We evaluate the performance of AskewSGD [W1/A32], AskewSGD [W2/A4], and AskewSGD [W4/A4] on TinyImageNet and ImageNet (Russakovsky et al. 2015) datasets with a ResNet-18 network. For AskewSGD we project NN weights onto the set of quantization values before evaluating it on the test set. For ImageNet, we keep the

first convolution layer in full-precision. We use the same pre-processing (centering and data normalization) for all the methods: we resize the input images to 256×256 and then randomly crop them to 224×224 while centering them to the appropriate sizes during training. Standard multiclass cross entropy loss is used. All models are fine-tuned for 200 epochs using the Adam (Kingma and J. Ba 2014) optimizer with dynamics of 0.9 and 0.999 and a batch of size 512. All methods are trained with an initial learning rate of 0.06 for TinyImagenet and 0.1 for ImageNet. The same hyperparameters are used as in the previous section for TinyImageNet. For ImageNet, the learning rate at epochs $[30, 60, 90]$ is divided by 10 for all methods. We have run the code[3] from LUQ and adapted it to TinyImageNet dataset. For a fair comparison we compute neural gradients in full precision. The results for the method Ultra-low (Sun, N. Wang, C.-Y. Chen, Ni, Agrawal, Cui, Venkataramani, El Maghraoui, et al. 2020) are taken from Chmiel et al. 2021.

We decided not to include the regularisation-based binarization approach (Ding et al. 2019), which addresses the activation binarization problem, in our benchmark. We have also not included in our benchmark improvements of BC methods which have been proposed in (S. Zhou et al. 2016; Z. Liu, B. Wu, et al. 2018; Bethge et al. 2020; Rastegari et al. 2016; Martinez et al. 2019); these methods are all based on the STE (Courbariaux, Bengio, and David 2015) optimizer to update quantized weights. These methods have been shown to be outperformed by AdaBin (Tu et al. 2022) and ReacNet Z. Liu, Shen, et al. 2020. The latter are currently SOTA methods for energy-friendly inference on the ImageNet dataset. Note that these binary approaches still have a gap in terms of full precision performance, which needs to be addressed by modifying the NN structure (Z. Liu, Shen, et al. 2020). For ReacNet and AdaBin, we have reported the best results of Tu et al. 2022 for ResNet-18 on ImageNet.

Table 2.3: Best Test accuracy (single run for ImageNet due to longer training time) after 200 training epochs. * indicates the results are directly reported from existing literature.

Method	TinyImageNet	ImageNet
Full-precision [W32/A32]	56.46 ± 0.46	69.32
ReacNet [W1/A1] (2 steps)	-	65.5*
AdaBin [W1/A1] (2 steps)	-	66.4*
Ultra-low [W4/A4]	-	68.27*
LUQ [W2/A4]	54.14 ± 0.42	-
LUQ [W4/A4]	55.69 ± 0.32	68.41
AskewSGD [W2/A4]	53.54 ± 0.28	66.45
AskewSGD [W4/A4]	**55.85** ± 0.30	**68.51**

AskewSGD performs better than or on par with state of the art QNN methods and offers a shorter gap to full precision performances compared with best BNNs.

2.5 Conclusion

In this chapter, we present AskewSGD a novel framework for QNN training based on an annealed sequence of interval-constrained nonconvex optimization problems solved by an algorithm inspired by Muehlebach and Jordan 2021. For each of these subproblems we give theoretical guarantees. AskewSGD outperforms or is on par with other QNN training methods on all considered tasks.

[3] https://openreview.net/forum?id=clwYez4n8e8

Chapter 3

A heuristic to obtain BNN

In this second chapter dedicated to Quantized Neural Networks, we move the focus toward the training of BNNs (Binary Neural Networks). In this chapter, part of the contributions are kept private.

As a CIFRE PhD candidate, I was advised, and working with the "Intelligent Computing and Communications" team at Huawei Technologies France. Under this context, a heuristic for training BNNs was developed and presented in the patent V. M. Nguyen and Leconte 2022:

Nguyen Van Minh and **Louis Leconte** (2022). Apparatus and method for training binary deep neural networks. url:https://patents.google.com/patent/WO2023217370A1/.

In this chapter, we explain how this method is applied to the training and fine-tuning of DNNs, with energy efficiency constraints. This work is currently under review.

3.1 Introduction

Running deep models, i.e., *inference*, requires considerable computational resources, it is yet the tip of the iceberg. Deep model *training* is an iterative process involving abundant computation and data for learning. It incurs storing multiple temporal variables and buffers for gradient computation and parameter optimization. This intense process is further repeated for hyper-parameter tuning, running for weeks or months on specialized equipment, resulting in another order of magnitude of carbon footprint and computational resource requirement (Strubell, Ganesh, and McCallum 2019).

The vast majority of works that target the resource constrained training bottleneck focus on the number of arithmetic operations (García-Martín et al. 2019; Qin et al. 2020) rather than the consumed energy/memory. However, it has been shown that OPs number is meaningless, even harmful, because it does not map directly to actual system complexity. Instead, energy and memory consumption are the consistent and efficient measures of computing hardware (Sze et al. 2017; Sze et al. 2020; T.-J. Yang, Y.-H. Chen, Emer, et al. 2017; Strubell, Ganesh, and McCallum 2019). In particular, data movement dominates computing in energy consumption and is strictly tied to system architecture, memory hierarchy, and dataflow (Kwon et al. 2019; Sim, S. Lee, and L.-S. Kim 2019; X. Yang et al. 2020; Y.-H. Chen et al. 2016). Therefore, design effort subjected to reducing OPs *alone* is inefficient. Currently, the main approach to tackle such bottleneck is quantization. It is becoming popular for LLMs (Frantar et al. 2022; J. Lin et al. 2023; J. Kim et al. 2023) to enable inference on affordable devices. But only post-training quantization is available on standard GPUs. Better quantized models can be obtained through quantization-aware training (S. Gupta et al. 2015; D. Zhang et al. 2018; Jin et al. 2021; Yamamoto 2021; C.-W. Huang, T.-W. Chen, and J.-D. Huang 2021; Umuroglu et al. 2017), and quantized training (J. Chen et al. 2020; Sun, N. Wang, C.-Y. Chen, Ni, Agrawal, Cui, Venkataramani, Maghraoui, et al. 2020; Yukuan Yang et al. 2022; Chmiel et al. 2021), that reduce the numeric precision of weights, activations, and dataflow from full-precision (FP) to finite-precision format.

A special case of quantization-aware training is binarized neural networks (BNNs) which were first proposed by Courbariaux, Bengio, and David 2015; Courbariaux, Hubara, et al. 2016 and have been followed by a huge amount of subsequent contributions (Gholami et al. 2021; W. Zhao et al. 2020; Y. Guo 2018; Nagel et al. 2021). This design usually binarizes weights and activations to obtain principal forward computation blocks in binary. It learns binarized weights via full-precision latent ones, which are updated by the classical gradient descent backpropagation. The gradient of the binarized variables is usually approximated by a differential proxy of the binarization function, which is most often the identity proxy. Many concurrent approaches (Bai, Y.-X. Wang, and Liberty 2018; Ajanthan, Dokania, et al. 2019; Ajanthan, K. Gupta, et al. 2021; M. Lin et al. 2020; Leconte, Schechtman, and Moulines 2023) formulated the BNN learning task as a constrained optimization problem and discussed different methods to generate binary weights from real-valued latent ones. In practice, these works showed that BNNs could achieve state-of-the-art accuracy in study-level classification problems such as CIFAR-10 or MNIST, but suffer significant accuracy drop on more challenging problems such as ImageNet (Rastegari et al. 2016; Qin et al. 2020). Besides the reduced network approximation capacity due to lower data precision (S. Zhou et al. 2016), the use of full-precision optimizers for estimating binary weights are the causes of this degradation. To compensate for this accuracy loss, most recent prominent works (Z. Liu, Shen, et al. 2020; Nie et al. 2022; N. Guo et al. 2022) used multiple full-precision components in the network, whereas only a few dataflows have remained binary.

In this chapter we aim to answer whether energy-friendly deep learning is possible both for inference and training all while maintaining performance. For that, we explore Boolean notions to define networks that are predominantly Boolean, with low energy demands, and that

are trained in the binary domain. Extensive experimental evaluation is conducted on a set of common vision tasks requiring moderate to higher levels of accuracy.

3.2 Related works

Energy consumption is a fundamental metric for measuring hardware complexity. However, it requires specific knowledge of computing systems and makes it hard to estimate. Only few results are available, though experimental-based and limited to specific tested models, e.g., Y. Gao et al. 2020; Shao and Brooks 2013; Mei et al. 2014; Bianco et al. 2018; Canziani, Paszke, and Culurciello 2016; García-Martín et al. 2019. Although experimental evaluation is precise, it requires considerable implementation efforts while not generalizing. In addition, most relevant works are only limited to inference and not training (Y.-H. Chen et al. 2016; Kwon et al. 2019; X. Yang et al. 2020). Therefore, developing an analytic method to efficiently estimate training energy consumption is desirable.

Regarding NN architectures, significant advances have been made on BNNs (Binarized Neural Networks) for the ImageNet classification task, driving their performance to higher grounds (N. Guo et al. 2022; Z. Liu, Shen, et al. 2020; Tu et al. 2022; C. Lee et al. 2022; Y. Zhang, Zhiru Zhang, and Lew 2022; Xing et al. 2022; Martinez et al. 2019; Y. Wang et al. 2023). These works, which attempt to reduce the accuracy gap between BNNs and full-precision networks, typically target the primary sources of the computational burden of CNNs, essentially convolutions, data-streams memory (both numeric type and size) and network depth. Consequently, modern BNNs are improved over the following three main areas.

Binarization strategy. It seeks to efficiently binarize real-valued data. The sign function is the primary alternative to binarize data-streams, with additional constraints on the data like clipping Y. Zhang, Zhiru Zhang, and Lew 2022; N. Guo et al. 2022. ReActNet (Z. Liu, Shen, et al. 2020) is a prominent work that proposes RSign, a more general alternative to sign, which deals with the fact that distributions may be shifted or biased. A more recent option, Tu et al. 2022 argues that binary values $\{1, -1\}$ might restrain the approximation capabilities of BNNs, which is why they binarize activations to two real values for more representative power.

Optimization strategy. Since latent-based training (Courbariaux, Bengio, and David 2015) remains the underlying method for updating binarized weights, a differential proxy of sign is required. Different or modified alternatives to straight-through-estimator (STE) have been proposed (Z. Liu, Shen, et al. 2020). Piece-wise polynomials and hyper-parameterized tanh have been used (Nie et al. 2022). The latent-based approach requires storing both binary and real parameters during training. Furthermore, this approach typically requires a sequential training of multiple stages where activations and weights get progressively converted from full-precision to binary types (N. Guo et al. 2022; Y. Zhang, Zhiru Zhang, and Lew 2022; Xing et al. 2022), resulting in longer training times. Modern BNN methodologies (Xing et al. 2022; Z. Liu, Shen, et al. 2020; N. Guo et al. 2022; Y. Zhang, Zhiru Zhang, and Lew 2022; Tu et al. 2022; C. Lee et al. 2022; C. Liu et al. 2022) agree on the fact that using knowledge distillation (KD) closes the gap between BNNs and full-precision models for which additional data augmentation is needed to reduce the common overfitting of BNNs (N. Guo et al. 2022). In most cases, a single teacher like ResNet34 or ResNet50 suffices to significantly increase the accuracy. More recently, using multi-KD with four teachers, BNext (N. Guo et al. 2022), reached performances not reported before. In this sense, existing works have investigated BNNs from the perspective that network binarization is considered as a plugin feature to an existing full-precision DNN. For KD, the goal is to transfer the knowledge of a teacher model to a smaller student model. If trained from scratch, the student model generally performs worse than its teacher. However, under the

supervision of the teacher network, the binary network can preserve the learning capability and thus obtain comparable performance to the teacher network. Consequently, the process still requires full-precision model training, and cannot tackle the complexity problem of the network training. Furthermore, KD-based training depends on specialized teachers on a particular task, thus reducing functionality on new data. Helwegen et al. 2019; E. Wang et al. 2021 proposed some heuristic and improvement of the classic BNN latent-based optimizer.

Architecture design. ResNet (Z. Liu, Shen, et al. 2020; Z. Liu, B. Wu, et al. 2018; N. Guo et al. 2022; Bethge et al. 2020) and MobileNet (Z. Liu, Shen, et al. 2020; N. Guo et al. 2022) are the most frequent layouts. In S. Zhou et al. 2016; Rastegari et al. 2016 the authors experimented with Alexnet. Among these methodologies, the basic blocks have been greatly transformed. Common modifications include additional shortcuts, automatic channel scaling with Squeeze-and-Excitation (Y. Zhang, Zhiru Zhang, and Lew 2022) or block duplication plus concatenation in the channel domain (Z. Liu, Shen, et al. 2020; N. Guo et al. 2022). Recent alternatives incorporate modules that better adapt the input domain to binary dataflows Xing et al. 2022, replace standard convolutions with lighter pointwise convolutions (C. Liu et al. 2022), or propose 1-bit alternatives of linear projections (Hongyu Wang et al. 2023).

Since the release of ReActNet, the best results are obtained by alternating low-precision dataflows to full-precision after every binary convolution within the network. These works substantially rely on real-valued dataflows during feedforward such as PReLU, Batch Normalization, FP scaling, and further boost accuracy via KD. This highlights the need for native binary neural networks (V. M. Nguyen 2023), and a precise complexity evaluation method to be able to assess gains in regards of memory, energy, and latency.

3.3 Method

3.3.1 Boolean training

The design and training of Boolean layers follows the principle proposed by V. M. Nguyen 2023. For illustration purpose, Algorithm 3 presents a pseudo code for a Boolean fully-connected layer that uses Boolean logic B = XNOR. In the forward pass, at iteration t, input $x^{l,t}$ is buffered for later use in the backward, and the jth neuron output at kth sample is computed as:

$$x_{k,j}^{l+1,t} = w_{0,j}^{l} + \sum_{i=1}^{d_I} \mathrm{B}\left(x_{k,i}^{l}, w_{i,j}^{l}\right), \tag{3.1}$$

$\forall k \in [1,b], \forall j \in [1,d_O]$ where b, d_I, d_O are, respectively, the training mini-batch, layer input and output size. The processing of Boolean design includes pointwise logic followed by counting and majority vote. Since real arithmetic has been the base of engineering systems, all existing software tools have been optimized for supporting real arithmetic operations (in fact 16 or 32 bits). Therefore, instead of designing a Boolean processing tool, we implement real arithmetic, and use the following map:

$$\begin{cases} \text{True} \to +1 \\ \text{False} \to -1. \end{cases} \tag{3.2}$$

In practice, we use Pytorch, and all weights w have a value in the binary set $\{+1,-1\}$ instead of {True, False}. Also, all XNOR and counting are translated into pointwise multiplications and additions. As a consequence, for the choice B = XNOR, Equation (3.1) before activation translates into $x_{k,j}^{l+1,t} = w_{0,j}^{l} + \sum_{i=1}^{d_I} x_{k,i}^{l} w_{i,j}^{l}$. The sums are in the integer range and the output of the neuron

Algorithme 3 : Pseudo-code for Boolean training with B = XNOR.

Input : Learning rate η, nb iterations T;
Initialize
  $m_{i,j}^{l,0} = 0$; $\beta^0 = 1$;
end
for $t = 0, \ldots, T-1$ **do**
  /* 1. Forward */
  Receive and buffer $x^{l,t}$;
  Compute $x^{l+1,t}$ following Equation (3.1);
  /* 2. Backward */
  Receive $g^{l+1,t}$;
  /* 2.1 Backpropagation */
  Compute and backpropagate $g^{l,t}$ following Equation (3.3);
  /* 2.2 Weight update */
  $C_{\text{tot}} := 0$, $C_{\text{kept}} := 0$;
  foreach $w_{i,j}^{l}$ **do**
    Compute $q_{i,j}^{l,t+1}$ following Equation (3.4);
    Update $m_{i,j}^{l,t+1} = \beta^t m_{i,j}^{l,t} + \eta^t q_{i,j}^{l,t+1}$;
    $C_{\text{tot}} \leftarrow C_{\text{tot}} + 1$;
    if $XNOR(m_{i,j}^{l,t+1}, w_{i,j}^{l,t}) = \text{True}$ **then**
      $w_{i,j}^{l,t+1} \leftarrow \neg w_{i,j}^{l,t}$; /* invert */
      $m_{i,j}^{l,t+1} \leftarrow 0$;
    else
      $w_{i,j}^{l,t+1} \leftarrow w_{i,j}^{l,t}$; /* keep */
      $C_{\text{kept}} \leftarrow C_{\text{kept}} + 1$;
    end
  end
  Release buffer $x^{l,t}$;
  Update $\beta^{t+1} \leftarrow C_{\text{kept}}/C_{\text{tot}}$;
  Update η^{t+1};
end

is given as follows: $y_{k,j}^{l+1,t} = 2 \cdot \mathbb{1}(x_{k,j}^{l+1,t} \geq \tau) - 1$, where τ is a scalar threshold. In other words, in the proposed BNN, the output of a layer is just the sign function (translated with τ) applied to the dot product of the input and the weights.

In the backward pass, layer l receives $g^{l+1,t}$ from downstream layer $l+1$. Then, backpropagated signal $g^{l,t}$ (line 8 in Algorithm 3), is computed following V. M. Nguyen 2023 as:

$$g_{k,i}^{l,t} = \sum_{j=1}^{d_O} \mathbb{1}_{\{g_{k,j}^{l+1,t} w_{i,j}^{l,t} \geq 0\}} |g_{k,i,j}^{l,t}| - \sum_{j=1}^{d_O} \mathbb{1}_{\{g_{k,j}^{l+1,t} w_{i,j}^{l,t} < 0\}} |g_{k,i,j}^{l,t}|, \tag{3.3}$$

$\forall k \in [1, b], \forall i \in [1, d_I]$. Here the XNOR and counting operations have been replaced by the

standard product and sum operations. Optimization signal at line 11 in Algorithm 3 is given according to V. M. Nguyen 2023 as:

$$q_{i,j}^{l,t+1} = \sum_{k=1}^{b} \mathbb{1}_{\{q_{i,j,k}^{l,t} \geq 0\}} |q_{i,j,k}^{l,t}| - \sum_{k=1}^{b} \mathbb{1}_{\{q_{i,j,k}^{l,t} < 0\}} |q_{i,j,k}^{l,t}|, \tag{3.4}$$

$\forall i \in [1, d_I], \forall j \in [1, d_O]$. Finally, the weights are updated in lines 14–20 of Algorithm 3 following the rule formulated in V. M. Nguyen 2023: a weight w is inverted only when its associated accumulator m reaches a defined value.

3.4 Experiments

Our design was benchmarked on different computer vision tasks: classification (using CIFAR-10 (Krizhevsky, G. Hinton, et al. 2009) and ImageNet (Krizhevsky, Sutskever, and G. E. Hinton 2012)) and super-resolution (using DIV2k (Agustsson and Timofte 2017; Timofte et al. 2017), Set5 (Bevilacqua et al. 2012), Set14 (Zeyde, Elad, and Protter 2012), BSD100 (J.-B. Huang, Singh, and Ahuja 2015), and Urban100 (Martin et al. 2001)).

In addition, to conceal that our Boolean Logic is advantageous for edge device learning, we explore the scenario where a pretrained model is deployed to an edge device, i.e. fine tuning. On this regard, we analyze two tasks: classification and segmentation. For classification, the trained Boolean VGG-Small architecture is fine-tuned over CIFAR-100. For segmentation, the trained Boolean ResNet18 is used as backbone on DeepLabv3 (L.-C. Chen et al. 2017) and fine-tuned over the Cityscapes (Cordts et al. 2016), and Pascal VOC 2012 (Everingham et al. n.d.) datasets.

In all benchmarks, the Boolean model was built following the sketch of the baseline full-precision (FP) architecture such that its arithmetic layers are Boolean and removing FP-specific components, such as ReLU, PReLU activations or BatchNorm (unless mentioned otherwise). Following the literature (Chmiel et al. 2021), the first and the last layers were kept in FP. Adam (Kingma and J. Ba 2014) was used as the optimizer of these FP layers, while our Boolean optimizer was used on the remaining Boolean part. The full details of all experiments are kept private for now.

3.4.1 Image classification

Proof of the proposed concept was initially validated on CIFAR-10 with VGG-Small (Simonyan and Zisserman 2014) baseline. In the experiments, our boolean architecture follows the layout of Courbariaux, Bengio, and David 2015, except that we exclude batch normalization. This configuration obtained a top-1 accuracy of 90.29±0.09% (estimated over six repetitions), showing similar performance to Courbariaux, Bengio, and David 2015, which has 32-bit activations and is full-precision during training (see Table 3.1). Higher performances are obtained when including batch normalization after convolutions and the activation from Z. Liu, B. Wu, et al. 2018 (referred to as Boolean with BN in the table), with a classification performance equal to $92.37 \pm 0.01\%$ (estimated over five repetitions) which is almost 1 point closer to the FP counterpart.

3.5 Conclusion

We have presented a method for training deep neural networks that is provably efficient for resource-constrained environments. Our results suggest that full-precision performance can be

Method	W/A	Acc.(%)	Cons.(%)	Gain (×)
Full-precision (D. Zhang et al. 2018)	32/32	93.80	100.00	1.00
BinaryConnect (Courbariaux, Bengio, and David 2015)	1/32	90.10	PRIVATE.	PRIVATE.
XNOR-Net (Rastegari et al. 2016)	1/1	89.83	PRIVATE.	PRIVATE.
BNN (Hubara et al. 2017)	1/1	89.85	PRIVATE.	PRIVATE.
Boolean w/o BN (Ours)	1/1	90.29	PRIVATE.	PRIVATE.
Boolean with BN (Ours)	1/1	**92.37**	PRIVATE.	PRIVATE.

Table 3.1: Experimental results with the standard VGG-Small (ending with 3 FC layers) baseline on CIFAR-10. Energy consumption is evaluated on 1 iteration. 'Cons' and 'Gain' are the energy consumption and gain w.r.t. the FP baseline.

totally recovered by enlarged Boolean models while gaining multifold complexity reduction. One can fine-tune these energy-efficient models on edge devices for specific tasks. Our experiments highlight that Boolean models can handle finer tasks, contrary to the misbelief that binary models only work for image classification.

Chapter 4

Efficient BNN training in the FL context

We tackle in the next two chapters the challenging problem of training jointly a set of nodes (e.g. edge devices with low computational/communication capacity) in the synchronous FL context. In standard centralized FL, one assumes that (i) all nodes can compute gradients on their local dataset (i.e. energy/memory is not a bottleneck), and (ii) all nodes are synchronous with the CS (i.e. bandwidth and compute time are not bottlenecks).

As a first step, in this chapter we demonstrate how one can integrate the method presented in the previous Chapter 3 into the FL framework. Here we assume nodes are edge devices with constrained computational power, and we consider 2 options to tackle the bandwidth bottleneck: `FedBool` and `MajBool`. These algorithms are presented in the following patent:

Louis Leconte and Nguyen Van Minh (2023). Method and system for performing federated learning. Filled in Sept.23, to be published.

We also had the opportunity to present our work at the FMEC/FLTA 2023 conference. This chapter is mainly based on the corresponding publication (Leconte, Moulines, et al. 2023):

Louis Leconte, Eric Moulines, and Van Minh Nguyen. (2023). "Federated Boolean Neural Networks Learning". In: 2023 Eighth International Conference on Fog and Mobile Edge Computing (FMEC). IEEE, pp. 247–253.

4.1 Introduction

Training machine learning models can often be a very challenging process, requiring significant computational resources and time. The advent of federated learning (FL) has generated significant interest in both academia and industry as the need for large-scale, distributed machine learning systems grows. In particular, FL can protect privacy-sensitive data by keeping them on local devices. In the FL paradigm, a shared central model is trained by participating agents under the guidance of a central server, while training data is securely stored on edge devices. This approach has been highlighted in notable studies such as Y. Zhao et al. 2018; Horváth, Sanjabi, et al. 2022.

The inherent design of FL ensures privacy as each client participates in training on the device and shares model updates only with the central server. However, this feature creates a major bottleneck - communicating updates from local clients to the central server. As the number of parameters and workers increases, this problem becomes more apparent. Addressing this communication cost is a key issue in the quest for more efficient federated learning (Kairouz, H. B. McMahan, et al. 2021; J. Wang, Charles, et al. 2021).

Another advantage of FL is that it distributes the computational load across the networked agents. Although this is beneficial from a privacy perspective, implementing federated learning algorithms in IoT systems comes with a number of challenges. For one, FL requires a level of computational power from each agent that typically exceeds what is required for a purely inferential task, leading to potential computational bottlenecks. A delicate balance must be struck between the frequency of sending local updates and the performance of the central model.

The use of DNNs on computing hardware such as mobile and IoT devices is becoming increasingly important. IoT devices often have limitations in terms of memory and computational capacity. Quantization is a potential solution to this problem. Quantized neural networks (QNNs) represent weights, activations, or even gradients with a small number of bits, allowing for efficient fixed-point inference and reduced memory bandwidth usage; see e.g. (Courbariaux, Bengio, and David 2015; Jacob et al. 2018; Lei Deng et al. 2020; Leconte, Schechtman, and Moulines 2023; Chmiel et al. 2021; Tu et al. 2022). In this area, the general goal is to keep the loss of precision to an acceptable level compared to floating-point implementations. But few works address the training of QNNs in the setting of FL.

In this chapter, we take a step toward answering this question by proposing 2 FL algorithms tailored for training clients with low computational capacity: `FedBool`, and `MajBool`.

The approach of `FedBool` borrows from Ji and L. Chen 2022, but without the need for full-precision updates. Each client uses a pure Boolean network as defined by V. M. Nguyen 2023, which facilitates the computation of inference and novel approximate backpropagation rules on its local dataset. Importantly, `FedBool` does not require full precision computations or updates on the client side. The backpropagation signals are further vector-quantized using Leconte, Dieuleveut, et al. 2021 and then forwarded to the central server. Aggregation and optimization on this central server follows the algorithm proposed by V. M. Nguyen 2023.

In contrast, `MajBool` proposes to apply the optimization rules as defined by V. M. Nguyen 2023 directly at each client. Then, the updated Boolean weights are forwarded to the central server. We introduce a new majority rule inspired by V. M. Nguyen 2023 for aggregating the Boolean weights of the clients and then updating the Boolean weights of the server.

4.2 Related works

Quantization of neural networks attempts to reduce the computation bit width while maintaining model accuracy. Several works perform DNN compression, especially with network pruning (Han, H. Mao, and Dally 2015), efficient network design (Howard et al. 2017; H. Chen et al. 2020) or network quantization (Jin et al. 2021; Yamamoto 2021; Fei et al. 2022; Gholami et al. 2021). Network quantization is remarkably promising because it cleverly reduces both memory and inference simultaneously. While conventional Convolutional Neural Networks (CNNs) rely on 32-bit-wide data during inference and training, network quantization reduces the numerical precision of data flows, weights, and activations to computationally lighter values that still have sufficient representational power to maintain model accuracy; see Jin et al. 2021; Chmiel et al. 2021. Several works have obtained excellent results with quantized neural networks. In general, quantized aware training (QAT) yields better performance than post training quantization (PTQ). A special case of network quantization that offers an interesting trade-off between complexity and performance is binary neural networks (BNNs), in which the activations and operations use computationally-efficient binary primitives. However, designing BNNs is a difficult task because minimizing the encoding bits of the representation often leads to a significant reduction in accuracy. For a given number of neurons, each neuron is susceptible to potential errors due to the introduced quantization step. Consequently, some researchers have trained neural networks with binary constraints, while others have relaxed the bit-width constraint to multiple bits; see Gholami et al. 2021; W. Zhao et al. 2020; Y. Guo 2018; Nagel et al. 2021. BinaryConnect (BC) (Courbariaux, Bengio, and David 2015; Hubara et al. 2016) uses latent weights with full-precision. On the forward path, the latent weights are quantized. On the backward path, classical backpropagation is applied to update the latent weights, using a differentiable proxy of the quantization function in the gradient calculation. The most common implementation uses the identity proxy, resulting in the straight-through estimator (STE). Although the NN parameters are highly compressed (and quantization errors can be large), the BC-STE estimator and its numerous recent improvements perform satisfactorily in many benchmarks and have become a de facto standard; see Hu, P. Wang, and Cheng 2018; Faraone et al. 2018; H. Le et al. 2021; Anderson and Berg 2018. Extensions of STE have also been used for more general QNN of Chmiel et al. 2021; Sun, N. Wang, C.-Y. Chen, Ni, Agrawal, Cui, Venkataramani, El Maghraoui, et al. 2020; Choi et al. 2018; Z. Wang et al. 2019. However, there is very little work that addresses binary neural network training in the context of FL. In Yuzhi Yang, Zhaoyang Zhang, and Q. Yang 2021, the updates of the local latent weights are based on maximum likelihood estimation from the averaged value on the server side. BNNs are trained with the STE optimizer on the client side and transmit latent or binary weights. Sihua Wang et al. 2022 also quantizes the weights using STE, but with an adaptive number of bits. Then the quantized weights are averaged at the server side as in FedAvg (B. McMahan et al. 2017). FedQNN (Ji and L. Chen 2022) quantizes weights, activations, and neural gradients. It also sparsifies and quantizes gradient signals before communicating with the base station. On the server side, the gradients (not the parameters) are averaged for the global model update phase. Then FedQNN uses binary masks and entropy coding to send the weights back to the clients.

FL may face severe communication bottlenecks due to the communication between the server and the clients. Many techniques have been proposed to tackle the communication obstacle, such as infrequent aggregation, sparse compression, and quantization. The key idea of quantization is to use fewer bits to represent the model updates (Alistarh et al. 2017; Bernstein et al. 2018; Leconte, Dieuleveut, et al. 2021; Gorbunov, Burlachenko, et al. 2021), which introduces a trade-off between the communication workload and representation errors. Recently, DASHA (Tyurin and Richtárik 2022) has improved MARINA Gorbunov, Burlachenko, et al. 2021, and the client

always sends unaltered and compressed signals to the central server. In AQG (Y. Mao et al. 2022), a gradient-based FL setting is described, where the number of bits for gradient transmission is adjusted during training and for each client. AdaCGD (Makarenko et al. 2022) also allows clients to adaptively choose arbitrary contractive compression mechanisms during training. Jhunjhunwala et al. 2021 proposes an adaptive quantization strategy. However, their results suggest to increase bit accuracy during training, which is in contrast to FedDQ (Qu, Song, and Tsui 2021).

4.3 Boolean Federated Learning

In the proposed solution, local training on edge devices is based on Boolean learning V. M. Nguyen 2023 to reduce complexity. Local gradients (`FedBool`) or local binary weights (`MajBool`) are reported to the central server, which performs aggregation and sends the final binary weights back to the edge devices.

4.3.1 Boolean Edge Training

Boolean learning presented in V. M. Nguyen 2023 proposes a new method for learning BNNs. As in the previous Chapter 3, we consider a linear layer with an input of size d_I. Given Boolean weights and inputs, $(w_{0,j},\dots,w_{d_I,j,})$, and $(x_1,\dots,x_{d_I})$, the jth Boolean neuron pre-activation is formulated as follows:

$$s_j = w_{0,j} + \sum_{i=1}^{d_I} x_i w_{i,j}, \tag{4.1}$$

where the sums are in the integer range and the output of the neuron is given as follows:

$$y_j = 2 \cdot \mathbb{1}(s_j \geq \tau) - 1, \tag{4.2}$$

where τ is a scalar threshold. As explained in Chapter 3, the Boolean design $\{\pm 1\}$ replaces real-valued weights and activations with Boolean ones, and replaces the floating point multiplication with Boolean logic. Computing the gradient of the loss function w.r.t. some weight w requires to compute the derivative of the activation. During the forward pass we strictly follow Equation (4.1) and Equation (4.2). However during the backward pass, all indicator functions $\mathbb{1}(\cdot)$ are intertwined with $\tanh(\sigma_j^{-1} s_j)$ functions, where σ_j^2 is a variance scaling parameter computed layerwise (V. M. Nguyen 2023).

We briefly describe network training; more details are given in V. M. Nguyen 2023. Let us denote $\mathrm{nn}(x; w)$ as the output of the Boolean neural network given an input x and binary weights w. Following V. M. Nguyen 2023, at the k-th iteration, a mini-batch consisting of b observations, denoted as $\{(y^i, x^i)\}_{i \in \mathcal{B}_k}$, is selected. Then, the gradient of the loss function $\mathcal{L}$ on the mini-batch $g_k = b^{-1} \sum_{i \in \mathcal{B}_k} \nabla \mathcal{L}(\mathrm{nn}(x^i; w), y^i)$ is computed. The mini-batch gradient is then used to update the accumulator

$$m_{k+1}[i] = \beta_k[i] m_k[i] + \eta g_k[i], \quad i \in \{1,\dots,d\}, \tag{4.3}$$

where η is a stepsize and $\beta_k = (\beta_k[1],\dots,\beta_k[d]) \in [0,1]^d$ is a vector given by

$$\beta_k[i] = (1/|\mathcal{N}_i|) \sum_{j \in \mathcal{N}_i} \mathbb{1}(w_k[j] \neq w_{k-1}[j]), \tag{4.4}$$

where $\mathcal{N}_i$ is a predefined neighborhood of the i-th coordinate.

We set $m_0 = \mathbf{0}_{d\times 1}$, $\boldsymbol{\beta}_0 = (\beta_0[1],\dots,\beta_0[d]) = \mathbf{1}_{d\times 1}$. The individual component of the weights

$w_k = (w_k[1], \ldots, w_k[d])$ are updated component-wise according to the following rule: if $w_k[i] =$ True and $m_k[i] \geq 1$, or $w_k[i] =$ False and $m_k[i] \leq -1$, then the weight is inverted $w_{k+1}[i] \leftarrow \neg w_k[i]$ and the accumulator is reset $m_k[i] \leftarrow 0$.

4.3.2 FedBool

Section 4.3.1 describes how to optimize a Boolean neural network on a single client. In FedBool we choose to aggregate multiple gradients coming from different clients. We assume that n clients are available and the central server randomly selects s of them. Each selected client i receives at iteration k the same Boolean weight from the central server and starts computing some optimization signals according to the rule defined in V. M. Nguyen 2023. These gradients $\{g_k^i\}_{i \leq s}$ are compressed and sent to the central server. We choose to use a vector quantization technique (Leconte, Dieuleveut, et al. 2021) to reach high compression rate. More details about this technique will be given in the next Chapter 5. The idea behind vector quantization is to quantize a vector rather than each of its coordinates. A vector quantizer $\mathcal{Q}$ is a mapping which maps $x \in \mathbb{R}^d$ to an element of a codebook $\mathcal{C}_M$, which is a finite subset of $\mathbb{R}^d$ with M elements. Central server and clients share some random seeds that allow them to generate codebooks. Then on the client side, one selects the closest codeword of g_k^i by computing $\arg\min_{c \in \mathcal{C}_M} \|g_k^i - c\|$. Only the index of the closest codeword is transmitted to the central server (thus consuming $\log_2(M)$ bits). At the central server, gradients $\{g_k^i\}_{i \leq s}$ are decompressed and averaged before being fed into the central accumulator m:

$$m_{k+1} = \beta_k m_k + \eta \sum_{i=1}^{s} \mathcal{Q}^{-1}(\mathcal{Q}(g_k^i)), \tag{4.5}$$

We propose to update the central model using the same MAJ logic rule from V. M. Nguyen 2023; see Algorithm 4. One can note all costly operations are done on the central server.

4.3.3 MajBool

The first steps are similar to FedBool. We again assume n clients are available, and the central server randomly selects s of them. Every selected client i receives the same Boolean weight from the central server. But from this point, each client entirely performs locally the optimization steps defined in V. M. Nguyen 2023, and sends its new Boolean weight w^i to the central server. We propose to update the central model using a similar majority logic rule (V. M. Nguyen 2023). First, a pre-vote signal p is computed as the sum of the s client weights:

$$p = \sum_{i=1}^{s} w^i. \tag{4.6}$$

After that, a threshold activation is applied to give a Boolean vote and update the value of the central weight:

$$w_{k+1} = \begin{cases} \text{True}, & \text{if } p \geq sT, \\ \text{False}, & \text{if } p \leq -sT, \\ w_k, & \text{otherwise.} \end{cases} \tag{4.7}$$

The pseudo-code of the proposed method is described in Algorithm 5. We assume that the server performs a number of training epochs $K \geq 1$. At each time step $k \in \{1, \ldots, K\}$, the server holds a Boolean model w_k. At initialization, the central server transmits identical parameters w_0 to all devices. At each time step k, the central server selects a subset $\mathcal{S}_k$ of s clients selected uniformly at random, and the server multicasts its model to all clients in $\mathcal{S}_k$. All clients in $\mathcal{S}_k$

Algorithme 4 : FedBool

Input : Number of steps T, LR η, Selection Size s, Maximum local steps E, Vector quantizer $\mathcal{Q}$;

/* At the Central Server */

Initialize
  Initialize parameters w_0, and server accumulator $m_0 = 0$;
end
for $k = 1, \dots, T$ **do**
  Generate set $\mathcal{S}_k$ of s clients uniformly at random;
  for *all clients* $i \in \mathcal{S}_k$ **do**
    Client receives w_k from server;
    Client accumulates gradients with Bool() ;
    Client quantizes accumulated gradients;
    Client sends quantized gradients to server;
  end
  Update server accumulator $m_{k+1} \leftarrow \beta_k m_k + \eta \sum_{i=1}^{s} \mathcal{Q}^{-1}(\mathcal{Q}(g_k^i))$;
  if w_k = True *and* $m_k \geq 1$ *OR* w_k = False *and* $m_k \leq -1$ **then**
    Flip weight $w_{k+1} \leftarrow \neg w_k$;
    Reset accumulator $m_k \leftarrow 0$;
  end
end

/* At Client i */

function Bool():
  Initialize
    Client receives w and E from the Server;
    Local variables $w^i = w$;
  end
  for $t = 1, \dots, E$ **do**
    Compute local gradient g_k^i following V. M. Nguyen 2023 ;
  end
end function

update their local models based on V. M. Nguyen 2023. Next, the contacted clients transmit their local models $\{w_k^i, i \in \mathcal{S}_k\}$ back to the server. When all requested models arrive at the server, the server computes a pre-vote signal p according to a simple average (see Line 11). Note that, up to this step, our method is similar to FedAvg (B. McMahan et al. 2017; J. Wang, Charles, et al. 2021) where binary weights are averaged. Then, the server updates its weight based on p and on the threshold T.

4.4 Experiments

We test FedBool and MajBool performance on two image classification tasks: MNIST (Li Deng 2012), and CIFAR-10 (Krizhevsky, G. Hinton, et al. 2009). For the MNIST dataset, two training sets are considered: an IID and a non-IID split. In the first case, the training images are randomly distributed among the n clients. In the second case, each client takes two classes (out of the ten possible) without replacement. This process introduces heterogeneity among the clients.

Algorithme 5 : MajBool

Input : Number of steps T, LR η, Selection Size s, Maximum local steps E ;

/* At the Central Server */

Initialize
  Initialize parameters w_0;
end
for $k = 1, \dots, T$ **do**
  Generate set $\mathcal{S}_k$ of s clients uniformly at random;
  for *all clients* $i \in \mathcal{S}_k$ **do**
    Client receives w_k from server;
    Client updates its weight with Bool();
    Client sends its weight w^i to server;
  end
  Compute pre-vote $p \leftarrow \sum_{i \in \mathcal{S}_k} w^i$;
  if $p \geq sT$ **then**
    $w_{k+1} =$ True ;
  else if $p \leq -sT$ **then**
    $w_{k+1} =$ False ;
  else
    $w_{k+1} = w_k$;
  end
end

/* At Client i */

function Bool():
  Initialize
    Client receives w and E from the Server;
    Local variables $w^i = w, m^i = 0$;
  end
  for $t = 1, \dots, E$ **do**
    Compute local gradient g following V. M. Nguyen 2023 ;
    Update local accumulator $m^i \leftarrow \beta^i m^i + \eta g$;
    if $w^i =$ True *and* $m^i \geq 1$ *OR* $w^i =$ False *and* $m^i \leq -1$ **then**
      Flip local weight $w^i \leftarrow \neg w^i$;
      Reset local accumulator $m^i \leftarrow 0$;
    end
  end
end function

For FedBool, we do not directly quantize the whole vector, but split it into buckets of size 16. Without vector quantization one would need $32 \cdot 16 = 512$ bits to transmit each bucket. We randomly generate a codebook $\mathcal{C}_M$, of $M = 2^{10}$ elements. Thus, we only need 0.8 bits per coordinate for transmission.

The standard evaluation measure for FL is the number of communication rounds to achieve target accuracy. In Table 4.1, we compare the best performance obtained with FedBool and MajBool, and the sota Federated methods BiFL (Yuzhi Yang, Zhaoyang Zhang, and Q. Yang 2021), RL-based (Sihua Wang et al. 2022), and FedQNN (Ji and L. Chen 2022). To provide a fair

Table 4.1: Test accuracy (in %) for centralized and Federated frameworks (average and standard deviation when available), with the corresponding bit length for weight, activations and transmitted updates (W/A/B).

Method	W/A/B	MNIST IID	CIFAR-10
Centralized[32-bits]	32/32/–	99.32	90.12
Centralized[1-bit]	1/1/–	97.92	88.67
BiFL (Yuzhi Yang, Zhaoyang Zhang, and Q. Yang 2021)	1/1/1	89.41	–
RL-based (Sihua Wang et al. 2022)	1/1/3	< 65	< 25
FedQNN (Ji and L. Chen 2022)	1/1/4	95.67	62.08
FedQNN (Ji and L. Chen 2022)	1/1/8	96.57	70.54
FedBool	1/1/0.8	**97.15** $\pm$ 0.13	87.58 $\pm$ 0.86
MajBool	1/1/1	95.09 $\pm$ 0.18	**88.77** $\pm$ 0.80

comparison, we track the performance of each algorithm by evaluating the server model against an unseen validation dataset, and average it over 10 random experiments. In all experiments, the last fully connected layer is kept at full precision - we follow here the setting of (Z. Liu, Shen, et al. 2020; Chmiel et al. 2021). This is not a major drawback since they only account for a limited portion of the model parameters.

After simulating n clients, a set of s clients are sampled at random without replacement, we follow the framework proposed in Zakerinia et al. 2022 to simulated Federated Learning. We use the standard data augmentations and normalizations for all methods. FedBool and MajBool are implemented in Pytorch, and experiments are performed on an NVIDIA Tesla-P100 GPU. Standard multiclass cross entropy loss is used for all experiments. All models are fine-tuned with $n = 100$ clients, and $E = 5$ local epochs. We fix the batch at size 128 for CIFAR-10, and 500 for MNIST.

We first report the accuracy of a tiny neural network (two convolutional layers, and two fully connected layers) trained on MNIST. The learning rate for the last (fully connected) layer is set to 0.01, and the Boolean learning rate is set to 10. To be fairly comparable to Yuzhi Yang, Zhaoyang Zhang, and Q. Yang 2021; Sihua Wang et al. 2022; Ji and L. Chen 2022, the total communication rounds is set to 300. The vote threshold in MajBool is set to $T = 0$ for MNIST experiments. We also compare the performance of a VGGsmall (Simonyan and Zisserman 2014) with the CIFAR-10 dataset (Krizhevsky, G. Hinton, et al. 2009), which consists of 50000 training images and 10000 test images (in 10 classes). For CIFAR-10, the learning rate for the last (fully connected) layer is set to 0.001, and the learning rate for Boolean layers is set to 20. In Figure 4.1, we investigate the convergence of FedBool and MajBool methods on the MNIST dataset. They both perform well, and require few server steps to close the gap w.r.t. to the full precision centralized approach. Figure 4.2 indicates that MajBool is sensitive to data heterogeneity. When local datasets are too heterogeneous, local weights flip and reach different points. Aggregating this very different Boolean values results in a weight that offers poor performances. On the other hand, FedBool aggregates gradients, and takes an optimization step after the aggregation. This is known to perform better under data heterogeneity (B. McMahan et al. 2017; Leconte, V. M. Nguyen, and Moulines 2023). In Figure 4.3 we analyse the effect of the voting threshold T on the convergence behaviour. Compared to MNIST experiments, keeping $T = 0$ does not work well on CIFAR-10. This dataset is challenging enough to impact performances when a weight is flipped without a large consensus between clients. On the other hand, asking for a

Figure 4.1: Validation accuracy on the MNIST dataset with an IID split in between $n = 100$ total nodes. Central server selects $s = 20$ clients at each round.

Figure 4.2: Validation accuracy on the MNIST dataset with a non-IID split in between $n = 100$ total nodes.

strict consensus between clients (when $T = 1$, all clients report the same weight) does not offer the best accuracy. An intermediate setting ($T = 0.5$) allows us to get closer to the full precision centralized performance.

4.5 Conclusion

We have provided two centralized Federated Learning methods which incorporate Boolean neural networks. Empirical evaluation shows that `FedBool` and `MajBool` are more efficient than synchronous state-of-the-art mechanisms on several image classification tasks. On-going work investigate the possibility to adapt `FedBool` and `MajBool` to the asynchronous Federated Learning setting where clients may have different computational speeds (Mishchenko, F. Bach, et al. 2022; Leconte, V. M. Nguyen, and Moulines 2023).

Figure 4.3: Validation accuracy for `Ma jBoo1` on the CIFAR-10 dataset with an IID split in between $n = 100$ total nodes. Central server selects $s = 10$ clients at each round.

Chapter 5

An unbiased quantization scheme for highly compressed FL communications

In the previous Chapter 4 we have mentioned the use of vector quantization (VQ) for compressing gradient signals sent to the CS. In this chapter we present an *unbiased* VQ method to deal with the high bandwidth cost of communicating gradient updates between nodes. This technique is successfully applied to the centralized synchronous FL setting in the unpublished/ongoing work Leconte, Dieuleveut, et al. 2021:

Louis Leconte, Aymeric Dieuleveut, Edouard Oyallon, Eric Moulines, and Gilles Pagès. (2021). "DoStoVoQ: Doubly Stochastic Voronoi Vector Quantization SGD for Federated Learning". In: url:https://openreview.net/pdf?id=URc7gYBcjVn.

5.1 Introduction

In this chapter, we consider the Federated Learning framework, in which a potentially large number n of *workers* cooperate to solve the distributed optimization problem:

$$\min_{w\in\mathbb{R}^d} \sum_{i=1}^{n} f_i(w), \tag{5.1}$$

where each function $f_i : \mathbb{R}^d \to \mathbb{R}$ represents the empirical risk on worker $i \in [n]$ (where $[n] = \{1,\dots,n\}$) and d is the ambient dimension of our problem. Each worker potentially holds a fraction of the data, and can share information with a central server, which progressively aggregates and updates the model accordingly (Konečný et al. 2016; Kairouz, H. B. McMahan, et al. 2021).

Stochastic gradient algorithms (Robbins and Monro 1951) are particularly well suited in the *large scale learning* setting (Bottou 1999). The methods can easily be adapted to the distributed (and more generally federated) learning framework; see Kairouz, H. B. McMahan, et al. 2021 and references therein. For synchronous distributed Stochastic Gradient Descent, at every iteration, given the current parameter w_t, each worker computes an unbiased estimate $g_{i,t+1}(w_t)$ of the gradient of the local loss function f_i. The central server then aggregates those oracles and performs the update.

Communicating the gradients from the local workers to the central server is often a major bottleneck. The drastic increase both in the number of parameters and of workers over the last years, has made this problem even more acute. Alleviating the communication cost is one of the crucial challenges of federated learning (Kairouz, H. B. McMahan, et al. 2021, Sec. 3.5). A central idea to tackle this issue is *communication compression*, which consists in applying a lossy compression to the parameters or gradients to be transmitted. The design of new compression schemes (Seide et al. 2014; Alistarh et al. 2017; Bernstein et al. 2018; Yu et al. 2018) and the analysis and adaptation of the learning algorithms (Karimireddy, Rebjock, et al. 2019; N. Agarwal et al. 2018; Wangni et al. 2018; Sebastian U Stich, Cordonnier, and Jaggi 2018; A. Xu, Huo, and H. Huang 2020; Mishchenko, Gorbunov, et al. 2019; Philippenko and Dieuleveut 2020; Gorbunov, Burlachenko, et al. 2021; Gorbunov, Kovalev, et al. 2020; Safaryan, Shulgin, and Richtárik 2020) are extremely active fields of research.

Our main contribution is to introduce a novel **unbiased vector quantization** procedure allowing to reach **high-compression rate**, with a **small computational** overhead. More precisely, our contributions are as follow: first, we introduce Stovoq, a vector quantization algorithm based on unitarily invariant random codebooks to obtain **directionally unbiased** gradient oracles, and introduce a scalar **correction function**, that makes compression operator **unbiased** for a very modest computational cost. In summary, Stovoq is based on the following points, that are developed in Section 5.2.

1. **Vector quantization** The input $x \in \mathbb{R}^D$ is mapped onto its nearest neighbor in a codebook $\mathscr{C}_M = \{c_i\}_{i=1}^M$.

2. **Random codebook.** A **new codebook** is sampled every time a (new) quantization operation is performed. The proposed approach is different from classical VQ which typically uses a random codebook, but which is sampled once and then kept fixed.

3. **Bias removal.** By relying on unitarily invariant distribution for the codewords generation, the quantized value of each vector $x \in \mathbb{R}^D$ is **directionnally unbiased**. The bias only depends on the number and distributions of the random codewords, and on the norm of

the vector to be quantized $\|x\|$. This key property allows to derive a simple way to remove the quantization bias.

We demonstrate the effectiveness of Stovoq by analyzing and empirically comparing its *distortion* when compressing Gaussian vectors and "real" gradients. This simple but reliable metric is one of the best way to assert the quality of a compression technique, and is only rarely used in previous works (Saha, Pilanci, and Goldsmith 2021).
Then, we describe how to use Stovoq within any FL algorithm: this yields the Dostovoq algorithms, that aim at solving the optimization problem $\min_{w\in\mathbb{R}^d} \sum_{i=1}^n f_i(w)$, in dimension d. The process is described in Section 5.4:

4. **Splitting and normalizing gradients.** We split each gradient into *buckets* $(x_l)_{1\leq l\leq L}$ of dimension $D \ll d$, to apply Stovoq on each bucket.

5. **Synchronisation of random sequences of codebooks.** We ensure that those codebooks are independent, at each step and between each machine, by generating a new codebook each time. To avoid any subsequent communication cost, we synchronously generate the codebooks on the central and local servers, by initially sharing random seeds.

Remark that point 1 was also used in Dai et al. 2019. Points 2 to 3 and 5 are novel ideas that have not been leveraged in the FL framework. Finally, we demonstrate the effectiveness of Dostovoq for gradient compression by extensive experiments in Section 5.5 on standard benchmarks like CIFAR10 or ImageNet.

5.2 Stovoq algorithm

Several compression operators[1] have been introduced recently as bandwidth reduction for distributed learning became a major challenge. In this section, we first discuss the importance of unbiasedness of compression operators in Section 5.2.1. We then present the Stovoq compression scheme in Section 5.2.2. Finally, we compare Stovoq to competing approaches, both theoretically and empirically on a small scale example with a high compression rate.

5.2.1 Unbiased gradient estimate to mitigate high compression rates

We first here discuss an important property to mitigate high compression rates in FL settings. A *compression operator* Comp is a (random) mapping on $\mathbb{R}^D$. Consider the following assumption:

A7 (**Unbiased Compression with relatively bounded variance**). *A compression operator* Comp *is unbiased if for any* $x \in \mathbb{R}^d$, $\mathbb{E}[\mathrm{Comp}(x)] = x$ *(w.r.t. the distribution of the random compression operator). It is said to have a* ω*-bounded relative variance, for some* $\omega > 0$*, if it satisfies, for all* $x \in \mathbb{R}^D$, $\mathbb{E}[\|\mathrm{Comp}(x) - x\|^2] \leq \omega\|x\|^2$.

The most classical compressors, especially Q-SGD and Rand-H satisfy **A7** with different ω, see Section 5.3. On the other hand, some compression operators are biased, i.e., $\mathbb{E}[\mathrm{Comp}(x)] \neq x$ for some $x \in \mathbb{R}$. Those operators are often deterministic, as is the case for Top-H compressor. The most classical assumption for biased operators, is the following contractive property along the direction of descent (Karimireddy, Rebjock, et al. 2019; Gorbunov, Kovalev, et al. 2020):

A8 (**Biased Compression with contraction**). *For* $\delta > 0$*, a compression operator is said to be* $1/(1+\delta)$*-contractive if for any* $x \in \mathbb{R}^d$*, we have* $\mathbb{E}[\|\mathrm{Comp}(x) - x\|^2] \leq (1 - 1/(1+\delta))\|x\|^2$.

[1] See Section 5.3 for a detailed related work

Constants ω and δ from these two assumptions are both positive, and become larger as the compression rate increases. Alternative assumptions for the biased case have been introduced in Beznosikov, Horváth, et al. 2020.

Impact of unbiasedness on the compression of a single vector. [2] To understand the interaction between the number of workers n and the compression error, a simple situation is the case in which the workers use *independent and identically distributed compression operators* $(\mathrm{Comp}_i)_{i=1}^n$ to compress the *same vector* $x \in \mathbb{R}^d$. The central node aggregates $\{\mathrm{Comp}_i(x)\}_{i=1}^n$ into $n^{-1}\sum_{i=1}^n \mathrm{Comp}_i(x)$. A bias-variance decomposition of the quadratic error gives:

$$\mathbb{E}[\|n^{-1}\sum_{i=1}^n \mathrm{Comp}_i(x) - x\|^2] = \|\mathbb{E}[\mathrm{Comp}_1(x)] - x\|^2 + n^{-1}\mathbb{E}[\|\mathrm{Comp}_1(x) - \mathbb{E}[\mathrm{Comp}_1(x)]\|^2]. \quad (5.2)$$

The variance of the aggregated vector is reduced by a factor n^{-1} when averaging the messages send by the n workers, while the bias is independent of n. For example, if we use an unbiased compressor satisfying **A** 7, we get

$$\mathbb{E}\left[n^{-1}\sum_{i=1}^n \mathrm{Comp}_i(x)\right] = x, \quad (5.3)$$

and

$$\mathbb{E}\left[\left\|x - n^{-1}\sum_{i=1}^n \mathrm{Comp}_i(x)\right\|^2\right] \leq (\omega/n)\|x\|^2, \quad (5.4)$$

while for a deterministic biased compressor, we obtain that $n^{-1}\sum_{i=1}^n \mathrm{Comp}_i(x) = \mathrm{Comp}_1(x)$ has the same error as any of the individual compressed vector. We therefore pay particular attention to obtaining an unbiased compressor in the following.

5.2.2 Stovoq definitions and main properties.

The basic idea behind VQ is to quantize a vector rather than each of its coordinates. A Vector Quantizer is a mapping $\mathrm{VQ}(\cdot, \mathscr{C}_M) : \mathbb{R}^D \to \mathscr{C}_M$ which maps $x \in \mathbb{R}^D$ to an element of a codebook $\mathscr{C}_M$, which is a finite subset of $\mathbb{R}^D$ with M elements. The code of Stovoq is provided in Algorithm 6, and its crucial steps are described hereafter: we introduce the notion of **(a)** Voronoi quantization scheme, before describing more precisely **(b)** random codebooks, **(c)** whose distributions are invariant by unitary transforms. Then, **(d)** a method to obtain an unbiased Voronoi scheme is presented.

Algorithme 6 : Stovoq with distribution p

Input : $x \in \mathbb{R}^D$, p, M, P, seed s
Output : Codeword index $\mathbf{i}_c$, value $\mathbf{i}_r$

1 Sample $\mathscr{C}_M \sim p$ with seed s ; /* generate codebook with distribution p */
2 $c = \mathrm{VQ}(x, \mathscr{C}_M^p)$; /* perform Voronoi quant. */
3 $\mathbf{i}_c$ = index of c; /* get index of codeword */
4 $r = r_M^p(\|x\|)$; /* find radial bias in table */
5 $\mathbf{i}_r = \mathrm{SQ}(r^{-1})$; /* quantize r on P bits */

[2] The impact of unbiasedness for obtaining optimal convergence complexities in FL is discussed in Section 5.4.

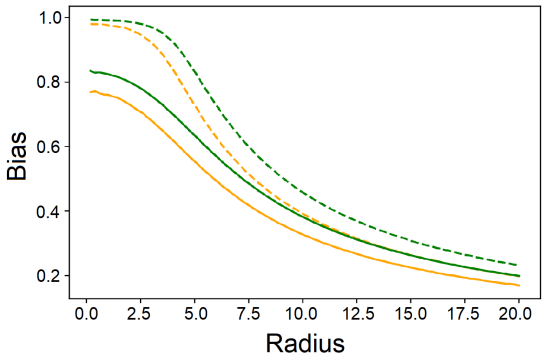


Figure 5.1: function r_M^p for $D = 4$ (dashed) and 16 (solid), $p = \mathcal{N}(0, \mathrm{I}_D)$ and $M = 2^{10}$(orange), and 2^{13}(green).

(a) Voronoi Quantization (VQ). VQ (Pagès and Printems 2003; Pagès and Wilbertz 2018), aims at selecting the closest codeword from $\mathscr{C}_M$, i.e.:

$$\mathrm{VQ}(x, \mathscr{C}_M) \triangleq \operatorname*{arg\,min}_{c \in \mathscr{C}_M} \|x - c\|. \tag{5.5}$$

Unfortunately, for any given $\mathscr{C}_M$, the Voronoi quantizer is not *unbiased*: indeed it is deterministic and $\mathrm{VQ}(x, \mathscr{C}_M) \neq x$ if $x \notin \mathscr{C}_M$. A classical approach to construct a bias-free VQ is to use the optimal "dual" VQ (or Delaunay quantization) (Pagès and Wilbertz 2018), but this approach is numerically expensive (see Section 5.3). To mitigate the bias, we rather use random codebooks.

(b) Random Codebook. A key ingredient of Stovoq is the use of a random codebook within the quantizer. We assume $\mathscr{C}_M = [C_1, \ldots, C_M]$ where *the codewords* $\{C_i\}_{i=1}^M$ are i.i.d. random vectors distributed according to p, the codeword distribution pdf. We denote $\mathscr{C}_M \sim p$ and use boldface to stress that $\mathscr{C}_M$ is random. When quantizing a sequence of vectors $\{x_t\}_{t=0}^\infty \subset \mathbb{R}^D$ we sample for each $t \in \mathbb{N}$ a **new codebook** $\mathscr{C}_{M,t} \sim p$, compute $\mathrm{VQ}(x, \mathscr{C}_{M,t})$ and transmit the index of the corresponding codeword $i_{c,t} \in [M]$. The codebook $\mathscr{C}_{M,t}$ is **not transmitted**: the transmitter and the receiver use the **same seeds** so that the same codebooks $\mathscr{C}_{M,t}$ are generated on both sides.

(c) Unitary invariant Codewords. Denote by $\mathrm{U}(D) = \{U, U^*U = \mathrm{I}\}$ the set of unitary transforms over $\mathbb{R}^D$. We assume in the sequel that the codeword distribution p is unitary invariant, meaning that:

A9. *The distribution of the codewords p is invariant under the unitary group, i.e. for all $U \in \mathrm{U}(D)$, and any $x \in \mathbb{R}^D$, $p(Ux) = p(x)$.*

Examples of such distributions include isotropic Gaussian distributions ($p = \mathcal{N}(0, \sigma^2 \mathrm{I}_D)$, $\sigma^2 > 0$) and the uniform distribution on the Sphere (which is specifically discussed in Appendix B.4.1). Under **A**9, there exists a non-negative function p_{rad} on $\mathbb{R}_+$ such that, for all $x \in \mathbb{R}^d$, $p(x) = p_{\mathrm{rad}}(\|x\|)$.

(d) The quantization bias is radial. Under **A**9, we have the following crucial unitary invariance property. For $A \subset \mathbb{R}^D$, $U \in \mathrm{U}(D)$, we write $UA = \{Ux, x \in A\}$.

Lemma 6. *Assume **A**9. For any nonnegative measurable function f, any $U \in \mathrm{U}(D)$ and $x \in \mathbb{R}^D$, $\mathbb{E}_{\mathscr{C}_M \sim p}[f(\mathrm{VQ}(Ux, \mathscr{C}_M))] = \mathbb{E}_{\mathscr{C}_M \sim p}[f(U\,\mathrm{VQ}(x, \mathscr{C}_M))]$.*

The proof is postponed to Appendix B.2.3. Taking $f(x) = x$, the previous result implies that for any $x \in \mathbb{R}^D$ and $U \in \mathrm{U}(D)$, it holds that $\mathbb{E}_{\mathscr{C}_M \sim p}[\mathrm{VQ}(Ux, \mathscr{C}_M)] = U\mathbb{E}_{\mathscr{C}_M \sim p}[\mathrm{VQ}(x, \mathscr{C}_M)]$. A consequence is that the quantization error is radial (proof is given in Appendix B.2.4):

Theorem 7 (Quantization bias)**.** *Assume **A**9. Then, for all $M \in \mathbb{N}$, there exists a function $r_M^p : \mathbb{R}_+ \mapsto \mathbb{R}_+$ such that for all $x \in \mathbb{R}^D$, $\mathbb{E}_{\mathscr{C}_M \sim p}[\mathrm{VQ}(x, \mathscr{C}_M)] = r_M^p(\|x\|)x$.*

In words, the quantized vector $\mathrm{VQ}(x, \mathscr{C}_M)$ is **directionally unbiased** i.e., its expectation is *colinear* to the vector x. Moreover, this radial bias only depends on $\|x\|$, M and p. Consequently, in Stovoq, we can remove the bias of $\mathrm{VQ}(x, \mathscr{C}_M)$ by re-scaling the corresponding codeword by $1/r_M^p(\|x\|)$. We display r_M^p for $p = \mathcal{N}(0, \mathrm{I}_D)$ in Figure 5.1. Though r_M^p is not explicitely tractable, it is straightforward to pre-compute it using Monte-Carlo method, with arbitrary precision.

Further, we use a scalar quantizer SQ to transmit $1/r_M^p(\|x\|)$. Because the range of values taken by $1/r_M^p(\|x\|)$ is limited, a small number of bits P is sufficient (we typically use $P = 3$ bits). The total number of transmitted bits is $\log_2(M) + \log_2(P)$. We use a random unbiased scalar quantizer (see e.g. Dai et al. 2019, Eq. (2)), a random mapping for $\mathbb{R} \to \mathcal{S}_P$ an ordered subset of $\mathbb{R}$ with 2^P elements. A scalar quantizer is said to be unbiased if $\mathbb{E}[\mathrm{SQ}(r)] = r$ for all $r \in \mathbb{R}$. Assuming that SQ is independent of $\mathscr{C}_M$, we get for all $x \in \mathbb{R}^D$, $\mathbb{E}[\mathrm{SQ}(1/r_M^p(\|x\|))]\mathbb{E}_{\mathscr{C}_M \sim p}[\mathrm{VQ}(x, \mathscr{C}_M)] = x$. **Overall, Algorithm 6 is thus unbiased.** Details on scalar quantization are given in Appendix B.3.1.

5.3 Related work

We compare Stovoq with competing (random) compressors; additional details are given in Appendix B.2.1.

QSGD. Alistarh et al. 2017 compresses each coordinate of the scaled vector $x/\|x\|$ on $s+1$ codewords. QSGD is a scalar quantizer which requires $\mathcal{O}(\sqrt{D}\log_2(D))$ bits in its highest compression setting ($s = 1$, only two possible levels for each coordinate). The vector norm is transmitted with full precision $\|x\|$ (16 or 32 bits). This is in general substantially higher than the number of bits used by VQ methods. In deep learning problems, it reduces the communication cost by a factor of 4 to 7 (Alistarh et al. 2017, Sec. 5).

Top-H/Rand H. Achieving higher compression rates is possible through *sparsification* operators, that only transmit a few coordinates. The most popular schemes are Top-H and Rand-H compressors, that respectively map the vector to either its H largest coordinates, or a random subset of cardinality H, rescaled by D/H to ensure unbiasedness. Top-H is a biased operator, and the performance of Rand-H are poor on deep learning tasks (Beznosikov, Horváth, et al. 2020, Figures 4 and 5).

Low-rank. Vogels, Karimireddy, and Jaggi 2019 propose a low rank approximation of the gradients for deep learning. Although high compression factors ($\sim 100\times$) are obtained on some tasks, most methods are either biased (do not benefit from increasing the number of workers), or with a very large variance.

HyperSphere Quantization (HSQ). HSQ was introduced by Dai et al. 2019. Two versions are considered: (1) a - greedy- Voronoi VQ referred to as HSQ-greed in Table 5.1, which is biased, and for which the theoretical guarantee provided in the paper (in their Lemma 3 and Theorem 3, which corresponds to a variant of **A**8 and the subsequent convergence rate) *worsens* as M increases, making it mostly vacuous; (2) an unbiased version VQ (HSQ-span), which uses a minimum-norm decomposition of $x \in \mathrm{Span}(\mathscr{C}_M)$ the linear subspace generated by the codewords - this version suffers from a large variance (see Table 5.1) and potentially an ill-conditioning. Moreover, the performance of HSQ-span does not improve with M.

Stovoq builds on HSQ-greed, that achieves high compression factors (up to 60-100 to obtain close to SOTA performance on CIFAR10), while preserving a good flexibility w.r.t. the compression level. Stovoq approach allows to remove its inherent bias and obviously benefits from the number of workers n, as it is unbiased.

Table 5.1: Distortion for Gaussian inputs, for a fixed budget of 16 bits with $D = 16$.

Method	Sign	Top-2	Rand-2	Polytope	HSQ-span	HSQ-greed	Stovoq
# Bits (obj =16)	16	2×8	2×8	$\log_2(2\times 16)\times 2+6$	$\log_2(2^{10})+6$	$\log_2(2^{10})+6$	$\log_2(2^{13})+3$
Unbiased			✓	✓	✓		✓
$n=1$	6.4	8.7	110	121	147	9.1	11.0
$n=20$	6.4	8.5	5.4	5.9	7.2	8.8	0.53

Dual Quantization and Cross-polytope. An approach to constructing unbiased VQ is to use the dual VQ, also referred to as Delaunay Quantization (DQ); see Pagès and Wilbertz 2018. DQ is unbiased for any $x \in \text{ConvHull}(\mathscr{C}_M)$, the convex hull of $\mathscr{C}_M$. DQ requires to compute the barycentric coordinates for $x \in \text{ConvHull}(\mathscr{C}_M)$, that is to solve $(\lambda_1^x,\dots,\lambda_M^x) = \arg\min_{\lambda_1,\dots,\lambda_M} \|x - \sum_{i=1}^M \lambda_i c_i\|^2$, under the constraints $\lambda_i \geq 0, \sum_{i=1}^M \lambda_i = 1$. The quantizer is obtained by drawing a codeword c_i with probability $[\lambda_1^x,\dots,\lambda_M^x]$. Computing the barycentric coordinates is in general very demanding unless $\mathscr{C}_M$ has a very simple structure (see Appendix B.3 for details).

The `Cross-Polytope` method (Gandikota et al. 2021) is a simple instance of DQ, with a codebook $\mathscr{C}_{2D}^{\text{CP}}$ composed of the $2D$ canonical vectors $\left\{\pm\sqrt{D}e_i = \pm(0,\dots,0,\sqrt{D},0\dots 0), i\in[D]\right\}$, that relies on the inclusion $\text{B}_2(0;1) \subset \text{B}_1(0;\sqrt{D}) = \text{ConvHull}(\mathscr{C}_{2D}^{\text{CP}})$. The barycentric decomposition can then easily be computed. Unfortunately, this method suffers from a large variance, as the quantization error $\|\text{VQ}^{\text{CP}}(x,\mathscr{C}_M) - x\|$ of *any* x is *lower bounded* by $\sqrt{D}-1$, which means the error has the same quadratic error as the `Rand-1` compressor.

Other vector quantization schemes. Vogels, Karimireddy, and Jaggi 2019 propose `PowerSGD` and Yu et al. 2018 `GradiVeq`, which are also vector compression schemes. Yet, both schemes result in a biased compression, and thus do not fully benefit from an increase in the number of workers. Finally, they are supported by a "model" of the gradients to compress (low rank for `PowerSGD`, highly correlated for `GradiVeq`). On the other hand, our method does not make any such assumption on the gradients distribution. Lastly, `Atomo` (Hongyi Wang, Sievert, et al. 2018) relies on a form of Delaunay quantization. However, this requires to solve a meta-optimization problem at each step, resulting in substantial computational overhead.[3]

Numerical comparisons: In Table 5.1, we compare the average empirical distortions achieved by the compression methods given $|\mathcal{D}| = 10^4$ vectors in Table 5.1. For a communication budget of 16 bits for $D = 16$ and assuming that the input distribution is $q = \mathcal{N}(0, \text{I}_D)$. The compression factor is 32 (assuming 32 bits floating point per coordinate). Such a compression rate is out of reach for QSGD, that requires, even for $s = 1$ at least $\sqrt{D}\log(D) + R$ bits, where R is the number of bits to encode the norm (32 in Alistarh et al. 2017). For `QSGD` we have quantized the norm (using an uniform quantizer) on 3 bits and obtained an averaged distortion of 36.10 (for $n = 1$) and 1.82 for ($n = 20$) - the total number of bits is 19. We use $H = 2$ for `Top-`H and `Rand-`H and use a scalar quantizer with 8 bits. For `HSQ`, we use 6 bits for the norm, using the unbiased uniform quantizer given in Dai et al. 2019 and a Voronoi optized codebook for the uniform distribution on the unit-sphere with $M = 2^{10}$ codewords. For `Stovoq` we use a random codebook with $M = 2^{13}$ codewords, sampled from a $\mathcal{N}(0,(1+2/D)\text{I}_D)$, and 3 bits are allocated for the scalar quantization of $1/r_M^p$ (the inverse of the radial bias). Finally, we average the result of 2 independent compressions for `Polytope` (following the replication technique described in Gandikota et al. 2021). For `Stovoq` with $n = 20$, the codebooks of the different workers are

[3] Moreover, the best performance reported on test accuracy (Hongyi Wang, Sievert, et al. 2018, Fig. 3a), on CIFAR10 (with either ResNet-18 or VGG11) is below 80% (more than 10% below any of our results, with a weaker compression).

independent.

Overall, for $n = 1$, Stovoq achieves the best distortion among unbiased compressors[4], and is nearly on par with biased methods, and for $n = 20$, Stovoq strongly outperforms all methods.

5.4 Dostovoq algorithm

We now describe how the Stovoq compression scheme can be implemented in FL. As a *representative example*, we present the adaptation to Federated-SGD algorithm. At iteration $t+1$, the crucial steps are:

1. Worker $i \in [n]$ computes the norm $\|g_{i,t+1}\|$ of the $d \times 1$ gradient $g_{i,t+1}$ and then splits the scaled gradient $g_{i,t+1} \times \sqrt{d}/\|g_{i,t+1}\|$ into *L-buckets* of size D: $g_{i,t+1} \times \sqrt{d}/\|g_{i,t+1}\| = [b^1_{i,t+1}, \dots, b^L_{i,t+1}]$. The norm $\|g_{i,t+1}\|$ is transmitted to the central node using a high-resolution scalar quantizer.

2. Each worker quantizes the buckets $\{b^1_{i,t+1}, \dots, b^L_{i,t+1}\}$ using Stovoq. **Independent** codebooks $\{\mathscr{C}_{M,i,t+1}\}_{i\in[n]}$ are used to ensure that the quantizers remain conditionally independent. The double stochasticity (each worker uses random codebooks, independent between workers and across iterations) motivates the name Dostovoq. At iteration t, the same codebook is used for all buckets of worker i. Formally, for $\ell \in [L]$ we apply (in parallel) $\text{Stovoq}(b^\ell_{i,t+1}, p, M, P, s_{i,t+1})$, with a sequence of different seeds $(s_{i,t+1})_{i\in[n],t\geq 0}$. This sequence is shared between the workers and the central node at initialization.

3. The central node computes $(\hat{g}_{i,t+1})_{i\in n}$ from all messages received, performs the update on w_t, and broadcasts w_{t+1} to the workers.

Algorithme 7 : Dostovoq-SGD over T iterations

Input : T nb of steps, $(\eta_t)_{t\geq 0}$ LR, w_0, p, M, P ;
Output : $(w_t)_{t\geq 0}$

for $t = 1, \dots, T$ **do**
  $worker_0$ sends w_{t-1} and different seeds $s_{i,t}$ to all w_i;
  for $i = 1, \dots, n$ **do**
    Compute local gradient $g_{i,t}$ at w_{t-1};
    Split $g_{i,t} \times \sqrt{d}/\|g_{i,t}\|$ on $[b^1_{i,t}, \dots, b^L_{i,t}]$;
    for $\ell = 1, \dots, L$ *(in parallel)* **do**
      $(\mathbf{i}_c^{t,i,\ell}, \mathbf{i}_r^{t,i,\ell}) = \text{Stovoq}(b^\ell_{i,t}, p, M, P, s_{k,t})$
    end
    Send $(\|g_{i,t}\|, (\mathbf{i}_c^{t,i,\ell}, \mathbf{i}_r^{t,i,\ell})_{\ell\in[L]})$ to CS ;
  end
  Reconstruct $(\hat{g}_{i,t})_{i\in n}$;
  Update: $w_t = w_{t-1} - \eta_t \frac{1}{n} \sum_{i=1}^n \hat{g}_{i,t}$;
end

[4]The directionally unbiased version of Stovoq (coined GRVQ in Appendix B.5, e.g. in Table B.9) achieves the lowest distortion of 6.8 for $n = 1$.

Table 5.2: Average accuracy over 5 experiments, after 100 epochs on CIFAR-10. RBB = Raw bits per bucket; ECF = Effective compression factor.

Alg.	SGD	QSGD 2 bits	QSGD 4 bits	QSGD 8 bits	HSQ $d=16$	HSQ $d=8$	Dos. $d=16$	Dos. $d=8$
RBB	$32d$	$\sqrt{d}\log(d)$			$\log(d)$			
ECF	1	~ 13	~ 8	~ 4	34	17	38	20
$n=1$	91.9	91.7	92.1	91.9	92.0	92.0	92.0	92.1
$n=8$	92.0	91.8	91.8	92.0	91.8	92.0	91.8	92.1

Table 5.3: Distortion on a subset $\mathcal{G}$ of gradients of a layer of CIFAR-10, for a budget of 16 bits with $D=16$.

Alg.	Top-2	Rand-2	Polytope	HSQ-span	HSQ-greed	Dostovoq
RBB	2×8	2×8	$2\log_2(2D)+6$	$\log_2(2^{10})+6$		$\log_2(2^{13})+3$
Unbiased		✓	✓	✓		✓
$n=1$	0.51	5.66	7.6	7.74	0.59	0.42
$n=8$	0.51	0.76	0.97	0.97	0.49	0.053

5.5 Numerical experiments

Least Squares Regression (LSR) We consider a least-squares problem with $|\mathcal{D}| = 2^{14}$ samples, a bucket size $D = 16$, $d = 2^9$, and $n = 32$ workers; each worker has access to a subset $b = 2^{11}$ samples (picked with replacement) to introduce a dependency in the data used by the workers. For $j \in [|\mathcal{D}|]$, we assume $X_j \sim \mathcal{N}(0, \mathrm{I}_d)$ and $Y_j \sim \mathcal{N}(X_j^\top \omega_*, 1)$ where $\omega_* \in \mathbb{R}^D$. We solve $\inf_{\omega\in\mathbb{R}^d} \sum_{j=1}^{|} \mathcal{D}|\|Y_j - X_j^\top \omega\|^2$ via a gradient descent with step size $1/\alpha L$ where α is fine-tuned for each quantization method and $L \approx 2|\mathcal{D}|$ is the smoothness constant. We use Dostovoq with $M = 2^{13}$ codewords sampled from $\mathcal{N}(0, (1+2/D)\mathrm{I}_d)$ for Dostovoq and $M = 2^{10}$ on the unit Sphere for HSQ s.t. the number of bits transmitted at each round by the worker is set to 16 (see Table 5.1). Figure 5.2 reports the excess-log of the train loss over $T = 10$ iterations, for a standard GD. Dostovoq outperforms HSQ-greed: indeed the linear convergence rate of distributed GD is faster for an unbiased compressor than for the biased approach.

Applications to Deep Neural Networks. We now describe our experimental framework for training two standard models of Deep Neural Networks: a VGG-16 (Simonyan and Zisserman 2014) and a ResNet-18 (He et al. 2016). We follow the standard procedure of training those models both on CIFAR-10 and ImageNet; the hyper-parameters are fine-tuned to *optimize the accuracy without quantization*. We do not compress the affine constant part of the affine convolutional layers and batch normalization layers. We apply independent Dostovoq on batches of 32 buckets of size $D = 16$ (i.e. we transmit a high-resolution norm for $d = 32 \cdot 16 = 512$ coefficients).

Distortion: In Table 5.3 we report the distortion of a random subset of gradients $\mathcal{G} = \{g_t, t \in [|\mathcal{G}|]\}$ (with $|\mathcal{G}| = 10^2$, $D = 16$, $d = 2^5 \times D$) obtained from a given layer of a VGG on CIFAR-10, i.e.: $|\mathcal{G}|^{-1} \sum_{g_t\in\mathcal{G}} \left\| n^{-1} \sum_{i=1}^n \hat{g}_{i,t} - g_t \right\|^2$, where $(\hat{g}_{i,t})_{i\in[n]}$ correspond to i independent workers compressing a stochastic gradient g_t. The choice of the layer does not affect significantly the results. Even

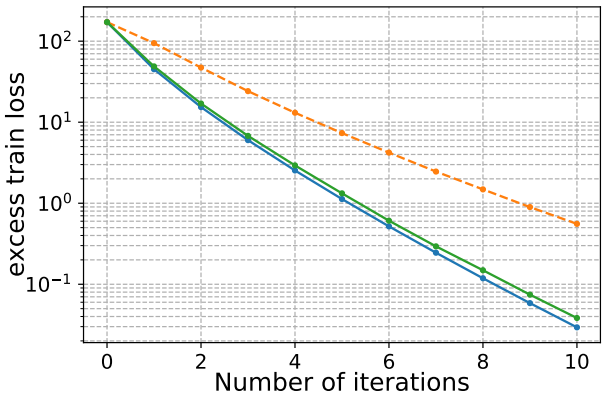


Figure 5.2: Comparison between GD (blue), HSQ-greed (orange) and Dostovoq (green), on a LSR problem in dimension $d = 2^9$.

with the actual gradient distribution, Dostovoq outperforms for a given compression factor each unbiased method. This is on pair with the observation that the gradients of a Deep Neural Network are approximately Gaussian distributed (Banner et al. 2018; A. Xu, Huo, and H. Huang 2020; Bernstein et al. 2018). Additional experiments (e.g., gradients sampled from LSR) can be found in Appendices B.5.1 to B.5.2.

While distortion for a fixed bit-budget is a relevant metric to compare compression[5], for completeness, we also compare on end-to-end training.

Accuracy on CIFAR-10. We use the implementation of HSQ (Dai et al. 2019): the batch size is 256 for CIFAR-10, the total number of epochs is 100, the initial learning rate is 0.1, which is divided by 10 and 50 at epochs 51 and 71. We report the accuracy of Dostovoq, QSGD, and HSQ-greed in Table 5.2. By design, the compression factor of Q-SGD for $D = 16$ is 13, which is significantly less than HSQ or Dostovoq. Both HSQ and Dostovoq perform similarly and the accuracy gap between the two methods are under the sample variance (computed over 5 seeds, about 0.2).

We provide results in Appendix B.1.2 for various other scenarios, especially 1) with a different network architecture, 2) comparing to more methods, showing that Dostovoq outperforms Top-k and Cross-Polytope, 3) with strong distribution heterogeneity (using Dostovoq-DIANA), 4) with Error Feedback (gaining 1%). Furthermore, we also report computational and memory overheads, give a sensibility analysis to the choice of D, L and M, and discuss implementation challenges in Appendix B.1.2.

Accuracy on ImageNet. For ImageNet, we use different bucket sizes, the standard batch size of 256, and only $n = 1$ worker for energy savings (recall Imagenet training last about 1 day for a single worker on academic hardware). An initial learning rate of 0.1 is divided by 10 at epoch 30 and 60, while the model is trained for 90 epochs. A ResNet here obtains 69.9%, and with a compression factor of 8, the performance drops by 2.5%. Using $D = 16$, we reach a compression factor of 38, while the Top-1 accuracy drops by only 4.8%: this is a substantially higher compression rate than the concurrent work QSGD on the ImageNet dataset. We stress that

[5] "aggregated" metrics (performance of advanced algorithms, with specific hyperparameters – batch, learning rate schedule, etc.), may not reflect the quality of a compression scheme, and highly depend on amount of tuning.

we performed no parameter tuning for the Dostovoq run, and used exactly the same parameters than the ones optimized for SGD.

5.6 Conclusion

In this chapter, we propose and analyze a compression technique, relying on unitarily invariant random codebooks. Our focus is thus on analyzing its properties: we demonstrate the performance of this compression technique, *focusing on a simple but reliable metric*, the distortion on the compressed vectors, that we carefully analyze and evaluate experimentally on various sources of vectors.

We also describe how Stovoq can be integrated within (any) Federated Learning algorithm and demonstrate that we can leverage many of the convergence guarantees provided in the literature.

Chapter 6

FL with different node computation capacities, and a central clock

In the previous Chapters 4 and 5 we have presented solutions to work with limited nodes and a limited bandwidth in the FL context. In the next chapters, we consider the asynchronous FL setting: we *do not* assume anymore that nodes contribute to the CS at the same pace. In particular, nodes can have very different computational speeds and/or a very different access to the bandwidth. This creates a serious bottleneck: in practical scenari, one does not want to wait for the slowest node at every CS update.

This chapter details our first solution to work with asynchronous nodes: FAVANO. The CS follow a fixed clock for its updates, and it can interrupt/trigger any node at any time. This work (Leconte, V. M. Nguyen, and Moulines 2023) has been accepted for publication at the ICASSP2024 conference:

Louis Leconte, Van Minh Nguyen, and Eric Moulines (2023). "FAVAS: Federated AVeraging with ASynchronous clients". In: arXiv preprint arXiv:2305.16099.

6.1 Introduction

Federated learning, a promising approach for training models from networked agents, involves the collaborative aggregation of locally computed updates, such as parameters, under centralized orchestration (Konečnỳ, B. McMahan, and Ramage 2015; B. McMahan et al. 2017; Kairouz, H. B. McMahan, et al. 2021). The primary motivation behind this approach is to maintain privacy, as local data is never shared between agents and the central server (Y. Zhao et al. 2018; Horváth, Sanjabi, et al. 2022). However, communication of training information between edge devices and the server is still necessary. The central server aggregates the local models to update the global model, which is then sent back to the devices. Federated learning helps alleviate privacy concerns, and it distributes the computational load among networked agents. However, each agent must have more computational power than is required for inference, leading to a computational power bottleneck. This bottleneck is especially important when federated learning is used in heterogeneous, cross-device applications.

Most approaches to centralized federated learning (FL) rely on synchronous operations, as assumed in many studies (B. McMahan et al. 2017; J. Wang, Charles, et al. 2021). At each global iteration, a copy of the current model is sent from the central server to a selected subset of agents. The agents then update their model parameters using their private data and send the model updates back to the server. The server aggregates these updates to create a new shared model, and this process is repeated until the shared model meets a desired criterion. However, device heterogeneity and communication bottlenecks (such as latency and bandwidth) can cause delays, message loss, and stragglers, and the agents selected in each round must wait for the slowest one before starting the next round of computation. This waiting time can be significant, especially since nodes may have different computation speeds.

To address this challenge, researchers have proposed several approaches that enable asynchronous communication, resulting in improved scalability of distributed/federated learning (Xie, Koyejo, and I. Gupta 2019; Y. Chen et al. 2020; Z. Chen et al. 2021; C. Xu et al. 2021). In this case, the central server and local agents typically operate with inconsistent versions of the shared model, and synchronization in lockstep is not required, even between participants in the same round. As a result, the server can start aggregating client updates as soon as they are available, reducing training time and improving scalability in practice and theory.

Our proposed algorithm `FAVANO` is designed to allow clients to perform their local steps independently of the server's round structure, using a fully local, possibly outdated version of the model. Upon entering the computation, all clients are given a copy of the global model and perform at most $K \geq 1$ optimization steps based on their local data. The server randomly selects a group of s clients in each server round, which, upon receiving the server's request, submit an *unbiased* version of their progress. Although they may still be in the middle of the local optimization process, they send reweighted contributions so that fast and slow clients contribute equally. The central server then aggregates the models and sends selected clients a copy of the current model. The clients take this received server model as a new starting point for their next local iteration.

6.2 Related Works

Federated Averaging (FedAvg), also known as local SGD, is a widely used approach in federated learning. In this method, each client updates its local model using multiple steps of stochastic gradient descent (SGD) to optimize a local objective function. The local devices then submit their model updates to the central server for aggregation, and the server updates its own model

parameters by averaging the client models before sending the updated server parameters to all clients. FedAvg has been shown to achieve high communication efficiency with infrequent synchronization, outperforming distributed large mini-batches SGD (T. Lin et al. 2019).

However, the use of multiple local epochs in FedAvg can cause each device to converge to the optima of its local objective rather than the global objective, a phenomenon known as client drift. This problem has been discussed in previous work; see Karimireddy, Kale, et al. 2020. Most of these studies have focused on synchronous federated learning methods, which have a similar update structure to FedAvg (J. Wang, Qinghua Liu, et al. 2020; Karimireddy, Kale, et al. 2020; Qu, Song, and Tsui 2021; Makarenko et al. 2022; Y. Mao et al. 2022; Tyurin and Richtárik 2022). However, synchronous methods can be disadvantageous because they require all clients to wait when one or more clients suffer from high network delays or have more data, and require a longer training time. This results in idle time and wasted computing resources.

Moreover, as the number of nodes in a system increases, it becomes infeasible for the central server to perform synchronous rounds among all participants, and synchrony can degrade the performance of distributed learning. A simple approach to mitigate this problem is node sampling, e.g. Smith et al. 2017; Keith Bonawitz, Eichner, et al. 2019, where the server only communicates with a subset of the nodes in a round. But if the number of stragglers is large, the overall training process still suffers from delays.

Synchronous FL methods are prone to stragglers. One important research direction is based on FedAsync (Xie, Koyejo, and I. Gupta 2019) and subsequent works. The core idea is to update the global model immediately when the central server receives a local model. However, when staleness is important, performance is similar to FedAvg, so it is suboptimal in practice. ASO-Fed (Y. Chen et al. 2020) proposes to overcome this problem and handles asynchronous FL with local streaming data by introducing memory-terms on the local client side. AsyncFedED (Q. Wang et al. 2022) also relies on the FedAsync instantaneous update strategy and also proposes to dynamically adjust the learning rate and the number of local epochs to staleness. Only one local updated model is involved in FedAsync-like global model aggregations. As a result, a larger number of training epochs are required and the frequency of communication between the server and the workers increases greatly, resulting in massive bandwidth consumption. From a different perspective, QuAFL (Zakerinia et al. 2022) develops a concurrent algorithm that is closer to the FedAvg strategy. QuAFL incorporates both asynchronous and compressed communication with convergence guarantees. Each client must compute K local steps and can be interrupted by the central server at any time. The client updates its model with the (compressed) central version and its current private model. The central server randomly selects s clients and updates the model with the (compressed) received local progress (since last contact) and the previous central model. QuAFL works with old variants of the model at each step, which slows convergence. However, when time, rather than the number of server rounds, is taken into account, QuAFL can provide a speedup because the asynchronous framework does not suffer from delays caused by stragglers. A concurrent and asynchronous approach aggregates local updates before updating the global model: FedBuff (J. Nguyen et al. 2022) addresses asynchrony using a buffer on the server side. Clients perform local iterations, and the base station updates the global model only after Z different clients have completed and sent their local updates. The gradients computed on the client side may be stale. The main assumption is that the client computations completed at each step come from a uniform distribution across all clients. Fedbuff is asynchronous, but is also sensitive to stragglers (must wait until Z different clients have done all local updates). Similarly, Koloskova, Sebastian U Stich, and Jaggi 2022 focus on Asynchronous SGD, and provide guarantees depending on some τ_{max}. Similar to J. Nguyen et al. 2022 the algorithm is also impacted by stragglers, during the transitional regime at least. A recent work by Fraboni et al. 2023 extend the idea of Koloskova, Sebastian U Stich, and Jaggi

2022 by allowing multiple clients to contribute in one round. But this scheme also favors fast clients. J. Liu et al. 2021 does not run on buffers, but develops an Adaptive Asynchronous Federated Learning (AAFL) mechanism to deal with speed differences between local devices. Similar to FedBuff, in J. Liu et al. 2021's method, only a certain fraction of the locally updated models contribute to the global model update. Most convergence guarantees for asynchronous distributed methods depend on staleness or gradient delays (J. Nguyen et al. 2022; Toghani and Uribe 2022; Koloskova, Sebastian U Stich, and Jaggi 2022). Only Mishchenko, F. Bach, et al. 2022 analyzes the asynchronous stochastic gradient descent (SGD) independently of the delays in the gradients. However, in the heterogeneous (non-IID) setting, convergence is proved up to an additive term that depends on the dissimilarity limit between the gradients of the local and global objective functions.

6.3 Algorithm

We consider optimization problems in which the components of the objective function (i.e., the data for machine learning problems) are distributed over n clients, i.e.,

$$\min_{w\in\mathbb{R}^d} f(w);\ f(w) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}_{(x,y)\sim D_i^{\text{data}}}[\ell(\text{nn}(x,w),y)], \tag{6.1}$$

where d is the number of parameters (network weights and biases), n is the total number of clients, ℓ is the training loss (e.g., cross-entropy or quadratic loss), $\text{nn}(x,w)$ is the DNN prediction function, D_i^{data} is the training distribution on client i. In FL, the distributions D_i^{data} are allowed to differ between clients (statistical heterogeneity).

Each client maintains three key values in its local memory: the local model w^i, a counter q^i, and the value of the initial model with which it started the iterations w^i_{init}. The counter q^i is incremented for each SGD step the client performs locally until it reaches K, at which point the client stops updating its local model and waits for the server request. Upon the request to the client i, the local model and counter q^i are reset. If a server request occurs before the K local steps are completed, the client simply pauses its current training process, reweights its gradient based on the number of local epochs (denoted by E^i_{t+1}), and sends its current *reweighted* model to the server.

In Zakerinia et al. 2022, we identified the client update $w^i = \frac{1}{s+1}w_{t-1} + \frac{s}{s+1}w^i$ as a major shortcoming. When the number of sampled clients s is large enough, $\frac{s}{s+1}w^i$ dominates the update and basically no server term are taken into consideration. This leads to a significant client drift. As a consequence, QuAFL does not perform well in the heterogeneous case (see Section 6.5). Second, one can note that the updates in QuAFL are biased in favor of fast clients. Indeed each client computes gradients at its own pace and can reach different numbers of epochs while being contacted by the central server. It is assumed that clients compute the *same* number of local epochs in the analysis from Zakerinia et al. 2022, but it is not the case in practice. As a consequence, we propose `FAVANO` to deal with asynchronous updates without favoring fast clients. A first improvement is to update local weight directly with the received central model. Details can be found in Algorithm 8. Another idea to tackle gradient unbiasedness is to reweight the contributions from each of the s selected clients: these can be done either by dividing by the (proper) number of locally computed epochs, or by the expected value of locally computed epochs. In practice, we define the reweight $\alpha^i = \mathbb{E}[E^i_{t+1} \wedge K]$, or $\alpha^i = \mathbf{P}(E^i_{t+1} > 0)\cdot(E^i_{t+1} \wedge K)$, where $\wedge$ stands for min. We assume that the server performs a number of training epochs $T \geq 1$. At each time step $t \in \{1,\dots,T\}$, the server has a model w_t. At initialization, the central server

Algorithme 8 : FAVANO over T iterations. In red are highlighted the differences with QuAFL.

Input : Number of steps T, LR η, Selection Size s, Maximum local steps K ;
/* At the Central Server */
Initialize
  Initialize parameters w_0;
  Server sends w_0 to all clients;
end
for $t = 1, \dots, T$ **do**
  Generate set $\mathcal{S}_t$ of s clients uniformly at random;
  for *all clients* $i \in \mathcal{S}_t$ **do**
    Server receives $w^i_{unbiased}$ from client i;
  end
  Update central server model $w_t \leftarrow \frac{1}{s+1} w_{t-1} + (\frac{1}{s+1} \sum_{i \in \mathcal{S}_t} w^i_{unbiased})$;
  for *all clients* $i \in \mathcal{S}_t$ **do**
    Server sends w_t to client i;
  end
end
/* At Client i */
Initialize
  Client receives w_0 and K from the Server;
  Local variables $w^i = w_0, q^i = 0$;
end
Loop
  Run ClientLocalTraining() concurrently;
  When *Contacted by the Server* **do**
    Interrupt ClientLocalTraining();
    Define α^i following the reweighting strategy ;
    Send $w^i_{unbiased} := w^i_{init} + \frac{1}{\alpha^i}(w^i - w^i_{init})$ to the server;
    Receive w_t from the server;
    Update $w^i_{init} \leftarrow w_t, w^i \leftarrow w_t, q^i \leftarrow 0$;
    Restart ClientLocalTraining() from zero with updated variables;
  end
end
function ClientLocalTraining():
  while $q^i < K$ **do**
    Compute local stochastic gradient $\widetilde{g}^i$ at w^i;
    Update local model $w^i \leftarrow w^i - \eta \widetilde{g}^i$;
    Update local counter $q^i \leftarrow q^i + 1$;
  end
  Wait();
end function

transmits identical parameters w_0 to all devices. At each time step t, the central server selects a subset $\mathcal{S}_t$ of s clients uniformly at random and requests their local models. Then, the requested

clients submit their *reweighted* local models back to the server. When all requested models arrive at the server, the server model is updated based on a simple average (see Line 10). Finally, the server multicasts the updated server model to all clients in $\mathcal{S}_t$. In particular, all clients $i \notin \mathcal{S}_t$ continue to run their individual processes without interruption.

Remark 8. *In FAVANO's setting, we assume that each client $i \in \{1,...,n\}$ keeps a full-precision local model w^i. In order to reduce the computational cost induced by the training process, FAVANO can also be implemented with a quantization function Q. First, each client computes backpropagation with respect to its quantized weights $Q(w^i)$. That is, the stochastic gradients are unbiased estimates of $\nabla f_i\left(Q\left(w^i\right)\right)$. Moreover, the activations computed at forward propagation are quantized. Finally, the stochastic gradient obtained at backpropagation is quantized before the SGD update. In our supplementary experiments, we use the logarithmic unbiased quantization method of Chmiel et al. 2021.*

6.4 Analysis

In this section we provide complexity bounds for FAVANO in a smooth nonconvex environment. We introduce an abstraction to model the stochastic optimization process and prove convergence guarantees for FAVANO.

6.4.1 Preliminaries.

We abstract the optimization process to simplify the analysis. In the proposed algorithm, each client asynchronously computes its own local updates without taking into account the server time step t. Here in the analysis, we introduce a different, but statistically equivalent setting. At the beginning of each server timestep t, each client maintains a local model w^i_{t-1}. We then assume that all n clients *instantaneously* compute local steps from SGD. The update in local step q for a client i is given by:

$$\widetilde{h}^i_{t,q} = \widetilde{g}^i\left(w^i_{t-1} - \sum_{s=1}^{q-1} \eta \widetilde{h}^i_{t,s}\right), \tag{6.2}$$

where $\widetilde{g}^i$ represents the stochastic gradient that client i computes for the function f_i. We also define n independent random variables $E^1_t, \dots, E^n_t$ in $\mathbb{N}$. Each random variable E^i_t models the number of local steps the client i could take before receiving the server request. We then introduce the following random variable: $\widetilde{h}^i_t = \sum_{q=1}^{E^i_t} \widetilde{h}^i_{t,q}$. Compared to Zakerinia et al. 2022, we do not assume that clients performed the same number of local epochs. Instead, we reweight the sum of the gradients by weights α^i, which can be either *stochastic* or *deterministic*:

$$\alpha^i = \begin{cases} \mathbf{P}(E^i_{t+1} > 0)(E^i_{t+1} \wedge K) & \textit{stochastic} \text{ version}, \\ \mathbb{E}[E^i_{t+1} \wedge K] & \textit{deterministic} \text{ version}. \end{cases} \tag{6.3}$$

And we can define the *unbiased* gradient estimator: $\check{h}^i_t = \frac{1}{\alpha^i} \sum_{q=1}^{E^i_t \wedge K} \widetilde{h}^i_{t,q}$.

Finally, a subset $\mathcal{S}_t$ of s clients is chosen uniformly at random. This subset corresponds to the clients that send their models to the server at time step t. In the current notation, each client $i \in \mathcal{S}_t$ sends the value $w^i_{t-1} - \eta \check{h}^i_t$ to the server. We emphasise that in our abstraction, all clients compute E^i_t local updates. However, only the clients in $\mathcal{S}_t$ send their updates to the server, and each client $i \in \mathcal{S}_t$ sends only the K first updates. As a result, we introduce the following update

Table 6.1: How long one has to wait to reach an ϵ accuracy for non-convex functions. For simplicity, we ignore all constant terms. Each constant $C\_$ depends on client speeds and represents the unit of time one has to wait in between two consecutive server steps. L is the Lipschitz constant, and $F := (f(w_0) - f_*)$ is the initial conditions term. a_i, b are constants depending on client speeds statistics, and defined in Theorem 10.

Method	Units of time
FedAvg	$\left(\frac{FL\sigma^2+(1-\frac{s}{n})KG^2}{sK}\epsilon^{-2} + FL^{\frac{1}{2}}G\epsilon^{-\frac{3}{2}} + LFB^2\epsilon^{-1}\right)C_{FedAvg}$
FedBuff	$\left(FL(\sigma^2+G^2)\epsilon^{-2} + FL((\frac{\tau_{max}^2}{s^2}+1)(\sigma^2+nG^2))^{\frac{1}{2}}\epsilon^{-\frac{3}{2}} + FL\epsilon^{-1}\right)C_{FedBuff}$
AsyncSGD	$\left(FL(3\sigma^2+4G^2)\epsilon^{-2} + FLG(s\tau_{avg})^{\frac{1}{2}}\epsilon^{-\frac{3}{2}} + (s\tau_{max}F)^{\frac{1}{2}}\epsilon^{-1}\right)C_{AsyncSGD}$
QuAFL	$\frac{1}{E^2}FLK(\sigma^2+2KG^2)\epsilon^{-2} + \frac{n\sqrt{n}}{E\sqrt{Es}}FKL(\sigma^2+2KG^2)^{\frac{1}{2}}\epsilon^{-\frac{3}{2}} + \frac{1}{E\sqrt{s}}n\sqrt{n}FBK^2L\epsilon^{-1}$
FAVANO	$FL(\sigma^2\sum_i^n \frac{a_i}{n} + 8G^2b)\epsilon^{-2} + \frac{n}{s}FL^2(K^2\sigma^2 + L^2K^2G^2 + s^2\sigma^2\sum_i^n\frac{a_i}{n} + s^2G^2b)^{\frac{1}{2}}\epsilon^{-\frac{3}{2}} + nFB^2KLb\epsilon^{-1}$

equations:

$$\begin{cases} w_t = \frac{1}{s+1}w_{t-1} + \frac{1}{s+1}\sum_{i\in\mathcal{S}_t}(w_{t-1}^i - \eta\frac{1}{\alpha^i}\sum_{s=1}^{E_t^i\wedge K}\widetilde{h}_{t,s}^i), \\ w_t^i = w_t, \quad \text{for } i \in \mathcal{S}_t, \\ w_t^i = w_{t-1}^i, \quad \text{for } i \notin \mathcal{S}_t. \end{cases} \tag{6.4}$$

Assumptions and notations.

A10. *Uniform Lower Bound: There exists $f_* \in \mathbb{R}$ such that $f(w) \geq f_*$ for all $w \in \mathbb{R}^d$.*

A11. *Smooth Gradients: For any client i, the gradient ∇f_i is L-Lipschitz continuous for some $L > 0$, i.e. for all $w, v \in \mathbb{R}^d$: $\|\nabla f_i(w) - \nabla f_i(v)\| \leq L\|w - v\|$.*

A12. *Bounded Variance: For any client i, the variance of the stochastic gradients is bounded by some $\sigma^2 > 0$, i.e. for all $w \in \mathbb{R}^d$: $\mathbb{E}[\left\|\widetilde{g}^i(w) - \nabla f_i(w)\right\|^2] \leq \sigma^2$.*

A13. *Bounded Gradient Dissimilarity: There exist constants $G^2 \geq 0$ and $B^2 \geq 1$, such that for all $w \in \mathbb{R}^d$: $\sum_{i=1}^n \frac{\|\nabla f_i(w)\|^2}{n} \leq G^2 + B^2\|\nabla f(w)\|^2$.*

We define the notations required for the analysis. Consider a time step t, a client i, and a local step q. We define

$$\mu_t = \left(w_t + \sum_{i=1}^n w_t^i\right)/(n+1) \tag{6.5}$$

the average over all node models in the system at a given time t. The first step of the proof is to compute a preliminary upper bound on the divergence between the local models and their average. For this purpose, we introduce the Lyapunov function: $\Phi_t = \left\|w_t - \mu_t\right\|^2 + \sum_{i=1}^n \left\|w_t^i - \mu_t\right\|^2$.

Upper bounding the expected change in potential. A key result from our analysis is to upper bound the change (in expectation) of the aforementioned potential function Φ_t:

Lemma 9. *For any time step $t > 0$ we have:*

$$\mathbb{E}[\Phi_{t+1}] \leq (1-\kappa)\mathbb{E}[\Phi_t] + 3\frac{s^2}{n}\eta^2\sum_{i=1}^n \mathbb{E}\left\|\check{h}_{t+1}^i\right\|^2, \tag{6.6}$$

with $\kappa = \frac{1}{n}\left(\frac{s(n-s)}{2(n+1)(s+1)}\right)$.

The intuition behind Lemma 9 is that the potential function Φ_t remains concentrated around its mean, apart from deviations induced by the local gradient steps. The full analysis involves many steps and we refer the reader to Appendix C.1 for complete proofs. In particular, Lemmas 44 and 46 allow us to examine the scalar product between the expected node progress $\sum_{i=1}^n \check{h}_t^i$ and the true gradient evaluated on the mean model $\nabla f(\mu_t)$. The next theorem allows us to compute an upper-bound on the averaged norm-squared of the gradient, a standard quantity studied in non-convex stochastic optimization.

6.4.2 Convergence results

The following statement shows that FAVANO algorithm converges towards a first-order stationary point, as T the number of global epochs grows.

Theorem 10. *Assume **A**10 to **A**13 and assume that the learning rate η satisfies $\eta \leq \frac{1}{20B^2bKLs}$. Then FAVANO converges at rate:*

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla f(\mu_t)\right\|^2 \leq \frac{2(n+1)F}{Ts\eta} + \frac{Ls}{n+1}\left(\frac{\sigma^2}{n}\sum_i^n a^i + 8G^2b\right)\eta + L^2s^2\left(\frac{720\sigma^2}{n}\sum_i^n a^i + 5600bG^2\right)\eta^2, \tag{6.7}$$

with $F := (f(\mu_0) - f_*)$, *and*

$$\begin{cases} a^i = \frac{1}{\mathbf{P}(E_{t+1}^i>0)^2}\left(\frac{\mathbf{P}(E_{t+1}^i>0)}{K^2} + \mathbb{E}\left[\frac{\mathbb{1}(E_{t+1}^i>0)}{E_{t+1}^i \wedge K}\right]\right), \\ b = \max_i\left(\frac{1}{\mathbf{P}(E_{t+1}^i>0)}\right), \end{cases} \tag{6.8}$$

for $\alpha^i = \mathbf{P}(E_{t+1}^i > 0)(E_{t+1}^i \wedge K)$, *or*

$$\begin{cases} a^i = \frac{1}{\mathbb{E}[E_{t+1}^i \wedge K]} + \frac{\mathbb{E}[(E_{+1}^i \wedge K)^2]}{K^2\mathbb{E}[E_{t+1}^i \wedge K]}, \\ b = \max_i\left(\frac{\mathbb{E}[(E_{t+1}^i \wedge K)^2]}{\mathbb{E}[E_{t+1}^i \wedge K]}\right), \end{cases} \tag{6.9}$$

for $\alpha^i = \mathbb{E}[E_{t+1}^i \wedge K]$.

Note that the previous convergence result refers to the average model μ_t. In practice, this does not pose much of a problem. After training is complete, the server can ask each client to submit its final model. It should be noted that each client communicates $\frac{sT}{n}$ times with the server during training. Therefore, an additional round of data exchange represents only a small increase in the total amount of data transmitted.

The bound in Theorem 10 contains 3 terms. The first term is standard for a general non-convex target and expresses how initialization affects convergence. The second and third terms depend on the statistical heterogeneity of the client distributions and the fluctuation of the minibatch gradients. Table 6.1 compares complexity bounds along with synchronous and asynchronous methods. One can note the importance of the ratio $\frac{s}{n}$. Compared to J. Nguyen et al. 2022 or Koloskova, Sebastian U Stich, and Jaggi 2022, FAVANO can potentially suffer from delayed updates when $\frac{s}{n} \ll 1$, but FAVANO does *not* favor fast clients at all. In practice, it is not a major shortcoming, and FAVANO is more robust to fast/slow clients distribution than FedBuff/AsyncSGD (see Figure 6.2). We emphasize both FedBuff and AsyncSGD rely on strong assumptions: neither the queuing process, nor the transitional regime are taken into account in their analysis. In practice, during the first iterations, only fast clients contribute. It induces a serious bias. Our experiments indicate that a huge amount of server iterations has

to be accomplished to reach the stationary regime. Still, under this regime, slow clients are contributing with delayed information. J. Nguyen et al. 2022; Koloskova, Sebastian U Stich, and Jaggi 2022 propose to uniformly bound this delay by some quantity τ_{max}. We keep this notation while reporting complexity bounds in Table 6.1, but argue nothing guarantee τ_{max} is properly defined (i.e. finite). All analyses except that of Mishchenko, F. Bach, et al. 2022 show that the number of updates required to achieve accuracy grows linearly with τ_{max}, which can be very adverse. Specifically, suppose we have two parallel workers - a fast machine that takes only 1 unit of time to compute a stochastic gradient, and a slow machine that takes 1000 units of time. If we use these two machines to implement FedBuff/AsyncSGD, the gradient delay of the slow machine will be one thousand, because in the 1 unit of time we wait for the slow machine, the fast machine will produce one thousand updates. As a result, the analysis based on τ_{max} deteriorates by a factor of 1000.

In the literature, guarantees are most often expressed as a function of server steps. In the asynchronous case, this is *inappropriate* because a single step can take very different amounts of time depending on the method. For example, with FedAvg or Scaffold (Karimireddy, Kale, et al. 2020), one must wait for the slowest client for each individual server step. Therefore, we introduce in Table 6.1 constants $C_{\_}$ that depend on the client speed and represent the unit of time to wait between two consecutive server steps. Finally, optimizing the value of the learning rate η with Lemma 40 yields the following:

Corollary 11. *Assume **A**10 to **A**13. We can optimize the learning rate by Lemma 40 and* FAVANO *reaches an ϵ precision for a number of server steps T greater than (up to numerical constants):*

$$\frac{FL(\frac{\sigma^2}{n}\sum_i^n a^i + 8G^2 b)}{\epsilon^2} + \frac{FL^2(K^2\sigma^2 + L^2K^2G^2 + \frac{s^2\sigma^2}{n}\sum_i^n a^i + s^2G^2b)^{\frac{1}{2}}}{\epsilon^{\frac{3}{2}}s/n} + \frac{FB^2KLb}{\epsilon/n}, \tag{6.10}$$

where $F = (f(\mu_0) - f_*)$, and (a^i, b) are defined in Theorem 10.

The second term in Corollary 11 is better than the one from the QuAFL analysis (Zakerinia et al. 2022). Although this $(n+1)$ term can be suboptimal, note that it is only present at second order from ϵ and therefore becomes negligible when ϵ goes to 0 (Lu and De Sa 2020; Zakerinia et al. 2022).

Remark 12. *Our analysis can be extended to the case of quantized neural networks. The derived complexity bounds also hold for the case when the quantization function Q is biased. We make only a weak assumption about Q (we assume that there is a constant r_d such that for any $x \in \mathbb{R}^d$ $\|Q(x) - x\|^2 \leq r_d$), which holds for standard quantization methods such as stochastic rounding and deterministic rounding. The only effect of quantization would be increased variance in the stochastic gradients. We need to add to the upper bound given in Theorem 10 an "error floor" of $12L^2r_d$, which remains independent of the number of server epochs. For stochastic or deterministic rounding, $r_d = \Theta(d\frac{1}{2^{2b}})$, where b is the number of bits used. The error bound is the cost of using quantization as part of the optimization algorithm. Previous works with quantized models also include error bounds (H. Li et al. 2017; Zheng Li and Sa 2019).*

6.5 Numerical Results

We test FAVANO on three image classification tasks: MNIST (Li Deng 2012), CIFAR-10 (Krizhevsky, G. Hinton, et al. 2009), and TinyImageNet (Y. Le and X. Yang 2015). For the MNIST and CIFAR-10 datasets, two training sets are considered: an IID and a non-IIID split. In the first case, the training images are randomly distributed among the n clients. In the second case, each

Table 6.2: Final accuracy on the test set (average and standard deviation over 10 random experiments) for the MNIST classification task.

Methods	IID split	non-IID split ($\frac{2}{3}$ fast clients)	non-IID split ($\frac{1}{9}$ fast clients)
FedAvg	93.4 ± 0.3	38.7 ± 7.7	44.8 ± 6.9
QuAFL	92.3 ± 0.9	40.7 ± 6.7	45.5 ± 4.0
FedBuff	$\mathbf{96.0} \pm 0.1$	85.1 ± 3.2	67.3 ± 5.5
FAVANO	95.1 ± 0.1	$\mathbf{88.9} \pm 0.9$	$\mathbf{87.3} \pm 2.3$

client takes two classes (out of the ten possible) without replacement. This process leads to heterogeneity among the clients.

The standard evaluation measure for FL is the number of server rounds of communication to achieve target accuracy. However, the time spent between two consecutive server steps can be very different for asynchronous and synchronous methods. Therefore, we compare different synchronous and asynchronous methods w.r.t. *total simulation time* (see below). We also measured the loss and accuracy of the model in terms of server steps and total local client steps (see Appendix C.2.3). In all experiments, we track the performance of each algorithm by evaluating the server model against an unseen validation dataset. We present the test accuracy and variance, defined as $\sum_{i=1}^{n} \|w_t^i - w_t\|^2$.

We decide to focus on non-uniform timing experiments as in J. Nguyen et al. 2022, and we base our simulation environment on QuAFL's code[1]. After simulating n clients, we randomly group them into fast or slow nodes. We assume that at each time step t (for the central server), a set of s clients is randomly selected without replacement. We assume that the clients have different computational speeds, and refer to Appendix C.2.2 for more details. We assume that only one-third of the clients are slow, unless otherwise noted. We compare FAVANO with the classic synchronous approach FedAvg (B. McMahan et al. 2017) and two newer asynchronous methods QuAFL (Zakerinia et al. 2022) and FedBuff (J. Nguyen et al. 2022). Details on implementing other methods can be found in Appendix C.2.1.

We use the standard data augmentations and normalizations for all methods. FAVANO is implemented in Pytorch, and experiments are performed on an NVIDIA Tesla-P100 GPU. Standard multiclass cross entropy loss is used for all experiments. All models are fine-tuned with $n = 100$ clients, $K = 20$ local epochs, and a batch of size 128. Following the guidelines of J. Nguyen et al. 2022, the buffer size in FedBuff is set to $Z = 10$. In FedAvg, the total simulated time depends on the maximum number of local steps K and the slowest client runtime, so it is proportional to the number of local steps and the number of global steps. In QuAFL and FAVANO on the other hand, each global step has a predefined duration that depends on the central server clock. Therefore, the global steps have similar durations and the total simulated time is the sum of the durations of the global steps. In FedBuff, a global step requires filling a buffer of size Z. Consequently, both the duration of a global step and the total simulated time depend on Z and on the proportion of slow clients (see Appendix C.2.2 for a detailed discussion).

We first report the accuracy of a shallow neural network trained on MNIST. The learning rate is set to 0.5 and the total simulated time is set to 5000. We also compare the accuracy of a Resnet20 (He et al. 2016) with the CIFAR-10 dataset (Krizhevsky, G. Hinton, et al. 2009), which consists of 50000 training images and 10000 test images (in 10 classes). For CIFAR-10, the learning rate is set to 0.005 and the total simulation time is set to 10000. In Figure 6.1, we show

[1] https://github.com/ShayanTalaei/QuAFL

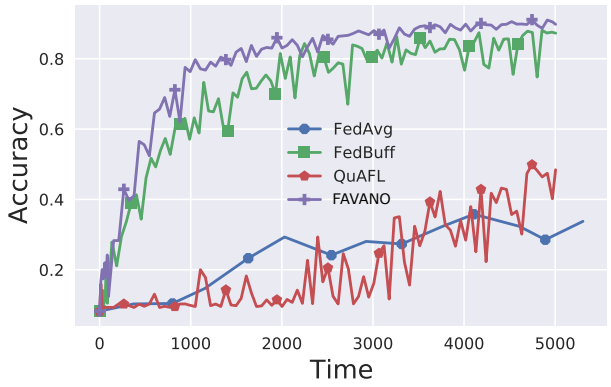


Figure 6.1: Test accuracy on the MNIST dataset with a non-IID split in between $n = 100$ total nodes, $s = 20$.

the test accuracy of FAVANO and competing methods on the MNIST dataset. We find that FAVANO and other asynchronous methods can offer a significant advantage over FedAvg when time is taken into account. However, QuAFL does not appear to be adapted to the non-IID environment. We identified client-side updating as a major shortcoming. While this is not severe when each client optimizes (almost) the same function, the QuAFL mechanism suffers from significant client drift when there is greater heterogeneity between clients. FedBuff is efficient when the

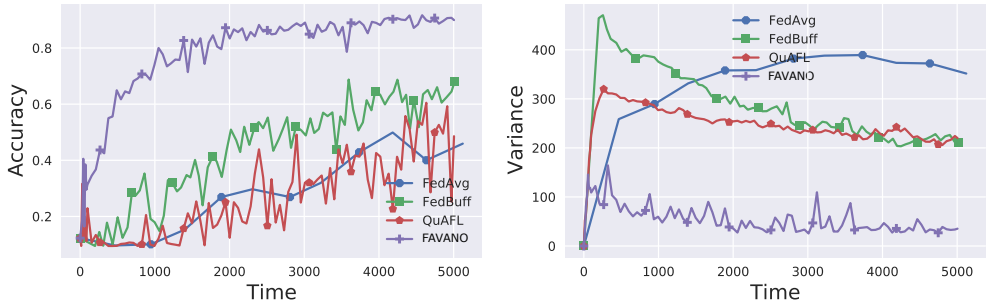
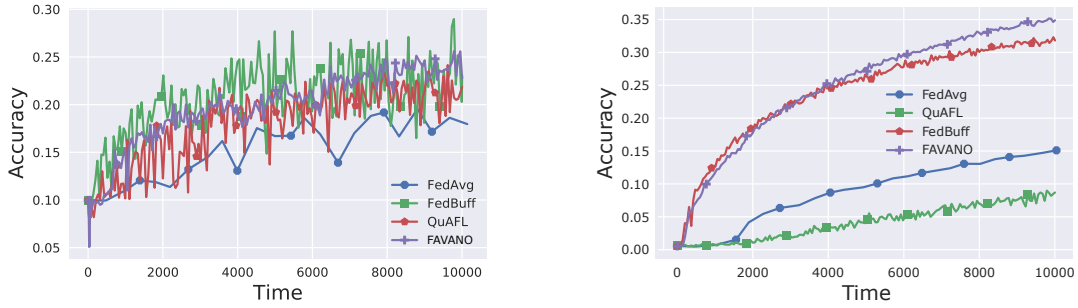


Figure 6.2: Test accuracy and variance on the MNIST dataset with a non-IID split between $n = 100$ total nodes. In this particular experiment, one-ninth of the clients are defined as fast.

number of stragglers is negligible compared to n. However, FedBuff is sensitive to the fraction of slow clients and may get stuck if the majority of clients are classified as slow and a few are classified as fast. In fact, fast clients will mainly feed the buffer, so the central updates will be heavily biased towards fast clients, and little information from slow clients will be considered. Figure 6.2 illustrates this phenomenon, where one-ninth of the clients are classified as fast. To provide a fair comparison, Table 6.2 gives the average performance of 10 random experiments with the different methods on the test set.

In Figure 6.3a, we report accuracy on a non-IID split of the CIFAR-10 dataset. FedBuff and FAVANO both perform better than other approaches, but FedBuff suffers from greater variance. We explain this limitation by the bias FedBuff provides in favor of fast clients. We also tested FAVANO on the TinyImageNet dataset (Y. Le and X. Yang 2015) with a ResNet18. TinyImageNet



(a) CIFAR-10 (non-IID)

(b) TinyImageNet (IID)

Figure 6.3: Test accuracy on CIFAR-10 and TinyImageNet datasets with $n = 100$ total nodes. Central server selects $s = 20$ clients at each round.

has 200 classes and each class has 500 (RGB) training images, 50 validation images and 50 test images. To train ResNet18, we follow the usual practices for training NNs: we resize the input images to 64×64 and then randomly flip them horizontally during training. During testing, we center-crop them to the appropriate size. The learning rate is set to 0.1 and the total simulated time is set to 10000. Figure 6.3b illustrates the performance of FAVANO in this experimental setup. While the partitioning of the training dataset follows an IID strategy, TinyImageNet provides enough diversity to challenge federated learning algorithms. Figure 6.3b shows that FAVANO scales much better on large image classification tasks than any of the methods we considered.

Remark 13. *We also evaluated the performance of FAVANO with and without quantization. We ran the code [2] from LUQ (Chmiel et al. 2021) and adapted it to our datasets and the FL framework. Even when the weights and activation functions are highly quantized, the results are close to their full precision counterpart (see Figure C.4 in Appendix C.2).*

6.6 Conclusion

We have presented FAVANO the first (centralised) Federated Learning method of federated averaging that accounts for asynchrony in resource-constrained environments. We established complexity bounds under verifiable assumptions with explicit dependence on all relevant constants. Empirical evaluation shows that FAVANO is more efficient than synchronous and asynchronous state-of-the-art mechanisms in standard CNN training benchmarks for image classification.

[2] https://openreview.net/forum?id=clwYez4n8e8

Chapter 7

Queuing dynamics in the context of FL with different node computation capacities

In this final chapter, we detail a new method to deal with asynchronous FL. This scheme is orthogonal to the algorithm FAVANO presented in the previous Chapter 6. Indeed, in all this chapter, nodes are not interrupted by the CS. However, each node is allowed to work on models with potential delays and contribute to updates to the CS at its own pace. This chapter is based on the work Leconte, Jonckheere, et al. 2024:

Leconte Louis, Matthieu Jonckheere, Sergey Samsonov, and Eric Moulines. (2024). "Queuing dynamics of asynchronous Federated Learning". In:International Conference on Artificial Intelligence and Statistics. PMLR.

7.1 Introduction

Federated learning (FL) is a distributed learning paradigm that allows agents to learn a model without sharing data (Konečnỳ, B. McMahan, and Ramage 2015; B. McMahan et al. 2017). A central server (CS) coordinates the entire process. In most implementations, the CS uses synchronous operations. During each epoch, the CS communicates with a subset of clients and waits for their "local updates". The CS then uses these local updates to update the global model; B. McMahan et al. 2017; Shaobo Wang et al. 2021. Nevertheless, different computational speeds, latencies, and/or transmission bandwidths lead to a cascade of issues such as delays and stragglers. In each epoch, CS must keep up with the pace of the slowest agent.

A solution called FedAsync eliminates the structured rounds of CS interaction and transitions to asynchronous optimization (Xie, Koyejo, and I. Gupta 2019). This approach, along with subsequent works in this direction (Y. Chen et al. 2020; Z. Chen et al. 2021; C. Xu et al. 2021), enables asynchronous operation for the CS and agents. FedAsync facilitates the aggregation of agents updates through the CS, rendering the solution highly scalable. More recently, Mishchenko, F. Bach, et al. 2022 has expanded the theoretical comprehension of purely asynchronous SGD within a homogeneous framework where all agents can access identical data; certain limitations still persist in heterogeneous scenarios.

In practical applications of asynchronous federated learning (FL), interactions between agents and the CS require the use of queues for processing (potentially) multiple jobs. The distribution of processing delays varies significantly across agents, and this variability has been shown to have a negative impact on optimization processes. In this chapter, we significantly improve the analysis delineated in Koloskova, Sebastian U Stich, and Jaggi 2022, exploring in depth an asynchronous algorithm—AsyncSGD. This algorithm empowers nodes to queue tasks, initiating communication with the central server upon task completion. The subsequent analysis adheres to a virtual iterates sequence under standard non-convex assumptions. However, previous studies made overly simplistic assumptions about the dynamics of queues, choosing to represent them with an upper bound on the processing delays encountered by the CS. In contrast, our theory intricately models the queuing dynamics using a **stationary closed Jackson network**. This approach allows capturing precisely the queuing dynamics - number of buffered tasks, processing delay, etc...-, as a function of agents speed. We integrate assumptions about the service time distributions, enabling us to define the explicit stationary distribution of the number of in-service tasks.

7.2 Related works

Up to this point, the focus has been on synchronous federated learning techniques, as evidenced by notable contributions such as (J. Wang, Qinghua Liu, et al. 2020; Qu, Song, and Tsui 2021; Makarenko et al. 2022; Y. Mao et al. 2022; Tyurin and Richtárik 2022). However, synchronous methods often suffer from suboptimal resource allocation and long training times. Moreover, as the number of participating agents grows, coordinating synchronous rounds with all participants becomes an increasingly difficult task for the central server (CS).

Synchronous federated learning methods are particularly vulnerable to the challenge of stragglers, prompting the emergence of research endeavors rooted in the principles of FedAsync and its subsequent extensions, as elucidated by Xie, Koyejo, and I. Gupta 2019. The core concept revolves around updating the global model upon receiving a local model at the central server (CS). ASO-Fed (Y. Chen et al. 2020) introduces memory-based mechanisms on the local client side. AsyncFedED (Q. Wang et al. 2022), drawing inspiration from FedAsync's instantaneous

update strategy, proposes dynamic adjustments to the learning rate and the number of local epochs to mitigate staleness.

Looking at the problem from a different perspective, QuAFL (Zakerinia et al. 2022) introduces a concurrent algorithm that aligns closely with the FedAvg strategy. QuAFL seamlessly integrates asynchronous and compressed communication methods while ensuring convergence. In this approach, each client is allowed a maximum of K local steps and can be interrupted. To address the variability in computational speeds across nodes, FAVANO (as discussed by Leconte, V. M. Nguyen, and Moulines 2023) strikes a balance between the slower and faster clients.

FedBuff (J. Nguyen et al. 2022) addresses asynchrony and concurrency by incorporating a buffer on the server side. Clients conduct local iterations, with the CS updating the global model solely upon completion by a predefined number of different clients.

Similarly, the work presented by Koloskova, Sebastian U Stich, and Jaggi 2022 revolves around Asynchronous SGD (AsyncSGD), offering guarantees contingent on the maximum delay. Recent developments by Fraboni et al. 2023 expand upon the ideas presented by Koloskova, Sebastian U Stich, and Jaggi 2022, allowing multiple clients to contribute within a single round. J. Liu et al. 2021 diverges from the buffer-centric approach and develops Adaptive Asynchronous Federated Learning (AAFL) to address speed disparities among local devices. Similar to FedBuff, J. Liu et al. 2021's method entails only a fraction of locally updated models contributing to the global model update. Convergence guarantees within asynchronous distributed frameworks commonly rely on an analysis contingent upon the maximum delay (J. Nguyen et al. 2022; Toghani and Uribe 2022; Koloskova, Sebastian U Stich, and Jaggi 2022), which substantially exceeds the average delay.

Tyurin and Richtárik 2023 introduces a novel asynchronous algorithm, presenting optimal convergence guarantees under the assumption of fixed computational speeds among workers over time. Notably, Mishchenko, F. Bach, et al. 2022 conducts an independent analysis of asynchronous stochastic gradient descent that does not rely on gradient delay. However, in the context of a heterogeneous (non-i.i.d.) setting, convergence is guaranteed up to an additive term linked to the dissimilarity limit between the gradients of local and global objective functions.

AsGrad (Islamov, Safaryan, and Alistarh 2023) is a recent contribution that proposes a general analysis of asynchronous FL under bounded gradient assumption, and adapt random shuffling SGD to the asynchronous case. Most of standard asynchronous baselines can be expressed in the general form proposed in Islamov, Safaryan, and Alistarh 2023, and strong convergence guarantees are provided. But all derivations assume delays are finite quantities.

While an impressive body of research has been dedicated to establishing theoretical tools for the performance evaluation of communication networks, including the development of intricate scheduling mechanisms and models (as exemplified in Malekpourshahraki et al. 2022; Stavrinides and Karatza 2018 and references therein), the predominant focus has revolved around performance metrics such as delays/completion times (measured in terms of physical time per node), queue lengths, and throughputs. When it comes to modeling federated learning, the application of stochastic network paradigms is significant, based on a wealth of results in the field. However, it is important to recognize that the key metrics to be computed in FL are significantly different from traditional network metrics, as we will discuss in more detail shortly. In particular, the measurement of delay in this context must take into account server steps, which introduces a more complicated dependence on the dynamics of each queue within the network. Moreover, optimizing these novel metrics may require completely different resource allocation paradigms.

7.3 Problem statement

We consider the optimization problem:

$$\min_{w\in\mathbb{R}^d} \sum_{i=1}^n \mathbb{E}_{(x,y)\sim D_i^{\text{data}}}[\ell_i(\text{nn}(x,w),y)]. \tag{7.1}$$

Here d is the number of parameters (network weights and biases), n is the total number of clients, ℓ_i is the local loss function, $\text{nn}(x,w)$ is the DNN prediction function, D_i^{data} is the training data distribution on node i. In FL, the distributions D_i^{data} are allowed to differ between clients (statistical heterogeneity). Let us denote by

$$f_i(w) := \mathbb{E}_{(x,y)\sim D_i^{\text{data}}}[\ell_i(\text{nn}(x,w),y)] \tag{7.2}$$

the local function optimized on node i and $f := \frac{1}{n}\sum_{i=1}^n f_i$. Each node i does not compute the true gradient of the function f_i, but has access to a *stochastic* version of the gradient, denoted by $\tilde{g}_i$.

We consider the *task* as a computation of a gradient (or possibly stochastic gradient) on one of the clients. We assume that n clients process a fixed number of tasks $\mathrm{C} \leq n$ in parallel, and the total number of CS steps is T. Once a task is completed by an agent, the corresponding update is sent to the CS, which updates the global model and then passes the updated parameters to a new agent with probabilities $(p_i)_{i=1}^n$. The selected agent might already be busy computing a previous update. When the agent is busy, the new job enters a queue that is serviced on a first-in-first-out basis (FIFO). To perform this analysis, we must establish the following definitions:

- $J_k \in [1,n]$ is the node completing a task at the k-th CS epoch (or step),
- $K_{k+1} \in [1,n]$ is the node selected at step $k \in [1,T]$,
- $X_{i,k}$ is the number of tasks in node i at step k, $i \in [1,n]$, $k \in [1,T]$.

All these random variables can be constructed as deterministic functions of the i.i.d. sequence $(R_l)_{l\in\mathbb{N}}$ which stands for the routing decisions and the sequences $(\xi_l^i)_{l\in\mathbb{N},i=1,\ldots,n}$ which stand for the service times (durations) of the tasks in each node. These two i.i.d. sequences are independent.

We denote by $\mathrm{M}_{i,k}^T$ the number of CS steps between the time that a task is sent to node i and the time it is completed:

$$\mathrm{M}_{i,k}^T = \mathbb{1}_{\{i\}}(K_{k+1}) \sum_{r=k}^T \mathbb{1}_{(\sum_{l=k}^r \mathbb{1}_{J_l=i})<X_{i,k}} \tag{7.3}$$

Finally, for $k \in \{0,\ldots,T\}$ we define

$$I_k^T = \sum_{l=0}^k l \cdot \mathbb{1}_{\{k-l\}}(\textstyle\sum_{i=1}^n \mathrm{M}_{i,\ell}^T), \tag{7.4}$$

which is the CS step corresponding when the task was dispatched to node J_k.

Direct analysis of the server iterate $(w_k)_{k\geq 0}$ is difficult because we do not have access to the joint distribution of $(J_k)_{k\geq 0}$, $(\mathrm{M}_{i,k}^T)_{k\geq 0}$ and $(I_k^T)_{k\geq 0}$. Koloskova, Sebastian U Stich, and Jaggi 2022 assumes an upper bound on the number of gradients that are pending at step k but have not yet been applied. Koloskova, Sebastian U Stich, and Jaggi 2022 proposes to select new nodes with uniform probability. In `Generalized AsyncSGD` (see Algorithm 9), we add some degree of freedom by allowing the central server to select a new node K_{k+1} with (possibly non-uniform) probability $\mathbf{p} = (p_j)_{j=1}^n$.

We denote

$$\mathrm{m}_{i,k}^T := \mathbb{E}[\mathrm{M}_{i,k}^T], \text{ and } \mathrm{m}_k^T := \sum_{i=1}^n \mathrm{m}_{i,k}^T/(n^2 p_i^2).$$

Algorithme 9 : `Generalized AsyncSGD`

Input : Number of server steps T, Number of tasks C ;
`/* At the Central Server */`
Initialize
  Initialize parameters w_0;
  Select initial set of clients $\mathcal{S}_0$, with $|\mathcal{S}_0| = \mathrm{C}$;
  Server sends w_0 to each client in $\mathcal{S}_0$;
  All clients in $\mathcal{S}_0$ compute gradients on w_0 ;
  Compute optimal $(\mathbf{p}, \eta)$ by minimizing (7.8) ;
end
for $k = 0, \ldots, T$ **do**
  Server receives stochastic gradient $\tilde{g}_{J_k}(w_{I_k})$;
  Update $w_{k+1} \leftarrow w_k - \frac{\eta}{np_{J_k}} \tilde{g}_{J_k}(w_{I_k})$;
  Sample a new client K_{k+1} with prob. $p_{K_{k+1}}$;
  Send new model w_{k+1} to K_{k+1} ;
end

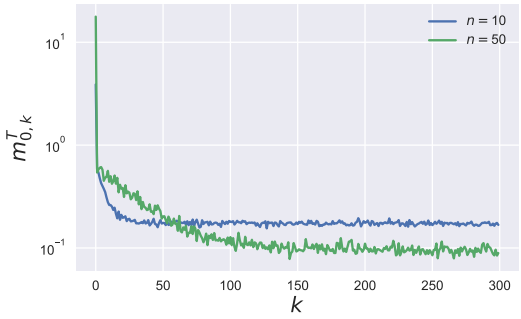


Figure 7.1: Evolution of $\mathrm{m}_{i,k}^T$ w.r.t. k, for two networks of size $n = 10, 50$ initialized with full concurrency.

It is worth noting that $\mathrm{m}_{i,k}^T$ depends on the sampling probability p_i, but for simplicity, we prefer not to index explicitly by $\mathbf{p} := (p_j)_{j=1}^n$. We will show in Section 7.5 that, $\lim_{k\to\infty} \lim_{T\to\infty} \mathrm{m}_{i,k}^T = \mathrm{m}_i$ (these expectations become stationary) see Proposition 16. Of course, the analysis of the transient behavior is very complex: simple upper bounds can be computed, but these are generally not expressive and hide the influence of key parameters. In Figure 7.1, we simulate $n = 10$ and $n = 50$ nodes with $\mathrm{C} = n$ initial tasks. In this simulation, nodes $\{0, 1, 2, 3, 4\}$ are 10 times faster than the other nodes to compute a task. Without loss of generality, we focus on the first node ($i = 1$), and we plot the value of $\mathrm{m}_{i,k}^T$ with respect to k, for $T = 500$. The value of $\mathrm{m}_{i,k}^T$ becomes stationary when $k > 50$ and $k > 150$, for $n = 10$ and $n = 50$, respectively.

In our analysis, in line with the approach presented in Koloskova, Sebastian U Stich, and

Jaggi 2022 (but the same idea was applied earlier), we introduce the *virtual iterates* μ_k as follows:

$$\begin{cases} \mu_0 = w_0, \\ \mu_1 = \mu_0 - \eta \sum_{i \in \mathcal{S}_0} \frac{1}{np_i} \tilde{g}_i(w_0), \\ \mu_{k+1} = \mu_k - \frac{\eta}{np_{K_k}} \tilde{g}_{K_k}(w_k), \quad k \geq 1 \,. \end{cases} \tag{7.5}$$

In fact μ_k is defined as if the selected client K_k was instantaneously contributing to the server update. Note that the gradients are computed on w_k, not on μ_k. The difference between μ_k and w_k consists of all the gradients computed (on potentially outdated w's) and not applied yet, $\mu_k - w_k = \sum_{(i,j) \in \mathcal{I}_k} \frac{-1}{np_i} \tilde{g}_i(w_j)$, with $\mathcal{I}_k = \{(i,j) \in [1,n] \times [1,k] | (X_{i,k} \neq 0) \text{ and } (\sum_{i=1}^n \mathrm{M}_{i,j}^T > k - j)\}$.

7.4 Non-convex bounds

Following the setting considered in Koloskova, Sebastian U Stich, and Jaggi 2022, J. Nguyen et al. 2022, we focus on the scenario of the optimization problem (7.1) with L-smooth and nonconvex objective functions f_i. Proofs are detailed in Appendix D.2. Our analysis is based on the following assumptions:

A14. *Uniform Lower Bound: There exists $f_* \in \mathbb{R}$ such that $f(w) \geq f_*$ for all $w \in \mathbb{R}^d$.*

A15. *Smooth Gradients: For any client i, the gradient ∇f_i is L-Lipschitz continuous for some $L > 0$, i.e. for all $w, \mu \in \mathbb{R}^d$: $\|\nabla f_i(w) - \nabla f_i(v)\| \leq L\|w - v\|$.*

A16. *Bounded Variance: For any client i, the variance of the stochastic gradients is bounded by some $\sigma^2 > 0$, i.e. for all $w \in \mathbb{R}^d$: $\mathbb{E}[\|\widetilde{g}_i(w) - \nabla f_i(w)\|^2] \leq \sigma^2$.*

A17. *Bounded Gradient Dissimilarity: There exist constant G, such that for all $w \in \mathbb{R}^d$: $\|\nabla f_i(w) - \nabla f(w)\|^2 \leq G^2$.*

The assumption **A**16 can be generalized to the *strong growth condition* (S. Vaswani, F. Bach, and Schmidt 2019):

$$\mathbb{E}[\|\widetilde{g}_i(w) - \nabla f_i(w)\|^2] \leq \sigma^2 + \rho^2 \|\nabla f_i(w)\|^2 \,,$$

following Beznosikov, Samsonov, et al. 2023. Full details are given in the appendix.

Theorem 14. *Assume **A**14 to **A**17 and let the learning rate η satisfy $\eta \leq \eta_{\max}(\mathbf{p})$, where*

$$\eta_{\max}(\mathbf{p}) =: \frac{1}{4L}\Big(\mathrm{C}^{-1/2} \max_{k \leq T} \{\mathrm{m}_k^T\}^{-1/2} \wedge 2/\sum_{i=1}^n \frac{1}{n^2 p_i}\Big). \tag{7.6}$$

Then `Generalized AsyncSGD` *converges at rate:*

$$\sum_{k=0}^T \frac{\mathbb{E}[\|\nabla f(w_k)\|^2]}{8(T+1)} \leq \frac{\mathbb{E}[f(\mu_0) - f(\mu_{T+1})]}{\eta(T+1)} + \frac{\eta LB}{n} \sum_{i=1}^n \frac{1}{np_i} + \frac{\eta^2 L^2 B \mathrm{C}}{n} \sum_{i=1}^n \frac{\sum_{k=0}^T m_{i,k}^T}{np_i^2 (T+1)}, \tag{7.7}$$

where $B = 2G^2 + \sigma^2$.

The upper bound in Theorem 14 includes three distinct terms. The first term is a standard component associated with a general nonconvex objective function; it expresses how the choice of initialization affects the convergence process.

The second and third terms depend on the statistical heterogeneity within the client distributions and the fluctuations of the minibatch gradients. If we assume uniform probabilities, the second term agrees with that of FedAvg: this is the bound that would be obtained with synchronous optimization. In contrast, the third term encapsulates the unique challenge posed by optimization within an asynchronous framework.

Before moving on to the main study, it is important to analyze the behavior of this bound. Note first that the upper bound is minimized by $T \to \infty$ if we set $\eta = O(T^{-1/2})$. In this setting, the third term of the upper bound, which is proportional to η^2, becomes negligible. To obtain the optimal probability value $\mathbf{p}$, one should minimize $\sum_{i=1}^{n} 1/p_i$ in this regime, subject to the condition $\sum_{i=1}^{n} p_i = 1$. This minimization is achieved when $p_i = 1/n$. Thus, with $T \to \infty$, a uniform distribution of weights turns out to be a reasonable choice.

A worked-out example For regimes other than $T \to \infty$, the bound given by Eq. (7.7) proves difficult to handle due to the complicated relationship between $\mathrm{m}_{i,k}^{T}$ and the sampling distribution $\mathbf{p}$. In the next section, we will apply queuing theory methods to shed light on these quantities. However, before diving into this detailed analysis, we will first examine the behavior of the bound using a simple example. We choose the sampling probabilities $\mathbf{p}$ and the step size η by solving the constrained optimization problem $\min_{\mathbf{p},\eta} G(\mathbf{p},\eta)$ as a function of $\eta \leq \eta_{\max}(\mathbf{p})$, where

$$G(\mathbf{p},\eta) = \frac{A}{\eta(T+1)} + \frac{\eta LB}{n}\sum\nolimits_{i=1}^{n}\frac{1}{np_i} + \frac{\eta^2 L^2 BC}{n}\sum\nolimits_{i=1}^{n}\frac{\sum_{k=0}^{T}\mathrm{m}_{i,k}^{T}}{np_i^2(T+1)}, \tag{7.8}$$

and where $A = \mathbb{E}[f(\mu_0) - f(\mu_{T+1})]$. To better understand this bound, let us examine a simple case. Suppose we have $n = 100$ clients that are classified as either "fast" or "slow". There are $n_f = 90$ fast clients and $n - n_f = 10$ slow clients, which are assumed to have the same characteristic (within each group). We will focus on how the proposed bound behaves based on the ratio of the processing speed of the "fast" and "slow" clients, the proportions of fast and slow clients, and the concurrency. We will also examine two situations: one in which the processing time for gradient requests is fixed, and another in which it follows an exponential distribution. By default, slow clients process a gradient in a time unit of 1 (on average in the random case), while fast clients take $\frac{1}{\mu_f} \leq 1$ units on average. Let $p \in (0,1)$. We denote p the probability to select one of the fast clients, and $q = \frac{1}{n-n_f} - p\frac{n_f}{n-n_f}$ the probability of selecting one of the slow clients (we need $n_f p + (n-n_f)q = 1$). The parameters are $L = 1$, $B = 20$ (to assess the effects of gradient noise and statistical heterogeneity), $A = 100$ (to highlight the impact of initial conditions). The number of tasks is varied $\mathrm{C} = 10, 50, 100$, to assess the impact of concurrency. For the total number of CS iterations, we consider $T = 10^4$. We plot the selection probability p versus μ_f, ranging from 2 to 16 in Figure 7.2. Furthermore, we graphically illustrate the relative improvement of the upper bound when compared with the uniform selection problem in Figure 7.3. The results show that a significant improvement may be achieved (from 30% when $\mu_f = 2$ to 55% when $\mu_f = 16$). To achieve this improvement, we should decrease the probability of selecting fast clients to $p = 7.3 \cdot 10^{-3}$. The conclusion (that will be justified theoretically in the next section) is that fast agents should be selected less frequently than slow agents. Even though this result may appear to be counter intuitive, it is justified by the fact that by selecting slow customers more frequently, processing times are reduced. Our simulations shows the average delay is divided by 10 and 2, for fast and slow nodes respectively. More details are given in Appendix D.5.

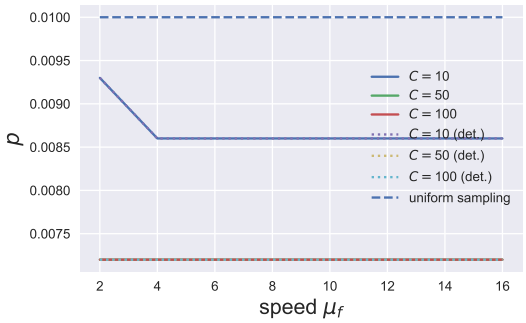


Figure 7.2: Optimal sampling probability p as a function of the speed for different concurrency levels. The number of nodes is fixed to $n = 100$ nodes.

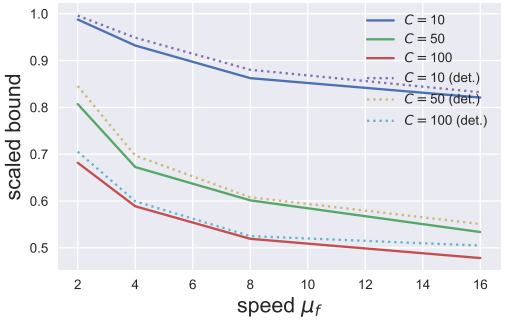


Figure 7.3: Relative improvements of the upper bounds as a function of the speed for different concurrency levels. The number of nodes is fixed to $n = 100$ nodes.

Finally, these experiments also show that the distribution of the working time required for gradient evaluation does not have a significant impact: results are very similar whether the working time is deterministic or distributed according to an exponential (and therefore random) distribution (provided that the mean are preserved).

Comparison with `FedBuff` and `AsyncSGD` We emphasize that previous analyses of both `FedBuf` and `AsyncSGD` are based on strong assumptions: the queuing process is not considered in their analysis. In practice, slow clients with delayed information contribute. J. Nguyen et al. 2022; Koloskova, Sebastian U Stich, and Jaggi 2022 propose to bound this delay uniformly by a quantity τ_{max}. We retain this notation while reporting complexity bounds in Table 7.1, but argue that nothing guarantees that τ_{max} is properly defined. In Figure 7.4 we compared the relative improvements of the upper bounds obtained with `Generalized AsyncSGD`, w.r.t. `FedBuff` and `AsyncSGD` for the scenario described in the previous paragraph. The plot illustrates the massive improvement achieved by `Generalized AsyncSGD` when optimal selection probabilities are used. It also illustrates that the bounds previously reported in the literature do not capture the essence of the problem. In particular, this comparison holds under the condition that the work time for gradient evaluation is deterministic, such that $\tau_{\max}$ equals C times the work time of a slow client. When the working time is exponential, the maximum delay as defined in the analyses of `FedBuff` and `AsyncSGD` is infinite, and the bounds in J. Nguyen et al. 2022 and Koloskova, Sebastian U Stich, and Jaggi 2022 are then empty.

Table 7.1: Asynchronous bounds (up to numerical constants) under non-convex assumption for T server steps. C is the number of initial tasks. $A = \mathbb{E}[f(\mu_0) - f_*]$, and $B = 2G^2 + \sigma^2$. $\tau_{\max}$ is defined in Toghani and Uribe 2022 as the maximum delay. $\tau_c, \tau^i_{\text{sum}}$ are defined in Koloskova, Sebastian U Stich, and Jaggi 2022 as the average number of active nodes, and the sum of delays of node i, respectively.

Method	Bounds	η
FedBuff	$\frac{A}{\eta(T+1)} + \eta LB + \eta^2 \tau_{\max}^2 L^2 Bn$	$\leq \frac{1}{L\sqrt{\tau_{\max}^3}}$
AsyncSGD	$\frac{A}{\eta(T+1)} + \eta LB + \eta^2 \tau_c L^2 B \sum_{i=1}^n \frac{\tau^i_{\text{sum}}}{T+1}$	$\leq \frac{1}{L\sqrt{\tau_c \tau_{\max}}}$
Generalized AsyncSGD	$\frac{A}{\eta(T+1)} + \eta LB \sum_{i=1}^n \frac{1}{n^2 p_i} + \eta^2 C L^2 B \sum_{i=1}^n \frac{\sum_{k=0}^T \mathrm{m}_{i,k}^T}{n^2 p_i^2 (T+1)}$	$\leq \frac{1}{L\sqrt{C \max_{k \leq T} \mathrm{m}_k^T}}$

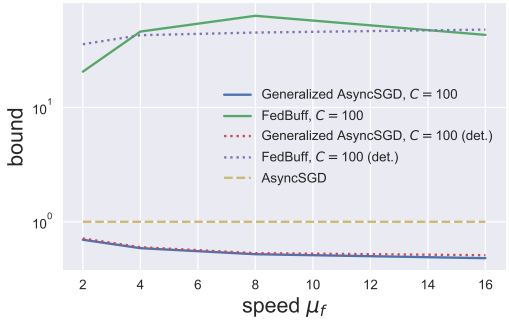


Figure 7.4: Relative improvement of Generalized AsyncSGD over FedBuff and AsyncSGD as a function of speed. The number of nodes is fixed to $n = 100$ nodes.

Physical time w.r.t. CS number of epochs The complexity bounds of `Generalized AsyncSGD` are based on the number of communication rounds. The conclusions would, of course, be different if we took physical time as the criterion. Indeed, when we determine complexity in terms of number of communications, we do not take into account the time intervals between two successive arrivals at the central server (whose law depends on the relative speeds of the agents and the weights $\mathbf{p}$). We discuss bounds for `Generalized AsyncSGD` w.r.t. physical time in Appendix D.4.2 and assess them.

7.5 Closed network

The aim of this section is to obtain theoretical guarantees using precise results on closed Jackson networks (R. Jackson 1954; J. R. Jackson 1957). We assume in this section that task duration follows an exponential distribution, with each user having its own mean. While it is feasible to extend these findings to deterministic durations and even almost arbitrary duration distributions, this complicates the theory. This approach not only captures the different speeds of the clients, but also allows for a precise assessment of the system dynamics. Consequently, we can accurately determine the critical values $\mathrm{m}_{i,k}^T$ in a steady state. Detailed proofs are postponed to the Appendix D.3.

7.5.1 Stationary distribution and key performance indicators

Let us denote by $(D_i(t))_{i=1,\dots,n}$ the number of task departures from node i at time t, with the convention that $D_i(0) = 0$. $(T_{i,l})_{l\in\mathbb{N}}$ are the jump times associated with the counting process D_i. We further denote by $N(t)$ the number of tasks arriving at the CS, given by

$$N(t) = \textstyle\sum_{i=1}^n D_i(t),$$

while $(T_l)_{l\in\mathbb{N}}$ are the jump times associated with N. Observe that the indices k used in the previous Section correspond to those jump times. Finally we define the sequences of times $(\tau_{i,l})_{l\in\mathbb{N}}$ as the arrival times to node i after time 0.

In what follows, we assume that the task duration is i.i.d. and exponentially distributed at rate μ_i, and that the routing decisions are also i.i.d. (and independent of everything else). As in the previous section, we denote by p_i the probability that the dispatcher sends a task to node i.

We denote by $X(t) = (X_1(t),\dots,X_n(t))$ the continuous-time stochastic process describing the number of tasks in each node. The unit vectors in $\mathbb{N}^n$ are denoted by $(e_i)_{i=1\dots,n}$. We have the following results for X.

Proposition 15. *Under the above assumptions, the dynamics of $(X(t), t \geq 0)$ is that of a closed Jackson network on the complete graph with n nodes and* C *tasks. The generator of the corresponding jump Markov process is given for all $x \in \mathbb{N}^n, i \in \mathbb{N}, j \in \mathbb{N}$ by*

$$q(x, x + e_i - e_j) = p_i \mu_j \mathbb{1}(x_j > 0).$$

Furthermore, defining $\theta_i = \frac{p_i}{\mu_i}$, the stationary distribution of X may be expressed as:

$$\pi_C(x_1,\dots,x_n) = H_{\mathrm{C}}^{-1} \textstyle\prod_{i=1}^n \theta_i^{x_i}, \tag{7.9}$$

with $H_{\mathrm{C}} = \sum_{x:\sum_i x_i = \mathrm{C}} \prod_{i=1}^n \theta_i^{x_i}$.

Building on the understanding gained in Section 7.4, our goal is to quantify the number of server steps that are executed when a new task arrives at a given node (say node i) and subsequently returns to the dispatcher.

For simplicity, the analysis is performed in the stationary regime. In particular, this means that at time 0 the distribution of the number of tasks in each node follows the product measure defined in Proposition 15. We denote by $\mathbb{E}^{\mathrm{C}}$ the stationary average of the closed Jackson network when the total number of tasks is equal to C.

We can now state the main result of this section:

Proposition 16. *Given the model assumptions and assuming stationarity, for all $k \in \mathbb{N}$,*

$$\lim_{T\to\infty} \mathrm{m}_{i,k}^T = \mathbb{E}^{\mathrm{C}-1}\Big[\int_0^{S_i} \sum_{j=1}^n \mu_j \mathbb{1}(X_j(s)>0)\,\mathrm{d}s\Big]. \tag{7.10}$$

From now on, we use the notation $\mathrm{m}_i = \lim_{T\to\infty} \mathrm{m}_{i,k}^T$. This quantity is in general difficult to simplify further. We consider in the sequel a specific regime in which we can obtain tractable expressions as the Jackson network gets close to saturation. We now describe how the queue length X_i, and m_i depend on the agents speed and selection probability $\mathbf{p}$, under this saturated stationary regime similar to the Halfin-Whitt regime in queuing theory (Halfin and Whitt 1981).

7.5.2 Scaling regime

We rely on scaling bounds to provide rules of thumb when certain traffic conditions are satisfied. We follow the derivation of Van Kreveld, Dorsman, and Mandjes 2021, which considers closed Jackson networks under a particular load regime. We assume without loss of generality that $\theta_n = \max_{i\in[1,n]}(\theta_i)$; where $\theta_i = p_i/\mu_i$ (see Proposition 15). Due to the closed nature of the network, rescaling all parameters through division by the maximum traffic load leads to a different normalization constant $\tilde{H}_C$, but otherwise has no effect on the stationary joint distribution:

$$\pi(x_1,\ldots,x_{n-1}) = \tilde{H}_C^{-1} \prod_{i=1}^{n-1} {\gamma_i}^{-x_i}, \tag{7.11}$$

where for all $i \in [1,n]$, $\gamma_i = \theta_n/\theta_i$. We consider a scaling regime where all nodes are saturated, but at different rates. In Appendix D.6 we also define a more complicated scenario where some queue lengths may degenerate to 0.

2 clusters under saturation We consider two clusters of nodes of size n_f and $n-n_f$, respectively. Nodes $i \in [1,n_f]$ are fast, the rest are slow. We assume that nodes from the same cluster have the same speed μ_f, μ_s, for fast and slow nodes, respectively ($\theta_f < \theta_s$). This gives the *scaled* intensity of the slow nodes $\gamma_s(\iota) = 1$, and the fast nodes $\gamma_f(\iota) := \frac{\theta_s}{\theta_f}$, where ι is the scaling parameter and the scaling regime corresponds to choosing those values as $\gamma_f(\iota) = 1 + c_f \iota^{\alpha-1}$; with $c_f > 0$ a fixed positive constant, and $\alpha \le 1$, while the total number of tasks also scales as follows:

$$\beta \iota^{1-\alpha} = \mathrm{C}+1\,.$$

Choosing $\alpha \le 1$ as in Van Kreveld, Dorsman, and Mandjes 2021 ensures that node loads approach 1 as $\iota \to \infty$, enabling the application of Corollary 2 from Van Kreveld, Dorsman, and Mandjes 2021. This yields precise results on saturated node queue lengths at high traffic loads, while queue lengths for the remaining nodes are determined by population size constraints. In this context, define X_i^ι as the stationary queue length for a scaling parameter ι and $\mathrm{m}_i(\iota)$ the corresponding value of m_i.

Proposition 17 (Corollary.2 in Van Kreveld, Dorsman, and Mandjes 2021)**.** *In stationary regime, as the scaling parameter $\iota \to \infty$,*

$$c_f \iota^{\alpha-1} X_i^\iota \to_{d.} \chi_i\,, \tag{7.12}$$

where $\chi_i = \mathbb{E}\Big[\, E_i \,|\, \sum_{j=1}^{n_f} E_j/c_f \le \beta\Big]$, $i \in [1, n_f]$, and the $(E_j)_{j\le n}$ are independent unit mean exponential distributions.

As a consequence, using uniform integrability, we can estimate the following expected value (expected stationary queue lengths of fast, and slow nodes respectively) as follows:

$$\begin{cases} \iota^{\alpha-1}\mathbb{E}[X_i^\iota] \to \frac{\Gamma(c_f\beta)}{c_f}, \quad \forall i \in [1, n_f], \\ \iota^{\alpha-1}\mathbb{E}[X_i^\iota] \to \frac{1}{n-n_f}\Big(\beta - n_f \frac{1}{c_f}\Gamma(c_f\beta)\Big), \forall i \in [n_f+1, n]. \end{cases} \tag{7.13}$$

Denoting by $P(k,x) = 1 - \sum_{i=0}^{k-1} e^{-x}\frac{x^i}{i!}$, we have:

$$\Gamma(c) = \frac{\mathbb{P}(\sum_{j=1}^{n_f+2} E_j \le c)}{\mathbb{P}(\sum_{j=1}^{n_f+1} E_j \le c)} = \frac{P(n_f+2, c)}{P(n_f+1, c)}.$$

We now turn to bound the key quantity $\mathrm{m}_i(\iota)$ for large ι.

Proposition 18. *Using the same assumptions as those of Proposition 17 we get that :*

$$\begin{cases} \limsup_{\iota\to\infty} \iota^{\alpha-1}\mu_f\,\mathrm{m}_i(\iota) & \le \lambda\frac{\Gamma(c_f\beta)}{c_f}, \quad \forall i \in [1, n_f], \\ \limsup_{\iota\to\infty} \iota^{\alpha-1}\mu_s\,\mathrm{m}_i(\iota) & \le \lambda\frac{\beta - n_f\Gamma(c_f\beta)/c_f}{n-n_f}, \textit{otherwise}, \end{cases} \tag{7.14}$$

where $\lambda = \sum_{i=1}^n \mu_i$.

We expect these bounds to be sharp for large ι.

Numerical example Under the previous assumptions, we have $\lambda = n_f\mu_f + (n-n_f)\mu_s$. We will further assume $n_f = \frac{n}{2}$, and $p_i = \frac{1}{n}$. Under these conditions, we have that $\Gamma(c_f\beta)$ is close to 1. We can give a closed form approximations of the bounds of the expected delays:

$$\begin{cases} \mathrm{m}_i(\iota) \le \frac{n(\mu_f+\mu_s)}{2\mu_f(\mu_f/\mu_s - 1)}, \quad \forall i \in [1, n_f], \\ \mathrm{m}_i(\iota) \le \Big(\frac{2\mathrm{C}}{n} - \frac{1}{\mu_f/\mu_s - 1}\Big)\frac{n(\mu_f+\mu_s)}{2\mu_s}, \quad \forall i \in [n_f+1, n]. \end{cases} \tag{7.15}$$

All delays bounds estimations have a closed form in the 2-cluster saturated regime: they only depend on the number of tasks in the network C, on the number of nodes n, and on the intensity of nodes μ_f, μ_s. More details on the derivations, and on the following experiment are available in Appendix D.5. We consider a numerical simulation with $n = 10$ clients, split in two clusters of same size: fast nodes with rate $\mu_f = 1.2$, and slow nodes with rates $\mu_s = 1$. We saturate the network with $\mathrm{C} = 1000$ tasks, and we simulate up to $T = 10^6$ server steps, and plot the distribution of the delays (in number of server steps). Our numerical experiment in Figure 7.5 gives average delays (50 and 1950 for fast and slow nodes, respectively) and queue lengths that correspond to the theoretical expected values. It is also important to point out that the average delays are way smaller than the maximum delay experienced in the $T = 10^6$ steps. This further

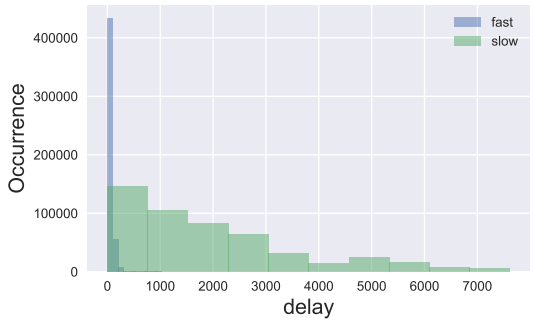


Figure 7.5: Histogram of fast and slow delays (in number of server steps) for a uniform sampling scheme.

highlights the necessity to switch from analysis that depend on the τ_{max} quantity, to our analysis that only depends on the expected delays.

7.6 Deep learning experiments

We evaluate FL algos performance on a classic image classification task: CIFAR-10 (Krizhevsky, G. Hinton, et al. 2009). We consider a non-i.i.d. split of the dataset: each client takes seven classes (out of the ten possible) without replacement. This process introduces heterogeneity among the clients.

We compare different asynchronous methods in terms of CS steps. In all experiments, we track the performance of each algorithm by evaluating the server model against an unseen validation dataset.

We decide to focus on nodes with different exponential service rates as in J. Nguyen et al. 2022. We build `AsyncSGD` and `Generalized AsyncSGD` codes from scratch. After simulating n clients, we randomly group them into fast or slow nodes. We assume that the clients have different computational speeds, and refer the readers to Appendix D.7.1 for further details. We have assumed that half of the clients are slow. We compare the classic asynchronous methods `FedBuff` (J. Nguyen et al. 2022), and `AsyncSGD` (Koloskova, Sebastian U Stich, and Jaggi 2022). Details about concurrent works implementation can be found in Appendix D.7.2.

We use the standard data augmentations and normalizations for all methods. All methods are implemented in Pytorch, and experiments are performed on an NVIDIA Tesla-P100 GPU. Standard multiclass cross entropy loss is used for all experiments. All models are fine-tuned with $n = 100$ clients, and a batch of size 128. We have finetuned the learning rate for each method. For `FedBuff` we tried several values for the buffer size, but finally found that the default one $Z = 10$ gives the best performances.

In Figure 7.6, we compare the performance of a Resnet20 (He et al. 2016) with the CIFAR-10 dataset, which consists of 50000 training images and 10000 test images (in 10 classes). The total number of CS steps is set to 200. Despite the heterogeneity between client datasets, we can achieve good performance on image classification. `FedBuff` has to fill up its buffer before performing an update, slowing down the training process. `AsyncSGD` provides acceptable performance, but we can go further by sampling fast nodes slightly less than the uniform (as suggested in Section 7.4), and this leads to much better accuracy.

We have additionally tested `Generalized AsyncSGD` on the TinyImageNet classification task

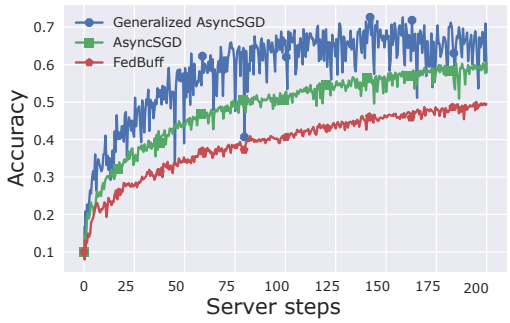


Figure 7.6: Accuracy on validation dataset on central server, for CIFAR-10 classification task.

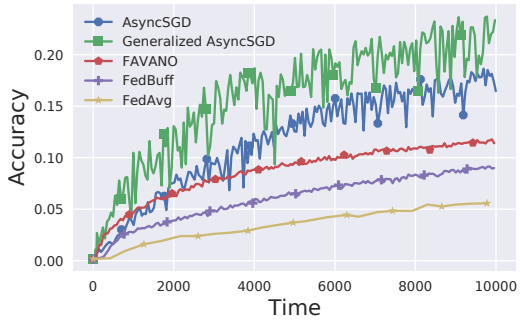


Figure 7.7: Test accuracy on TinyImageNet dataset with $n = 100$ total nodes.

(Y. Le and X. Yang 2015), with a ResNet18. We compare `Generalized AsyncSGD` with the classic synchronous approach FedAvg (B. McMahan et al. 2017) and two newer asynchronous methods FedBuff (J. Nguyen et al. 2022) and `FAVANO` (Leconte, V. M. Nguyen, and Moulines 2023). `FAVANO` (and the NN quantized QuAFL (Zakerinia et al. 2022)) follows a completely different method than we do. There are no queues: Clients are triggered at the CS and either withhold their results or are interrupted by the CS before the work is completed. In `FAVANO`, the clients can have a high latency. The update rate of the CS is limited by (slow) clients: The minimum time between two CS updates should be at least as long as the minimum time needed to process a gradient update. TinyImageNet has 200 classes and each class has 500 (RGB) training images, 50 validation images and 50 test images. To train ResNet18, we follow the usual practices for training NNs: we resize the input images to 64×64 and then randomly flip them horizontally during training. During testing, we center-crop them to the appropriate size. The learning rate is set to 0.001 and the total simulated time is set to 1000. Figure 7.7 illustrates the performance of `Generalized AsyncSGD` in this experimental setup. While the partitioning of the training dataset follows an IID strategy, TinyImageNet provides enough diversity to challenge federated learning algorithms. FedBuff is efficient when the number of stragglers is small. However, FedBuff is sensitive to the fraction of slow clients and may get stuck if the majority of clients in the buffer are more frequently the fast clients: this introduces a bias and few information from slow clients will be taken into account at the CS. `FAVANO` works better than FedBuff, but the CS updates should not be too small in order to allow slow clients to compute at least one local gradient step. However with AsyncSGD, no constraints are set on the time between two consecutive CS steps: it evolves freely based on the queuing processes. `Generalized AsyncSGD`

presents the same advantage, and in addition, samples clients with an optimal scheme. This leads to better performance, even on the challenging TinyImageNet benchmark.

7.7 Conclusion

In this chapter, we analyze the convergence of an Asynchronous Federated Learning mechanism in a heterogeneous environment. Through a detailed queuing dynamics analysis, we demonstrate significantly improved convergence rates for our algorithm `Generalized AsyncSGD`, eliminating dependence on the maximum delay $\tau_{\max}$ seen in previous works. Our algorithm enables non-uniform node sampling, enhancing flexibility. Empirical evaluations reveal `Generalized AsyncSGD` superior efficiency over both synchronous and asynchronous state-of-the-art methods in standard CNN training benchmarks for image classification tasks.

Chapter 8

Conclusion

The previous chapters cover the work produced during the past 3 years on a broad topic: efficiency for deep learning in a connected world. The very first research direction of this thesis was dedicated to Boolean/binary neural networks. Having a step back, we discovered that (i) the literature was large and diverse, and (ii) "efficiency" poses several problems and opens many research directions. In particular an important number of paper cover the topic of quantized neural network, but the "efficiency", when available, is only discussed in terms of FLOPS/BOPS (floating/binary operations). Very few articles propose a fair energy analysis or sound mathematical guarantees. In addition, all works assume that a single worker is amenable to manage and handle a learning process based on gradient descent. Taking into account the privacy and computational power limitations of (edge) devices yields the need of a Federated framework.

In this thesis, we investigated several aspects of stochastic optimization with the objective of reducing energy costs for possibly very heterogeneous devices.

In the opening pages of this thesis, we have provided a detailed overview of the contributions of this work. We will give a short summary of it in the following lines. Then, in Chapter 1, we propose a basic, yet essential introductory discussion about statistical learning, machine learning, deep learning, quantization, and federated learning.

In our first contribution, we present `AskewSGD` a novel framework for quantized neural network (QNN) training based on an annealed sequence of interval-constrained non-convex optimization problems solved by an algorithm inspired by Muehlebach and Jordan 2021. For each of the sub-problems, we give theoretical guarantees. We want to underline that the annealing strategy is elegant but not straightforward. We initially studied an Augmented Lagrangian optimization procedure (Zichong Li et al. 2021). The idea was to tackle the QNN training by a non-convex optimization with additional quantization constraints. Unfortunately, it was not possible to achieve decent performances on standard image classification tasks. We then moved on to `AskewSGD`. Even considering convex and non-convex classification tasks, `AskewSGD` outperforms or is on par with other QNN training methods on all considered tasks. All details are provided in Chapter 2.

In Chapter 3, we moved the focus toward a novel heuristic for training deep neural networks, that is provably efficient for resource-constrained environments. In particular, we have developed a method to estimate the energy consumption of NN training and apply it to our Boolean architectures. Our results suggest that full-precision performance can be totally recovered by enlarged Boolean models while gaining multi-fold complexity reduction. One can fine-tune

these energy-efficient models on edge devices for specific tasks. Our experiments highlight that Boolean models can handle finer tasks, contrary to the misbelief that binary models only work for image classification. We did not present it in this document, but we also studied the convergence properties of Boolean training under non-convex assumptions. The full proof is publicly available in Leconte 2024. We are able to prove that a BNN that follows the Boolean strategy (V. M. Nguyen 2023) will converge to a first-order stationary point, up to an irreducible "error floor". Previous work with quantized models also include error bounds (H. Li et al. 2017; Zheng Li and Sa 2019). As a negative result, we also have tried to formulate the training of a binary neural network (BNN) as a variational inference Bayesian problem. We can see a BNN as a realisation/sample of a Bayesian neural network. Hence, one needs to optimize the underlying distribution. Following the work of Meng, Bachmann, and Khan 2020 based on Gumbel softmax strategy, we have implemented a naive Reinforce strategy (Andriy Mnih and Gregor 2014). We have obtained promising results on small non-convex tasks, but our strategy was suffering from a high variance for larger problems.

In our third contribution, we have extended the Boolean training idea to the distributed case. In Chapter 4, we have provided two centralized Federated Learning methods which incorporate Boolean neural networks. Empirical evaluation shows that `FedBool` and `MajBool` are more efficient than synchronous state-of-the-art mechanisms on several image classification tasks. Following these first steps in the Federated world, we propose in Chapter 5 an unbiased compression technique, relying on unitarily invariant random codebooks. We demonstrate the performance of this compression technique, focusing on a simple but reliable metric, the distortion on the compressed vectors, that we carefully analyze and evaluate experimentally on various sources of vectors. We also describe how `Stovoq` can be integrated within (any) Federated Learning algorithm and demonstrate that we can leverage many of the convergence guarantees provided in the literature.

Our last contributions consider the asynchronous FL setting: we *do not* assume that nodes contribute to the central server (CS) at the same pace. In particular, nodes can have very different computational speeds and/or a very different access to the bandwidth. This creates a serious bottleneck: in practical scenari, one does not want to wait for the slowest node at every CS update. In Chapter 6, we have presented `FAVANO` the first (centralised) Federated Learning method of federated averaging that accounts for asynchrony in resource-constrained environments. We established complexity bounds under verifiable assumptions with explicit dependence on all relevant constants. Empirical evaluation shows that `FAVANO` is more efficient than synchronous and asynchronous state-of-the-art mechanisms in standard CNN training benchmarks for image classification. In Chapter 7, we analyze the convergence of an Asynchronous Federated Learning mechanism in a heterogeneous environment. Through a detailed queuing dynamics analysis, we demonstrate significantly improved convergence rates for our algorithm `Generalized AsyncSGD`, eliminating dependence on the maximum delay $\tau_{\max}$ seen in previous works. Our algorithm enables non-uniform node sampling, enhancing flexibility. Empirical evaluations reveal `Generalized AsyncSGD` superior efficiency over both synchronous and asynchronous state-of-the-art methods in standard CNN training benchmarks for image classification tasks.

In summary, this thesis presents novel contributions to the field of quantized neural networks and Federated Learning, addressing critical challenges and offering innovative solutions for efficient and sustainable learning in a distributed and heterogeneous environment. This work is in line with a global initiative to make large-scale Machine Learning more environmentally friendly by minimizing its environmental impact. While the potential benefits are promising, particularly in terms of energy savings, it is crucial to exercise caution, as a rebound effect could occur: the use of faster and more energy-efficient algorithms may lead to an increase in their applications, potentially offsetting or even reversing the gains made through their design.

This negative impact is already evident in our daily interactions with technology (ads, video recommendations, music streaming services, etc.).

Chapter 9

Perspectives

We expose in this chapter perspectives for future work based on the contributions presented in the previous chapters, and inspired by the unsuccessful research directions that we mentioned in the previous Chapter 8.

First, several interesting research directions can be drawn from our initial contributions. As an example, all convergence guarantees were developed under the non-convex regime. Extending to convex assumptions can further provide insights on the rate and the scaling of the step sizes. Many algorithms can also be combined. For instance `AskewSGD` can be integrated into any FL framework such as `FAVANO` or `Generalized AsyncSGD`. Additionally, on top of that, the bandwidth consumption can be relieved through the use of `Dostovoq`. Moreover, our contributions in the asynchronous federated learning (FL) field open directions for further analysis and new algorithm. In particular, in lines with the literature, the complexity bounds we obtained in Chapters 6 and 7 present a dissimilarity terms "G^2" that plays a similar role as the noise from the stochastic gradient σ^2. Some works (Koloskova, Sebastian U Stich, and Jaggi 2022; Islamov, Safaryan, and Alistarh 2023) decouple these two terms (as factor of second order of the step size) by adding an additional assumption on the step size. But this is not satisfactory, the same assumption can be used to also eliminate G^2 for second order step size terms. Maybe a true decoupling can be obtained by refining the assumption on gradient dissimilarity. Last, but not least, a huge step forward will be done in the field of asynchronous FL if the data heterogeneity influence can be discarded in a manner reminiscent to the control variate strategy of Karimireddy, Kale, et al. 2020. To the best of our knowledge, control variate are *not* successful for asynchronous updates. While following the prove from Karimireddy, Kale, et al. 2020, we were not able to prove that the "control variate drift" term is contractive in the asynchronous regime.

Second, investing into a new technology is always a trade-off in between the new benefits we can gain from it, and its negative societal impact. Large language models (LLMs) are a good example of such. Notwithstanding its popularity and the practical time savings it offers to companies, the training but *also* the mere inference of a LLM consume hundreds of tons of CO_2e (Touvron et al. 2023). But on the other side, I believe LLM could benefit the most. In particular, for educational purposes, one could imagine that a pretrained LLM could be fine-tuned on an intermediary level courses dataset. With proper engineered limits, the resulting chatbot would perfectly fit as a personal teaching assistant. Underprivileged children will have a free help for their homework/questions: it would be a very useful tool to circumvent any biased educational system. However, to make this idea free and sustainable, one needs to reduce the environmental

cost of LLMs. To do so, the approach LQ-LORA (H. Guo et al. 2023) propose to decompose a fixed (full precision) pre-trained matrix W into the sum of a low-rank and quantized components. Based on this idea, we are currently studying an approach that goes further by implementing a vector quantization of the LLM matrices.

Bibliography

Agarap, Abien Fred (2018). "Deep learning using rectified linear units (relu)". In: *arXiv preprint arXiv:1803.08375*.

Agarwal, Naman et al. (2018). "cpSGD: Communication-efficient and differentially-private distributed SGD". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc., pp. 7564–7575. (Visited on 10/16/2020).

Agarwal, Saurabh et al. (Feb. 2021). "On the Utility of Gradient Compression in Distributed Training Systems". In: *arXiv e-prints*, arXiv:2103.00543, arXiv:2103.00543. arXiv: 2103.00543 [cs.DC].

Agustsson, Eirikur and Radu Timofte (July 2017). "NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*.

Ajanthan, Thalaiyasingam, Puneet K Dokania, et al. (2019). "Proximal mean-field for neural network quantization". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4871–4880.

Ajanthan, Thalaiyasingam, Kartik Gupta, et al. (2021). "Mirror descent view for neural network quantization". In: *International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 2809–2817.

Alistarh, Dan et al. (2017). "QSGD: Communication-efficient SGD via gradient quantization and encoding". In: *Advances in neural information processing systems* 30.

Anderson, Alexander G and Cory P Berg (2018). "The High-Dimensional Geometry of Binary Neural Networks". In: *International Conference on Learning Representations*.

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E Hinton (2016). "Layer normalization". In: *arXiv preprint arXiv:1607.06450*.

Baccelli, F. and P. Bremaud (2002). *Elements of Queueing Theory: Palm Martingale Calculus and Stochastic Recurrences*. Stochastic Modelling and Applied Probability. Springer Berlin Heidelberg. ISBN: 9783540660880. URL: https://books.google.fr/books?id=REcVXcy2sb0C.

Bach, Francis and Eric Moulines (2013). "Non-strongly-convex smooth stochastic approximation with convergence rate O (1/n)". In: *Advances in neural information processing systems* 26.

Bai, Yu, Yu-Xiang Wang, and Edo Liberty (2018). "ProxQuant: Quantized Neural Networks via Proximal Operators". In: *International Conference on Learning Representations*.

Banerjee, Kunal et al. (2020). "Exploring alternatives to softmax function". In: *arXiv preprint arXiv:2011.11538*.

Banner, Ron et al. (2018). "Scalable methods for 8-bit training of neural networks". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 5151–5159.

Beck, Amir and Marc Teboulle (2000). "Global optimality conditions for quadratic optimization problems with binary constraints". In: *SIAM Journal on Optimization* 11.1, pp. 179–188.

Bernstein, Jeremy et al. (2018). "signSGD: Compressed optimisation for non-convex problems". In: *International Conference on Machine Learning*. PMLR, pp. 560–569.

Bethge, Joseph et al. (2020). "MeliusNet: Can binary neural networks achieve mobilenet-level accuracy?" In: *arXiv preprint arXiv:2001.05936*.

Bevilacqua, Marco et al. (2012). "Low-Complexity Single-Image Super-Resolution based on Nonnegative Neighbor Embedding". In: *Proceedings of the British Machine Vision Conference*. BMVA Press, pp. 135.1–135.10. ISBN: 1-901725-46-4. DOI: http://dx.doi.org/10.5244/C.26.135.

Beznosikov, Aleksandr, Samuel Horváth, et al. (Feb. 2020). "On Biased Compression for Distributed Learning". In: *arXiv:2002.12410 [cs, math, stat]*. arXiv: 2002.12410. (Visited on 11/23/2020).

Beznosikov, Aleksandr, Sergey Samsonov, et al. (2023). "First Order Methods with Markovian Noise: from Acceleration to Variational Inequalities". In: *arXiv preprint arXiv:2305.15938*.

Bhalgat, Yash et al. (2020). "LSQ+: Improving low-bit quantization through learnable offsets and better initialization". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE Computer Society, pp. 2978–2985.

Bianco, Simone et al. (2018). "Benchmark Analysis of Representative Deep Neural Network Architectures". In: *IEEE Access* 6, pp. 64270–64277. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2877890.

Bonawitz, Keith, Hubert Eichner, et al. (2019). "Towards federated learning at scale: System design". In: *Proceedings of Machine Learning and Systems* 1, pp. 374–388.

Bonawitz, Keith, Vladimir Ivanov, et al. (2016). "Practical secure aggregation for federated learning on user-held data". In: *arXiv preprint arXiv:1611.04482*.

Borwein, Jonathan M. and Adrian S. Lewis (2006). *Convex analysis and nonlinear optimization*. Second. Vol. 3. CMS Books in Mathematics/Ouvrages de Mathématiques de la SMC. Theory and examples. Springer. ISBN: 0-387-29570-4. DOI: https://doi.org/10.1007/978-0-387-31256-9.

Bottou, Léon (1999). "On-line learning and stochastic approximations". In: DOI: 10.1017/CBO9780511569920.003.

Bradbury, James et al. (2018). *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. URL: http://github.com/google/jax.

Breiman, Leo (2001). "Random forests". In: *Machine learning* 45, pp. 5–32.

Bussieck, Michael R, Arne Stolbjerg Drud, and Alexander Meeraus (2003). "MINLPLib—a collection of test models for mixed-integer nonlinear programming". In: *INFORMS Journal on Computing* 15.1, pp. 114–119.

Caldas, Sebastian et al. (2018). "Expanding the reach of federated learning by reducing client resource requirements". In: *arXiv preprint arXiv:1812.07210*.

Canziani, Alfredo, Adam Paszke, and Eugenio Culurciello (2016). "An analysis of deep neural network models for practical applications". In: *arXiv preprint arXiv:1605.07678*.

Carreira-Perpinán, Miguel A and Yerlan Idelbayev (2017). "Model compression as constrained optimization, with application to neural nets. Part II: Quantization". In: *arXiv preprint arXiv:1707.04319*.

Cauchy, A (1847). "Méthode générale pour la résolution des systemes d'équations simultanées". In: *CR Acad. Sci. Paris* 25, pp. 536–538.

Chen, Hanting et al. (June 2020). "AdderNet: Do We Really Need Multiplications in Deep Learning?" In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Chen, Yu-Hsin et al. (2016). "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks". In: *IEEE Journal of Solid-State Circuits* 52.1, pp. 127–138.

Chen, Jianfei et al. (2020). "A statistical framework for low-bitwidth training of deep neural networks". In: *Advances in Neural Information Processing Systems* 33, pp. 883–894.

Chen, Liang-Chieh et al. (2017). "Rethinking atrous convolution for semantic image segmentation". In: *arXiv preprint arXiv:1706.05587*.

Chen, Yujing et al. (2020). "Asynchronous online federated learning for edge devices with non-iid data". In: *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, pp. 15–24.

Chen, Zheyi et al. (2021). "Towards asynchronous federated learning for heterogeneous edge-powered internet of things". In: *Digital Communications and Networks* 7.3, pp. 317–326.

Chmiel, Brian et al. (2021). "Logarithmic Unbiased Quantization: Simple 4-bit Training in Deep Learning". In: *arXiv preprint arXiv:2112.10769*.

Choi, Jungwook et al. (2018). "Bridging the accuracy gap for 2-bit quantized neural networks (QNN)". In: *arXiv preprint arXiv:1807.06964*.

Choukroun, Yoni et al. (2019). "Low-bit quantization of neural networks for efficient inference". In: *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. IEEE, pp. 3009–3018.

Cordts, Marius et al. (June 2016). "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Cortes, Corinna and Vladimir Vapnik (1995). "Support-vector networks". In: *Machine learning* 20, pp. 273–297.

Couillet, Romain, Denis Trystram, and Thierry Ménissier (2022). "The submerged part of the AI-Ceberg [Perspectives]". In: *IEEE Signal Processing Magazine* 39.5, pp. 10–17.

Courbariaux, Matthieu, Yoshua Bengio, and Jean-Pierre David (2015). "BinaryConnect: Training deep neural networks with binary weights during propagations". In: *Advances in Neural Information Processing Systems*, pp. 3123–3131.

Courbariaux, Matthieu, Itay Hubara, et al. (2016). "Binarized neural networks: Training neural networks with weights and activations constrained to +1 or -1". In: *arXiv preprint arXiv:1602.02830*.

Cybenko, George (1989). "Approximation by superpositions of a sigmoidal function". In: *Mathematics of control, signals and systems* 2.4, pp. 303–314.

Dai, Xinyan et al. (2019). "Hyper-sphere quantization: Communication-efficient sgd for federated learning". In: *arXiv preprint arXiv:1911.04655*.

Darabi, Sajad et al. (2018). "Bnn+: Improved binary network training". In: *arXiv preprint arXiv:1812.11800*.

Davis, Damek et al. (2020). "Stochastic Subgradient Method Converges on Tame Functions". In: *Found Comput Math* 20.1, pp. 119–154. DOI: 10.1007/s10208-018-09409-5.

Defazio, Aaron, Francis R Bach, and Simon Lacoste-Julien (2014). "SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives". In: *NIPS*.

Deng, Lei et al. (2020). "Model compression and hardware acceleration for neural networks: A comprehensive survey". In: *Proceedings of the IEEE* 108.4, pp. 485–532.

Deng, Li (2012). "The mnist database of handwritten digit images for machine learning research". In: *IEEE signal processing magazine* 29.6, pp. 141–142.

Ding, Ruizhou et al. (2019). "Regularizing activation distribution for training binarized deep networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11408–11417.

Esser, Steve K et al. (2015). "Backpropagation for energy-efficient neuromorphic computing". In: *Advances in neural information processing systems* 28.

Everingham, M. et al. (n.d.). *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

Faraone, Julian et al. (2018). "Syq: Learning symmetric quantization for efficient deep neural networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4300–4309.

Fei, Wen et al. (2022). "General Bitwidth Assignment for Efficient Deep Convolutional Neural Network Quantization". In: *IEEE Trans. Neural Netw. Learn. Syst.* 33.10, pp. 5253–5267. DOI: 10.1109/TNNLS.2021.3069886.

Fournarakis, Marios and Markus Nagel (2021). "In-Hindsight Quantization Range Estimation for Quantized Training". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3063–3070.

Fraboni, Yann et al. (2023). "A General Theory for Federated Optimization with Asynchronous and Heterogeneous Clients Updates". In: *Journal of Machine Learning Research* 24.110, pp. 1–43.

Frantar, Elias et al. (2022). "Gptq: Accurate post-training quantization for generative pre-trained transformers". In: *arXiv preprint arXiv:2210.17323*.

Freund, Yoav and Robert E. Schapire (1996). "Experiments with a New Boosting Algorithm". In: *Machine Learning, Proceedings of the Thirteenth International Conference (ICML '96), Bari, Italy, July 3-6, 1996*. Ed. by Lorenza Saitta. Morgan Kaufmann, pp. 148–156.

Fuchs, Eberhard, Gabriele Flügge, et al. (2014). "Adult neuroplasticity: more than 40 years of research". In: *Neural plasticity* 2014.

Gandikota, Venkata et al. (2021). "vqsgd: Vector quantized stochastic gradient descent". In: *International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 2197–2205.

Gao, Yanjie et al. (2020). "Estimating GPU memory consumption of deep learning models". In: *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1342–1352.

García-Martín, Eva et al. (2019). "Estimation of energy consumption in machine learning". In: *J Parallel Distr Com* 134, pp. 75–88.

Garey, Michael R and David S Johnson (1980). "Computers and intractability: A guide to the theory of NP-completeness". In: *Bull. Amer. Math. Soc* 3, pp. 898–904.

Gauss, Carl Friedrich (1809). *Theoria motus corporum coelestium in sectionibus conicis solem ambientium auctore Carolo Friderico Gauss*. sumtibus Frid. Perthes et IH Besser.

Geiping, Jonas et al. (2020). "Inverting gradients-how easy is it to break privacy in federated learning?" In: *Advances in Neural Information Processing Systems* 33, pp. 16937–16947.

Gersho, Allen and Robert M Gray (2012). *Vector quantization and signal compression*. Vol. 159. Springer Science & Business Media.

Gholami, Amir et al. (2021). "A survey of quantization methods for efficient neural network inference". In: *arXiv preprint arXiv:2103.13630*.

Gill, Philip E and Elizabeth Wong (2012). "Sequential quadratic programming methods". In: *Mixed integer nonlinear programming*. Springer, pp. 147–224.

Gong, Ruihao et al. (2019). "Differentiable soft quantization: Bridging full-precision and low-bit neural networks". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4852–4861.

Gorbunov, Eduard, Konstantin P Burlachenko, et al. (2021). "MARINA: Faster non-convex distributed learning with compression". In: *International Conference on Machine Learning*. PMLR, pp. 3788–3798.

Gorbunov, Eduard, Dmitry Kovalev, et al. (Oct. 2020). "Linearly Converging Error Compensated SGD". In: *arXiv:2010.12292 [cs, math]*. arXiv: 2010.12292. (Visited on 02/02/2021).

Gu, Shixiang et al. (2016). "MuProp: Unbiased Backpropagation for Stochastic Neural Networks." In: *ICLR (Poster)*.

Guo, Han et al. (2023). "Lq-lora: Low-rank plus quantized matrix decomposition for efficient language model finetuning". In: *arXiv preprint arXiv:2311.12023*.

Guo, Nianhui et al. (2022). "Join the High Accuracy Club on ImageNet with A Binary Neural Network Ticket". In: *arXiv preprint arXiv:2211.12933*.

Guo, Yunhui (2018). "A survey on methods and theories of quantized neural networks". In: *arXiv preprint arXiv:1808.04752*.

Gupta, Suyog et al. (July 2015). "Deep Learning with Limited Numerical Precision". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, pp. 1737–1746. URL: https://proceedings.mlr.press/v37/gupta15.html.

Halfin, Shlomo and Ward Whitt (1981). "Heavy-traffic limits for queues with many exponential servers". In: *Operations research* 29.3, pp. 567–588.

Han, Song, Huizi Mao, and William J. Dally (2015). "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding". In: *arXiv:1510.00149*. DOI: 10.48550/ARXIV.1510.00149. URL: https://arxiv.org/abs/1510.00149.

He, Kaiming et al. (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Hebb, Donald Olding (2005). *The Organization of Behavior: A Neuropsychological Theory*. Psychology press.

Helwegen, Koen et al. (2019). "Latent Weights Do Not Exist: Rethinking Binarized Neural Network Optimization". In: *Advances in Neural Information Processing Systems* 32, pp. 7533–7544.

Hettiarachchi, Don Lahiru Nirmal, Venkata Salini Priyamvada Davuluru, and Eric J Balster (2020). "Integer vs. floating-point processing on modern FPGA technology". In: *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, pp. 0606–0612.

Horst, Reiner and Hoang Tuy (2013). *Global optimization: Deterministic approaches*. Springer Science & Business Media.

Horváth, Samuel, Dmitry Kovalev, et al. (Apr. 2019). "Stochastic Distributed Learning with Gradient Quantization and Variance Reduction". In: *arXiv:1904.05115 [math]*. arXiv: 1904.05115.

Horváth, Samuel, Maziar Sanjabi, et al. (2022). "Fedshuffle: Recipes for better use of local work in federated learning". In: *arXiv preprint arXiv:2204.13169*.

Hou, Lu and James T Kwok (2018). "Loss-aware weight quantization of deep networks". In: *arXiv preprint arXiv:1802.08635*.

Howard, Andrew G et al. (2017). "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv preprint arXiv:1704.04861*.

Hu, Qinghao, Peisong Wang, and Jian Cheng (2018). "From hashing to cnns: Training binary weight networks via hashing". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1.

Huang, Cheng-Wei, Tim-Wei Chen, and Juinn-Dar Huang (2021). "All-You-Can-Fit 8-Bit Flexible Floating-Point Format for Accurate and Memory-Efficient Inference of Deep Neural Networks". In: *arXiv:2104.07329*. arXiv: 2104.07329 [cs.LG].

Huang, Jia-Bin, Abhishek Singh, and Narendra Ahuja (2015). "Single image super-resolution from transformed self-exemplars". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5197–5206.

Huba, Dzmitry et al. (2022). "Papaya: Practical, private, and scalable federated learning". In: *Proceedings of Machine Learning and Systems* 4, pp. 814–832.

Hubara, Itay et al. (2016). "Binarized neural networks". In: *Advances in neural information processing systems* 29.

Hubara, Itay et al. (2017). "Quantized neural networks: Training neural networks with low precision weights and activations". In: *The Journal of Machine Learning Research* 18.1, pp. 6869–6898.

Hunt, Earl B, Janet Marin, and Philip J Stone (1966). "Experiments in induction." In.

Ioffe, Sergey and Christian Szegedy (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. pmlr, pp. 448–456.

Islamov, Rustem, Mher Safaryan, and Dan Alistarh (2023). "AsGrad: A Sharp Unified Analysis of Asynchronous-SGD Algorithms". In: *arXiv preprint arXiv:2310.20452*.

Jackson, James R (1957). "Networks of waiting lines". In: *Operations research* 5.4, pp. 518–521.

Jackson, RRP (1954). "Queueing systems with phase type service". In: *Journal of the Operational Research Society* 5.4, pp. 109–120.

Jacob, Benoit et al. (2018). "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, pp. 2704–2713.

Jang, Eric, Shixiang Gu, and Ben Poole (2016). "Categorical Reparameterization with Gumbel-Softmax". In.

Jhunjhunwala, Divyansh et al. (2021). "Adaptive quantization of model updates for communication-efficient federated learning". In: *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pp. 3110–3114.

Ji, Yu and Lan Chen (2022). "FedQNN: a Computation-Communication Efficient Federated Learning Framework for IoT with Low-bitwidth Neural Network Quantization". In: *IEEE Internet of Things Journal*.

Jin, Qing et al. (2021). "F8Net: Fixed-Point 8-bit Only Multiplication for Network Quantization". In: *arXiv preprint arXiv:2202.05239*.

Johnson, Rie and Tong Zhang (Dec. 2013). "Accelerating stochastic gradient descent using predictive variance reduction". In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'13. Lake Tahoe, Nevada: Curran Associates Inc., pp. 315–323. (Visited on 04/23/2020).

Kairouz, Peter, Brendan McMahan, et al. (2021). "Practical and private (deep) learning without sampling or shuffling". In: *International Conference on Machine Learning*. PMLR, pp. 5213–5225.

Kairouz, Peter, H Brendan McMahan, et al. (2021). "Advances and open problems in federated learning". In: *Foundations and Trends® in Machine Learning* 14.1–2, pp. 1–210.

Karimireddy, Sai Praneeth, Satyen Kale, et al. (2020). "Scaffold: Stochastic controlled averaging for federated learning". In: *International Conference on Machine Learning*. PMLR, pp. 5132–5143.

Karimireddy, Sai Praneeth, Quentin Rebjock, et al. (2019). "Error Feedback Fixes SignSGD and other Gradient Compression Schemes". In: *CoRR* abs/1901.09847. arXiv: 1901.09847. URL: http://arxiv.org/abs/1901.09847.

Khan, Mohammad Emtiyaz and Håvard Rue (2021). "The Bayesian learning rule". In: *arXiv preprint arXiv:2107.04562*.

Kim, Jeonghoon et al. (2023). "Memory-Efficient Fine-Tuning of Compressed Large Language Models via sub-4-bit Integer Quantization". In: *arXiv preprint arXiv:2305.14152*.

Kim, Minje and Paris Smaragdis (2016). "Bitwise neural networks". In: *arXiv preprint arXiv:1601.06071*.

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Klenke, A. (2013). *Probability Theory: A Comprehensive Course*. Universitext. Springer London. ISBN: 9781447153603. URL: https://books.google.de/books?id=eTYMnQEACAAJ.

Koloskova, Anastasiia, Sebastian U Stich, and Martin Jaggi (2022). "Sharper convergence guarantees for asynchronous sgd for distributed and federated learning". In: *Advances in Neural Information Processing Systems* 35, pp. 17202–17215.

Konečnỳ, Jakub, Brendan McMahan, and Daniel Ramage (2015). "Federated optimization: Distributed optimization beyond the datacenter". In: *arXiv preprint arXiv:1511.03575*.

Konečný, Jakub et al. (Oct. 2016). "Federated Optimization: Distributed Machine Learning for On-Device Intelligence". In: *arXiv:1610.02527 [cs]*. arXiv: 1610.02527.

Krizhevsky, Alex, Geoffrey Hinton, et al. (2009). "Learning multiple layers of features from tiny images". In.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in Neural Information Processing Systems (NIPS)* 25, pp. 1097–1105.

Kwon, Hyoukjun et al. (2019). "Understanding reuse, performance, and hardware cost of DNN dataflow: A data-centric approach". In: *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 754–768.

Laplace, Pierre Simon (1814). *Théorie analytique des probabilités*. Courcier.

Laydevant, Jérémie et al. (2021). "Training Dynamical Binary Neural Networks with Equilibrium Propagation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4640–4649.

Le, Huu et al. (2021). "AdaSTE: An Adaptive Straight-Through Estimator to Train Binary Neural Networks". In: *arXiv preprint arXiv:2112.02880*.

Le, Ya and Xuan Yang (2015). "Tiny imagenet visual recognition challenge". In: *CS 231N* 7.7, p. 3.

Leconte, Louis (2024). *Boolean Logic as an Error feedback mechanism*. arXiv: 2401.16418 [stat.ML].

Leconte, Louis, Aymeric Dieuleveut, et al. (2021). "DoStoVoQ: Doubly Stochastic Voronoi Vector Quantization SGD for Federated Learning". In: URL: https://openreview.net/pdf?id=URc7gYBcjVn.

Leconte, Louis, Matthieu Jonckheere, et al. (2024). "Queuing dynamics of asynchronous Federated Learning". In: *International Conference on Artificial Intelligence and Statistics*. PMLR, tbd–tbd.

Leconte, Louis, Eric Moulines, et al. (2023). "Federated Boolean Neural Networks Learning". In: *2023 Eighth International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE, pp. 247–253.

Leconte, Louis, Van Minh Nguyen, and Eric Moulines (2023). "FAVAS: Federated AVeraging with ASynchronous clients". In: *arXiv preprint arXiv:2305.16099*.

Leconte, Louis, Sholom Schechtman, and Eric Moulines (2023). "AskewSGD: An annealed interval-constrained optimisation method to train quantized neural networks". In: *International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 3644–3663.

LeCun, Yann (1985). "A learning scheme for asymmetric threshold networks". In: *Proceedings of COGNITIVA* 85.537, pp. 599–604.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning". In: *nature* 521.7553, pp. 436–444.

LeCun, Yann, Léon Bottou, et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

Lee, Changhun et al. (2022). "INSTA-BNN: Binary Neural Network with INSTAnce-aware Threshold". In: *arXiv preprint arXiv:2204.07439*.

Legendre, Adrien Marie (1806). *Nouvelles méthodes pour la détermination des orbites des comètes: avec un supplément contenant divers perfectionnemens de ces méthodes et leur application aux deux comètes de 1805*. Courcier.

Leng, Cong et al. (2018). "Extremely low bit neural network: Squeeze the last bit out with admm". In: *Thirty-Second AAAI Conference on Artificial Intelligence*.

Li, Fengfu, Bo Zhang, and Bin Liu (2016). "Ternary weight networks". In: *arXiv preprint arXiv:1605.04711*.

Li, Hao et al. (2017). "Training quantized nets: A deeper understanding". In: *Advances in Neural Information Processing Systems* 30.

Li, Yuhang, Xin Dong, and Wei Wang (2019). "Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks". In: *arXiv preprint arXiv:1909.13144*.

Li, Zheng and Christopher De Sa (2019). *Dimension-Free Bounds for Low-Precision Training*. URL: https://openreview.net/forum?id=ryeX-nC9YQ.

Li, Zichong et al. (2021). "Rate-improved inexact augmented Lagrangian method for constrained nonconvex optimization". In: *International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 2170–2178.

Liang, Tailin et al. (2021). "Pruning and quantization for deep neural network acceleration: A survey". In: *Neurocomputing* 461, pp. 370–403.

Lin, Ji et al. (2023). "AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration". In: *arXiv preprint arXiv:2306.00978*.

Lin, Mingbao et al. (2020). "Rotated binary neural network". In: *Advances in neural information processing systems* 33, pp. 7474–7485.

Lin, Tao et al. (2019). "Don't Use Large Mini-batches, Use Local SGD". In: *International Conference on Learning Representations*.

Liu, Chunlei et al. (2022). "RB-Net: Training highly accurate and efficient binary neural networks with reshaped point-wise convolution and balanced activation". In: *IEEE Transactions on Circuits and Systems for Video Technology* 32.9, pp. 6414–6424.

Liu, Jianchun et al. (2021). "Adaptive Asynchronous Federated Learning in Resource-Constrained Edge Computing". In: *IEEE Transactions on Mobile Computing*.

Liu, Zechun, Zhiqiang Shen, et al. (2020). "Reactnet: Towards precise binary neural network with generalized activation functions". In: *European Conference on Computer Vision*. Springer, pp. 143–159.

Liu, Zechun, Baoyuan Wu, et al. (2018). "Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm". In: *Proceedings of the European conference on computer vision (ECCV)*, pp. 722–737.

Lu, Yucheng and Christopher De Sa (2020). "Moniqua: Modulo quantized communication in decentralized SGD". In: *International Conference on Machine Learning*. PMLR, pp. 6415–6425.

Maddison, C, A Mnih, and Y Teh (2017). "The concrete distribution: A continuous relaxation of discrete random variables". In: *Proceedings of the international conference on learning Representations*. International Conference on Learning Representations.

Makarenko, Maksim et al. (2022). "Adaptive Compression for Communication-Efficient Distributed Training". In: *arXiv preprint arXiv:2211.00188*.

Malekpourshahraki, Mojtaba et al. (Jan. 2022). "A Survey on Design Challenges of Scheduling Algorithms for Wireless Networks". In: *Int. J. Commun. Netw. Distrib. Syst.* 28.3, pp. 219–265. ISSN: 1754-3916. DOI: 10.1504/ijcnds.2022.122167. URL: https://doi.org/10.1504/ijcnds.2022.122167.

Mao, Yuzhu et al. (2022). "Communication-efficient federated learning with adaptive quantization". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 13.4, pp. 1–26.

Martin, David et al. (2001). "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics". In: *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*. Vol. 2. IEEE, pp. 416–423.

Martín Abadi et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: https://www.tensorflow.org/.

Martinez, Brais et al. (2019). "Training binary neural networks with real-to-binary convolutions". In: *International Conference on Learning Representations*.

Matsui, Yusuke et al. (2018). "A survey of product quantization". In: *ITE Transactions on Media Technology and Applications* 6.1, pp. 2–10.

McCulloch, Warren S and Walter Pitts (1943). "A logical calculus of the ideas immanent in nervous activity". In: *The Bulletin of Mathematical Biophysics* 5.4, pp. 115–133.

McDonnell, Mark D (2018). "Training wide residual networks for deployment using a single bit for each weight". In: *arXiv preprint arXiv:1802.08530*.

McMahan, Brendan et al. (2017). "Communication-efficient learning of deep networks from decentralized data". In: *Artificial intelligence and statistics*. PMLR, pp. 1273–1282.

Mei, Xinxin et al. (2014). "Benchmarking the memory hierarchy of modern GPUs". In: *IFIP International Conference on Network and Parallel Computing*. Springer, pp. 144–156.

Melekhov, Iaroslav et al. (2017). "Image-based localization using hourglass networks". In: *Proceedings of the IEEE international conference on computer vision workshops*, pp. 879–886.

Meng, Xiangming, Roman Bachmann, and Mohammad Emtiyaz Khan (2020). "Training binary neural networks using the bayesian learning rule". In: *International conference on machine learning*. PMLR, pp. 6852–6861.

Menon, Anil et al. (1996). "Characterization of a class of sigmoid functions with applications to neural networks". In: *Neural networks* 9.5, pp. 819–835.

Mishchenko, Konstantin, Francis Bach, et al. (2022). "Asynchronous sgd beats minibatch sgd under arbitrary delays". In: *arXiv preprint arXiv:2206.07638*.

Mishchenko, Konstantin, Eduard Gorbunov, et al. (June 2019). "Distributed Learning with Compressed Gradient Differences". In: *arXiv:1901.09269 [cs, math, stat]*. arXiv: 1901.09269.

Mishra, Asit and Debbie Marr (2017). "Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy". In: *arXiv preprint arXiv:1711.05852*.

Mishra, Asit, Eriko Nurvitadhi, et al. (2017). "WRPN: Wide reduced-precision networks". In: *arXiv preprint arXiv:1709.01134*.

Mitchell, John E, Panos M Pardalos, and Mauricio GC Resende (1998). "Interior point methods for combinatorial optimization". In: *Handbook of combinatorial optimization*. Springer, pp. 189–297.

Mnih, Andriy and Karol Gregor (2014). "Neural variational inference and learning in belief networks". In: *International Conference on Machine Learning*. PMLR, pp. 1791–1799.

Moons, Bert et al. (2017). "Minimum energy quantized neural networks". In: *2017 51st Asilomar Conference on Signals, Systems, and Computers*. IEEE, pp. 1921–1925.

Muehlebach, Michael and Michael I Jordan (2021). "On Constraints in First-Order Optimization: A View from Non-Smooth Dynamical Systems". In: *arXiv preprint arXiv:2107.08225*.

Murray, Walter and Kien-Ming Ng (2010). "An algorithm for nonlinear optimization problems with binary variables". In: *Computational optimization and applications* 47.2, pp. 257–288.

Nagel, Markus et al. (2021). "A white paper on neural network quantization". In: *arXiv preprint arXiv:2106.08295*.

Nesterov, Yurii (2013). *Introductory lectures on convex optimization: A basic course*. Vol. 87. Springer Science & Business Media.

Nguyen, Anh et al. (2021). "An analysis of state-of-the-art activation functions for supervised deep neural network". In: *2021 International Conference on System Science and Engineering (ICSSE)*. IEEE, pp. 215–220.

Nguyen, John et al. (2022). "Federated learning with buffered asynchronous aggregation". In: *International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 3581–3607.

Nguyen, Van Minh (2023). "Boolean Variation and Boolean Logic BackPropagation". In: *arXiv preprint arXiv:2311.07427*. arXiv: 2311.07427 [cs.LG].

Nguyen, Van Minh and Louis Leconte (2022). *Apparatus and method for training binary deep neural networks*. URL: https://patents.google.com/patent/WO2023217370A1/.

Nie, Guhong et al. (2022). "Binary Neural Networks as a general-propose compute paradigm for on-device computer vision". In: *arXiv preprint arXiv:2202.03716*. DOI: 10.48550/ARXIV.2202.03716. URL: https://arxiv.org/abs/2202.03716.

Pagès, Gilles and Jacques Printems (2003). "Optimal quadratic quantization for numerics: the Gaussian case". In: *Monte Carlo methods and applications* 9.2, pp. 135–165.

Pagès, Gilles and Benedikt Wilbertz (2018). "Sharp rate for the dual quantization problem". In: *Séminaire de Probabilités XLIX*. Vol. 2215. Lecture Notes in Math. Springer, Cham, pp. 405–454.

Panigrahi, Abhishek et al. (2019). "Non-Gaussianity of stochastic gradient noise". In: *arXiv preprint arXiv:1910.09626*.

Paszke, Adam et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Peters, Jorn WT and Max Welling (2018). "Probabilistic binary neural networks". In: *arXiv preprint arXiv:1809.03368*.

Philippenko, Constantin and Aymeric Dieuleveut (Nov. 2020). "Artemis: tight convergence guarantees for bidirectional compression in Federated Learning". In: *arXiv:2006.14591 [cs, stat]*. arXiv: 2006.14591. (Visited on 11/11/2020).

Polino, Antonio, Razvan Pascanu, and Dan Alistarh (2018). "Model compression via distillation and quantization". In: *arXiv preprint arXiv:1802.05668*.

Qin, Haotong et al. (2020). "Binary neural networks: A survey". In: *Pattern Recognition* 105, p. 107281.

Qu, Linping, Shenghui Song, and Chi-Ying Tsui (2021). "FedDQ: Communication-Efficient Federated Learning with Descending Quantization". In: *arXiv preprint arXiv:2110.02291*.

Raiko, Pekka et al. (2015). "Techniques for Learning Binary Stochastic Feedforward Neural Networks". In: *International Conference on Learning Representations*, pp. 1–10.

Rastegari, Mohammad et al. (2016). "XNOR-Net: Imagenet classification using binary convolutional neural networks". In: *European Conference on Computer Vision*. Springer, pp. 525–542.

Robbins, Herbert and Sutton Monro (1951). "A stochastic approximation method". In: *The annals of mathematical statistics*, pp. 400–407.

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). "U-net: Convolutional networks for biomedical image segmentation". In: *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*. Springer, pp. 234–241.

Roth, Wolfgang et al. (2019). "Training discrete-valued neural networks with sign activations using weight distributions". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, pp. 382–398.

Russakovsky, Olga et al. (2015). "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3, pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

Safaryan, Mher, Egor Shulgin, and Peter Richtárik (2020). "Uncertainty principle for communication compression in distributed and federated learning and the search for an optimal compressor". In: *arXiv preprint arXiv:2002.08958*.

Saha, Rajarshi, Mert Pilanci, and Andrea J Goldsmith (2021). "Distributed Learning and Democratic Embeddings: Polynomial-Time Source Coding Schemes Can Achieve Minimax Lower Bounds for Distributed Gradient Descent under Communication Constraints". In: *arXiv preprint arXiv:2103.07578*.

Sakr, Charbel and Naresh Shanbhag (2018). "Per-tensor fixed-point quantization of the back-propagation algorithm". In: *arXiv preprint arXiv:1812.11732*.

Sard, Arthur (1942). "The measure of the critical values of differentiable maps". In: *Bulletin of the American Mathematical Society* 48.12, pp. 883–890. DOI: bams/1183504867. URL: https://doi.org/.

Schmidhuber, Jürgen (1989). "The neural bucket brigade". In: *Connectionism in perspective*, pp. 439–446.

Seide, F. et al. (Jan. 2014). "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs". In: pp. 1058–1062.

Serfozo, R. (1999). *Introduction to Stochastic Networks*. Stochastic Modelling and Applied Probability. Springer New York. ISBN: 9780387987736. URL: https://books.google.fr/books?id=9IuVTP_GpfgC.

Shannon, Claude E (1948). "A mathematical theory of communication". In: *The Bell system technical journal* 27.3, pp. 379–423.

Shao, Yakun Sophia and David Brooks (2013). "Energy characterization and instruction-level energy model of Intelś Xeon Phi processor". In: *International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 389–394. DOI: 10.1109/ISLPED.2013.6629328.

Shekhovtsov, Alexander (2021). "Bias-variance tradeoffs in single-sample binary gradient estimators". In: *DAGM German Conference on Pattern Recognition*. Springer, pp. 127–141.

Shekhovtsov, Alexander and Viktor Yanush (2021). "Reintroducing straight-through estimators as principled methods for stochastic binary networks". In: *DAGM German Conference on Pattern Recognition*. Springer, pp. 111–126.

Sim, Jaehyeong, Somin Lee, and Lee-Sup Kim (2019). "An energy-efficient deep convolutional neural network inference processor with enhanced output stationary dataflow in 65-nm CMOS". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28.1, pp. 87–100.

Simonyan, Karen and Andrew Zisserman (2014). "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556*.

Smith, Virginia et al. (2017). "Federated multi-task learning". In: *Advances in neural information processing systems* 30.

Stavrinides, Georgios L. and Helen D. Karatza (2018). "Scheduling Techniques for Complex Workloads in Distributed Systems". In: *Proceedings of the 2nd International Conference on Future Networks and Distributed Systems*. ICFNDS '18. Amman, Jordan: Association for Computing Machinery. ISBN: 9781450364287. DOI: 10.1145/3231053.3231087. URL: https://doi.org/10.1145/3231053.3231087.

Stich, Sebastian U, Jean-Baptiste Cordonnier, and Martin Jaggi (2018). "Sparsified SGD with Memory". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc., pp. 4447–4458. (Visited on 09/07/2020).

Stich, Sebastian U. (May 2019). "Local SGD Converges Fast and Communicates Little". In: *arXiv:1805.09767 [cs, math]*. arXiv: 1805.09767. (Visited on 03/26/2020).

Stock, Pierre et al. (2020). "And the Bit Goes Down: Revisiting the Quantization of Neural Networks". In: *ICLR 2020-Eighth International Conference on Learning Representations*, pp. 1–11.

Strubell, Emma, Ananya Ganesh, and Andrew McCallum (July 2019). "Energy and Policy Considerations for Deep Learning in NLP". In: *Proceedings of the 57th Annual Meeting of the Associa-*

tion for Computational Linguistics. Florence, Italy: Association for Computational Linguistics, pp. 3645–3650. DOI: 10.18653/v1/P19-1355. URL: https://aclanthology.org/P19-1355.

Sun, Xiao, Naigang Wang, Chia-Yu Chen, Jiamin Ni, Ankur Agrawal, Xiaodong Cui, Swagath Venkataramani, Kaoutar El Maghraoui, et al. (2020). "Ultra-low precision 4-bit training of deep neural networks". In: *Advances in Neural Information Processing Systems* 33, pp. 1796–1807.

Sun, Xiao, Naigang Wang, Chia-Yu Chen, Jiamin Ni, Ankur Agrawal, Xiaodong Cui, Swagath Venkataramani, Kaoutar El Maghraoui, et al. (2020). "Ultra-Low Precision 4-bit Training of Deep Neural Networks". In: *Advances in Neural Information Processing Systems 33 pre-proceedings (NeurIPS 2020)*. URL: https://proceedings.neurips.cc/paper/2020/hash/13b919438259814cd5be8cb45877d577-Abstract.html.

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction*. MIT press.

Sze, Vivienne et al. (2017). "Efficient processing of deep neural networks: A tutorial and survey". In: *Proc. IEEE* 105.12, pp. 2295–2329. URL: https://ieeexplore.ieee.org/document/8114708.

— (2020). "How to evaluate deep neural network processors: Tops/w (alone) considered harmful". In: *IEEE Solid-State Circuits Mag.* 12.3, pp. 28–41.

Timofte, Radu et al. (July 2017). "NTIRE 2017 Challenge on Single Image Super-Resolution: Methods and Results". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*.

Toghani, Mohammad Taha and César A Uribe (2022). "Unbounded Gradients in Federated Leaning with Buffered Asynchronous Aggregation". In: *arXiv preprint arXiv:2210.01161*.

Touvron, Hugo et al. (2023). "Llama: Open and efficient foundation language models". In: *arXiv preprint arXiv:2302.13971*.

Tu, Zhijun et al. (2022). "Adabin: Improving binary neural networks with adaptive binary sets". In: *European Conference on Computer Vision*. Springer, pp. 379–395.

Tyurin, Alexander and Peter Richtárik (2022). "DASHA: Distributed nonconvex optimization with communication compression, optimal oracle complexity, and no client synchronization". In: *arXiv preprint arXiv:2202.01268*.

— (2023). "Optimal Time Complexities of Parallel Stochastic Optimization Methods Under a Fixed Computation Model". In: *arXiv preprint arXiv:2305.12387*.

Umuroglu, Yaman et al. (2017). "FINN: A framework for fast, scalable binarized neural network inference". In: *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 65–74.

Van Kreveld, LR, JL Dorsman, and MRH Mandjes (2021). "Scaling limits for closed product-form queueing networks". In: *Performance Evaluation* 151, p. 102220.

Vaswani, Ashish et al. (2017). "Attention is all you need". In: *Advances in Neural Information Processing Systems 30* 30.

Vaswani, Sharan, Francis Bach, and Mark Schmidt (2019). "Fast and faster convergence of SGD for over-parameterized models and an accelerated perceptron". In: *The 22nd international conference on artificial intelligence and statistics*. PMLR, pp. 1195–1204.

Vogels, Thijs, Sai Praneeth Karimireddy, and Martin Jaggi (2019). "PowerSGD: Practical Low-Rank Gradient Compression for Distributed Optimization". In: *Advances in Neural Information Processing Systems* 32, pp. 14259–14268.

Wang, Erwei et al. (2021). "Enabling binary neural network training on the edge". In: *Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning*, pp. 37–38.

Wang, Hongyi, Saurabh Agarwal, and Dimitris Papailiopoulos (2021). "Pufferfish: Communication-efficient Models At No Extra Cost". In: *Proceedings of Machine Learning and Systems* 3.

Wang, Hongyi, Scott Sievert, et al. (2018). "ATOMO: Communication-efficient Learning via Atomic Sparsification". In: *Advances in Neural Information Processing Systems* 31, pp. 9850–9861.

Wang, Hongyu et al. (2023). "BitNet: Scaling 1-bit Transformers for Large Language Models". In: *arXiv preprint arXiv:2310.11453*.

Wang, Jianyu, Zachary Charles, et al. (2021). "A field guide to federated optimization". In: *arXiv preprint arXiv:2107.06917*.

Wang, Jianyu, Qinghua Liu, et al. (2020). "Tackling the objective inconsistency problem in heterogeneous federated optimization". In: *Advances in neural information processing systems* 33, pp. 7611–7623.

Wang, Naigang et al. (2018). "Training deep neural networks with 8-bit floating point numbers". In: *Advances in neural information processing systems* 31.

Wang, Qiyuan et al. (2022). "AsyncFedED: Asynchronous Federated Learning with Euclidean Distance based Adaptive Weight Aggregation". In: *arXiv preprint arXiv:2205.13797*.

Wang, Shaobo et al. (2021). "High-Precision Binary Object Detector Based on a BSF-XNOR Convolutional Layer". In: *IEEE Access* 9, pp. 106169–106180.

Wang, Sihua et al. (2022). "Performance Optimization for Variable Bitwidth Federated Learning in Wireless Networks". In: *arXiv preprint arXiv:2209.10200*.

Wang, Yikai et al. (June 2023). "Compacting Binary Neural Networks by Sparse Kernel Selection". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 24374–24383.

Wang, Ziwei et al. (2019). "Learning channel-wise interactions for binary convolutional neural networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 568–577.

Wangni, Jianqiao et al. (2018). "Gradient Sparsification for Communication-Efficient Distributed Optimization". en. In: *Advances in Neural Information Processing Systems* 31, pp. 1299–1309. (Visited on 02/02/2021).

Wiedemann, Simon et al. (2020). "Dithered backprop: A sparse and quantized backpropagation algorithm for more efficient deep neural network training". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 720–721.

Woodworth, Blake et al. (2020). "Is local SGD better than minibatch SGD?" In: *International Conference on Machine Learning*. PMLR, pp. 10334–10343.

Xie, Cong, Sanmi Koyejo, and Indranil Gupta (2019). "Asynchronous federated optimization". In: *arXiv preprint arXiv:1903.03934*.

Xing, Xingrun et al. (2022). "Towards Accurate Binary Neural Networks via Modeling Contextual Dependencies". In: *arXiv preprint arXiv:2209.01404*.

Xu, An, Zhouyuan Huo, and Heng Huang (2020). "Optimal gradient quantization condition for communication-efficient distributed training". In: *arXiv preprint arXiv:2002.11082*.

Xu, Chenhao et al. (2021). "Asynchronous federated learning on heterogeneous devices: A survey". In: *arXiv preprint arXiv:2109.04269*.

Xu, Hang et al. (2020). *Compressed communication for distributed deep learning: Survey and quantitative evaluation*. Tech. rep.

Yamamoto, Kohei (2021). "Learnable Companding Quantization for Accurate Low-bit Neural Networks". In: *arXiv:2103.07156*. arXiv: 2103.07156 [cs.CV].

Yang, Tien-Ju, Yu-Hsin Chen, Joel Emer, et al. (Oct. 2017). "A method to estimate the energy consumption of deep neural networks". In: *2017 51st Asilomar Conference on Signals, Systems, and Computers*, pp. 1916–1920. DOI: 10.1109/ACSSC.2017.8335698.

Yang, Tien-Ju, Yu-Hsin Chen, and Vivienne Sze (2017). "Designing energy-efficient convolutional neural networks using energy-aware pruning". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5687–5695.

Yang, Xuan et al. (2020). "Interstellar: Using halide's scheduling language to analyze dnn accelerators". In: *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 369–383.

Yang, Yukuan et al. (2022). "Towards Efficient Full 8-bit Integer DNN Online Training on Resource-limited Devices without Batch Normalization". In: *Neurocomputing*. arXiv: 2105.13890 [cs.LG].

Yang, Yuzhi, Zhaoyang Zhang, and Qianqian Yang (2021). "Communication-efficient federated learning with binary neural networks". In: *IEEE Journal on Selected Areas in Communications* 39.12, pp. 3836–3850.

Yin, Penghang et al. (2018). "Binaryrelax: A relaxation approach for training deep neural networks with quantized weights". In: *SIAM Journal on Imaging Sciences* 11.4, pp. 2205–2223.

Yu, Mingchao et al. (2018). "GradiVeQ: Vector Quantization for Bandwidth-Efficient Gradient Aggregation in Distributed CNN Training". In: *Advances in Neural Information Processing Systems* 31, pp. 5123–5133.

Zagoruyko, Sergey and Nikos Komodakis (2016). "Wide residual networks". In: *arXiv preprint arXiv:1605.07146*.

Zakerinia, Hossein et al. (2022). "QuAFL: Federated Averaging Can Be Both Asynchronous and Communication-Efficient". In: *arXiv preprint arXiv:2206.10032*.

Zeyde, Roman, Michael Elad, and Matan Protter (2012). "On single image scale-up using sparse-representations". In: *Curves and Surfaces: 7th International Conference, Avignon, France, June 24-30, 2010, Revised Selected Papers 7*. Springer, pp. 711–730.

Zhang, Dongqing et al. (2018). "Lq-nets: Learned quantization for highly accurate and compact deep neural networks". In: *Proceedings of the European conference on computer vision (ECCV)*, pp. 365–382.

Zhang, Yichi, Zhiru Zhang, and Lukasz Lew (2022). "PokeBNN: A binary pursuit of lightweight accuracy". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12475–12485.

Zhang, Zhaoyang et al. (2021). "Differentiable Dynamic Quantization with Mixed Precision and Adaptive Resolution". In: *International Conference on Machine Learning*. PMLR, pp. 12546–12556.

Zhao, Wenyu et al. (2020). "A Review of Recent Advances of Binary Neural Networks for Edge Computing". In: *IEEE Journal on Miniaturization for Air and Space Systems*.

Zhao, Yue et al. (2018). "Federated learning with non-iid data". In: *arXiv preprint arXiv:1806.00582*.

Zhou, Aojun et al. (2017). "Incremental network quantization: Towards lossless cnns with low-precision weights". In: *arXiv preprint arXiv:1702.03044*.

Zhou, Shuchang et al. (2016). "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients". In: *arXiv preprint arXiv:1606.06160*.

Zhou, Yiren et al. (2018). "Adaptive quantization for deep neural network". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1.

Zhu, Chenzhuo et al. (2016). "Trained ternary quantization". In: *arXiv preprint arXiv:1612.01064*.

Appendix A

Supplementary material of Chapter 2

A.1 Proofs of Section 2.3

A.1.1 Preliminaries

Absolutely continuous curves. We say that a curve $\mathsf{y} : \mathbb{R}_+ \to \mathbb{R}^d$ is absolutely continuous (a.c.) if there is a curve $\mathsf{z} : \mathbb{R}_+ \to \mathbb{R}^d$, locally Lebesgue integrable, such that for every $t \geq 0$,

$$\mathsf{y}(t) - \mathsf{y}(0) = \int_0^t \mathsf{z}(u)\mathrm{d}u\,. \tag{A.1}$$

In this case, it holds that for almost every $t \geq 0$, y is differentiable and $\dot{\mathsf{y}}(t) = \mathsf{z}(t)$.

Tangent and normal cones. Let $C \subset \mathbb{R}^d$ be a closed set. For $w \in C$, the tangent cone of C to w, denoted by $T_C(w)$, is the set of vectors $v \in \mathbb{R}^d$ for which there exist $t_k \downarrow 0$ and $w_k \to w$, $w_k \in C$, such that $(w_k - w)/t_k \to v$. The normal cone of C at w, denoted $N_C(w)$, is the set of vectors $u \in \mathbb{R}^d$ such that for any $v \in T_C(w)$, $u^\top v \leq 0$. If $w \notin C$, then by convention $T_C(w), N_C(w) = \emptyset$.

The Mangasarian-Fromovitz constraint qualification (MFCQ) condition. Consider the case where $C = \{w \in \mathbb{R}^d : h(w) \geq 0\}$, for a smooth function $h : \mathbb{R}^d \to \mathbb{R}^{n_h}$. Denote $\mathrm{I}(w) = \{i \in \{1,\dots,n_h\}, h_i(w) \leq 0\}$ as the set of active constraints. We say that the MFCQ condition holds at $w \in \mathbb{R}^d$ if there exists $v \in \mathbb{R}^d$ such that $\nabla h_i(w)^\top v \geq 0$ for all $i \in \mathrm{I}(w)$. If the MFCQ condition holds at $w \in C$, then we can write down $T_C(w) = \{v \in \mathbb{R}^d, \nabla h_i(w)^\top v \geq 0, \text{for all } i \in \mathrm{I}(w)\}$ and $N_C(w) = \{-\sum_{i=1}^{n_g} \lambda_i \nabla h_i(w), \lambda_i \in \mathbb{R}_+ \text{ and } \lambda_i = 0 \text{ if } i \notin \mathrm{I}(w)\}$ (see, e.g., Borwein and Lewis 2006, Section 7.2). We might notice here, that in the context of Theorem 5 the MFCQ condition holds at every $w \in C_\epsilon$.

Differential inclusion. Consider a closed set $C \subset \mathbb{R}^d$ and $f : \mathbb{R}^d \to \mathbb{R}$ a smooth function. An essential ingredient of our proof will be the following differential inclusion (DI):

$$\dot{\mathsf{y}}(t) \in -\nabla f(\mathsf{y}(t)) - N_C(\mathsf{y}(t))\,. \tag{A.2}$$

We say that an a.c. curve $\mathsf{y} : \mathbb{R}_+ \to C$ is a solution to this DI if the inclusion holds for almost every $t \geq 0$. We say that f is a Lyapunov function for the set $\mathcal{Z} := \{w \in \mathbb{R}^d : 0 \in -\nabla f(w) - N_C(w)\}$ if for any such curve:

$$\text{for all } t > 0\,, \quad f(\mathsf{y}(t)) \leq f(\mathsf{y}(0))\,, \tag{A.3}$$

with strict inequality as soon as $\mathsf{y}(0) \notin \mathcal{Z}$. We have the following lemma.

Lemma 19. *Assume that MFCQ holds at every $w \in C$. Then f is a Lyapunov function for the DI* (A.2) *and the set $\mathcal{Z}$.*

Proof. The assumption that MFCQ holds at every $w \in C$ implies that C is Clarke regular (i.e. if $(w_k, u_k) \to (w, u) \in C \times \mathbb{R}^d$ with $(w_k, u_k) \in C \times N_C(w_k)$, then $u \in N_C(w)$). As shown in (Sections 5 and 6 of Davis et al. 2020), this implies that for almost every $t \geq 0$ and every $v \in N_C(\mathsf{y}(t))$, $\dot{\mathsf{y}}(t)^\top v = 0$. Therefore, for almost every $t \geq 0$,

$$\frac{\mathrm{d}}{\mathrm{d}t} f(\mathsf{y}(t)) = \nabla f(\mathsf{y}(t))^\top \dot{\mathsf{y}}(t) \tag{A.4}$$

$$\in -\|\dot{\mathsf{y}}(t)\|^2 - \dot{\mathsf{y}}(t)^\top \mathsf{v}(t) = -\|\dot{\mathsf{y}}(t)\|^2\,, \tag{A.5}$$

where $\mathsf{v}(t) = \nabla f(\mathsf{y}(t)) - \dot{\mathsf{y}}(t) \in -N_C(\mathsf{y}(t))$. This shows that $\mathsf{y}(t) - \mathsf{y}(0) = -\int_0^t \|\dot{\mathsf{y}}(u)\|^2 \,\mathrm{d}u$, which, by closedness of $\mathcal{Z}$, implies our statement. □

In Section 2.3 the set of interest will be C_ϵ. It can be easily seen that under the assumptions of Theorem 5 the MFCQ condition is satisfied at every $w \in C_\epsilon$. Thus, Lemma 19 implies that, in this context, f is a Lyapunov function for the DI: $\dot{\mathsf{y}}(t) \in -f(\mathsf{y}(t)) - N_{C_\epsilon}(\mathsf{y}(t))$.

Discrete approximations of differential inclusions. The idea of our proof is to apply the results of Davis et al. 2020 on the stochastic approximation of differential inclusions to our setting. To this end, we consider an $\mathbb{R}^d$-valued sequence (y_k) constructed as follows:

$$y_{k+1} = y_k - \eta_k \nabla f(y_k) + \eta_k \rho_{k+1} - \eta_k u_k\,, \tag{A.6}$$

where (η_k) is a sequence of positive step-sizes and $(\rho_k), (u_k)$ are some $\mathbb{R}^d$-valued sequences. Here, u_k represent some approximation of an element of $N_C(y_k)$, and ρ_{k+1} some (stochastic or deterministic) perturbation. In this chapter we adopt the suscript notation "k" instead of "t", not to confuse the discrete dynamics with the continuous of the DI. Therefore, (y_k) might be seen as an Euler-like discretization of the DI (A.2).

The following proposition follows from a general result of Davis et al. 2020, Theorem 3.2. We state it, applied to our particular case.

Theorem 20. *Assume that:*

1. *The sequence (η_k) satisfies $\sum_{k=0}^{+\infty} \eta_k = +\infty$ and $\sum_{k=0}^{+\infty} \eta_k^2 < +\infty$.*
2. *The sequence $\left(\sum_{j=0}^{k} \eta_j \rho_{j+1}\right)$ converges.*
3. *The sequence (y_k, u_k) is bounded.*
4. *If y_{k_j} is a subsequence such that $y_{k_j} \to y_\infty$, then $y_\infty \in C$ and the distance between $-N_C(y_\infty) - \nabla f(y_\infty)$ and $-1/n \sum_{j=1}^{n} \{\nabla f(y_{k_j}) + u_{k_j}\}$ goes to zero.*
5. *f is a Lyapunov function for the DI* (A.2)*.*
6. *The set $f(\mathcal{Z})$ is of empty interior.*

Then, $f(y_k)$ converges and $\limsup_{k \to +\infty} d(y_k, \mathcal{Z}) = 0$.

Proof. Apply Davis et al. 2020, Theorem 3.2, with $G = -\nabla f - N_C$ and $\phi = f$. □

Lemma 21. *We can replace the 4-th assumption in Theorem 20 by the following assumption: if (y_∞, u_∞) is a cluster point of (y_k, u_k), then $u_\infty \in N_C(y_\infty)$.*

Proof. If (y_{k_j}) is a subsequence such that $y_{k_j} \to y_\infty$, then $1/n\sum_{j=1}^n -\nabla f(y_{k_j}) \to_{n\to\infty} -\nabla f(y_\infty)$. Furthermore, for any $m \geq 0$, we can write:

$$\frac{1}{n}\sum_{j=1}^{n} u_{k_j} = \frac{1}{n}\sum_{j=1}^{m} u_{k_j} + \frac{n-m}{n}\left(\frac{1}{n-m}\sum_{j=m+1}^{n} u_{k_j}\right). \tag{A.7}$$

By the Caratheodory theorem, we can write $1/(n-m)\sum_{j=m}^n u_{k_j} = \sum_{i=1}^{d+1} \lambda_{m,n,i} u_{m,n,i}$, where $\lambda_{m,n,i} \geq 0$, $\sum_{i=1}^{d+1} \lambda_{m,n,i} = 1$ and $u_{m,n,i} \in \{u_{k_{m+1}}, \dots, u_{k_n}\}$. Denote $\mathcal{C} \subset N_C(y_\infty)$ the set of cluster points of the sequence u_{k_j}. Since the sequence (u_k) is bounded, for each $i \in \{1, \dots, d+1\}$, we can extract a convergent sequence from $(\lambda_{m,n,i}, u_{m,n,i})$ that converges to $(\lambda_m(i), u_m(i))$, with $u_m(i) \in \mathcal{C} \cup \bigcup_{j=m+1}^{+\infty}\{u_{k_j}\}$. Thus, $1/n\sum_{j=1}^n u_{k_j} \to \sum_{i=1}^{d+1} \lambda_m(i) u_m(i)$. As a consequence, we can write:

$$\lim_{n\to+\infty} 1/n \sum_{j=1}^{n} u_{k_j} = \lim_{m\to\infty} \sum_{i=1}^{d+1} \lambda_m(i) u_m(i). \tag{A.8}$$

For each $i \in \{1, \dots, d+1\}$, the sequences $(\lambda_m(i))_{m\geq 0}, (u_m(i))_{m\geq 0}$ are bounded. Therefore, up to an extraction of a subsequence, we can assume that they converge to some $\lambda(i), u(i)$. Notice that $u(i) \in \mathcal{C} \subset N_C(y_\infty)$. Therefore, $1/n\sum_{j=1}^n u_{k_j}$ converges to a convex combination of elements of $N_C(y_\infty)$. By convexity of $N_C(y_\infty)$ this implies that $1/n\sum_{j=1}^n u_{k_j}$ converges to an element of $N_C(y_\infty)$.

□

The following lemma provides a condition under which $f(\mathcal{Z})$ has an empty interior.

Lemma 22. *Assume that $f : \mathbb{R}^d \to \mathbb{R}$ is d-times continuously differentiable and that $C = [a_1, b_1] \times \cdots \times [a_d, b_d]$, where for $1 \leq i \leq d$, a_i, b_i are some real numbers. Consider $\mathcal{Z} = \{y \in \mathbb{R}^d : 0 \in -\nabla f(y) - N_C(y)\}$. It holds that $f(\mathcal{Z})$ is of empty interior.*

Proof. Denote $\mathring{C}$ as the interior of C. The fact that $f(\mathcal{Z} \cap \mathring{C})$ has an empty interior is a consequence of Sard's theorem and the fact that f is d-times differentiable (see Sard 1942). We now show that the image of f of any m-dimensional boundary of C intersected by $\mathcal{Z}$ also has an empty interior. Consider $m > 0$, and fix $m - d$ coordinates of C as $c_{m+1}, \dots, c_d$, where c_i is equal to a_i or b_i, and denote $C_m = (a_1, b_1) \times (a_2, b_2) \cdots \times (a_m, b_m) \times \{c_{m+1}\} \times \cdots \times \{c_d\}$. Note that if $y \in \mathcal{Z} \cap C_m$, then the m first coordinates of $\nabla f(y)$ are zero. Thus, if we call f_m the restriction of f to C_m, then $f_m : C_m \to \mathbb{R}$ is d times differentiable and $\mathcal{Z} \cap C_m$ is included in its set of critical points. Applying Sard's theorem to f_m, we obtain that $f(\mathcal{Z} \cap C_m)$ has an empty interior. Since C can be written as a union of these C_m, this completes the proof. □

A.1.2 A proof of Theorem 5

First we need to prove that the cluster point of the iterates w_∞ belongs to the constraints set C_ϵ.

Lemma 23. *Under assumptions of Theorem 5 it holds that* $\limsup_{k\to\infty} \mathrm{d}(w_k, C_\epsilon) = 0$ *almost surely.*

Sketch of proof. The detailed proof is given in the following section Appendix A.1.3. The main idea is that for any $i \in \{1,\dots,d\}$, if $\psi_\epsilon^i(w_k^i) < 0$ (i.e. w_k^i is outside of the constraints), then w_k^i is constantly pushed to the closest interval. Thus, the non-convergence might happen if and only if w_k^i visits one of the interval infinitely often. However, due to the fact, that η_k decreases to zero and that ∇f_j is bounded, this implies, for k large enough, that w_k^i will never leave the "region of attraction" of this interval (it will be kept at a distance of order η_k to this interval) and thus converge to it. □

Proof of Theorem 5. Our goal is to apply Theorem 20 and, hence, verify its assumptions. By a standard Martingale argument it holds that the sequence $\sum_{j=0}^k \eta_j \rho_{j+1}$, almost surely, converges to a finite random variable (a short proof of this result is given in Appendix A.1.3). Consider a realization for which $\sum_{j=0}^\infty \eta_j \rho_{j+1} < \infty$. Let (w_∞, u_∞) be a cluster point of the sequence (w_k, u_k) and let $(k_j)_{j\geq 0}$ be a subsequence such that $\lim_{j\to+\infty}(w_{k_j}, u_{k_j}) = (w_\infty, u_\infty)$. Lemma 23 shows that $w_\infty \in C_\epsilon$. Since $\sup_{k\geq k_{0,\epsilon}} |\lambda_k^i| < +\infty$, we can extract a subsequence from k_j, and assume that $\lambda_{k_j} \to \lambda$. Thus, $u_\infty^i = -\lambda^i \psi_\epsilon'(w_\infty^i)$. Since all of the $\lambda_{k_j}^i$ are positive, it holds that $\lambda^i \geq 0$. Moreover, notice that if $\psi_\epsilon(w_\infty^i) > 0$, then, for j large enough, $\psi_\epsilon(w_{k_j}^i) > 0$ and, therefore, $\lambda_{k_j}^i = 0$. Hence, for $i \notin \mathrm{I}(w_\infty)$, $\lambda^i = 0$. This shows $u_\infty \in N_{C_\epsilon}(w_\infty)$. As shown in Lemma 19, f is a Lyapunov function for the DI: $\dot{\mathrm{y}}(t) \in -\nabla f(\mathrm{y}(t)) - N_{C_\epsilon}(\mathrm{y}(t))$. In Lemma 22 we show that $f(\mathcal{Z})$ is of empty interior. Thus, with the help of Lemmas 23 and 24, the assumptions of Theorem 20 are satisfied, which concludes the proof. □

A.1.3 A martingale result and proof of Lemma 23

We first establish a result on the convergence of the weighted sequence of perturbations.

Lemma 24. *Assume A5-A6. Then, almost surely, $\sum_{j=0}^k \eta_j \rho_{j+1}$ converges.*

Proof. Denote by $\mathcal{F}_k$ the filtration generated by $\{w_1,\dots,w_k\}$. It holds that $\mathbb{E}[\widetilde{\nabla} f(w_k)|\mathcal{F}_k] = \nabla f(w_k)$. Furthermore, almost surely, $\mathbb{E}[\|\rho_{k+1}\|^2|\mathcal{F}_k] \leq 2\mathbb{E}[\|\widetilde{\nabla} f(w_k)\|^2|\mathcal{F}_k] + 2\|\nabla f(w_k)\|^2 < 4L_f$, where $L_f = \sup_{1\leq j\leq|\mathcal{D}|} L_{f_j}$. Thus, for $i \in \{1,\dots,d\}$, $\sum_{j=0}^k \eta_j \rho_{j+1}^i$ is a martingale with an almost surely bounded square variation (since $\sum_{j=0}^{+\infty} \eta_j^2 < +\infty$). The proof is concluded by applying Klenke 2013, Theorem 11.14. □

In all the sequel, it is implicitly assumed that ϵ was chosen small enough to satisfy the assumption of Theorem 5. Denote by $k_{0,\epsilon}$ the smallest integer after which we do not perform the clipping step in Algorithm 2.

$$k_{0,\epsilon} := \inf\{k \geq 0 : \text{ for } m \geq k, s_{\epsilon,\alpha}^c(\widetilde{\nabla} f(w_m), w_m) = s_{\epsilon,\alpha}(\widetilde{\nabla} f(w_m), w_m)\}. \tag{A.9}$$

Since $\limsup \mathrm{d}(w_k, C_\epsilon) = 0$, it holds that $\liminf \psi_\epsilon(w_k^i) \geq 0$ and, therefore, $k_{0,\epsilon}$ is almost surely finite. Thus, for $k \geq k_{0,\epsilon}$, $v_k^i = [s_{\epsilon,\alpha}(\widetilde{\nabla} f(w_k), w_k)]^i$, which implies:

$$v_k^i = -\widetilde{\nabla}_i f(w_k) + \lambda_k^i \psi_\epsilon'(w_k^i), \tag{A.10}$$

with $\lambda_k^i = 0$ if $\psi_\epsilon(w_k^i) > 0$ and $\lambda_k^i = (v_k^i + \widetilde{\nabla}_i f(w_k))/\psi_\epsilon'(w_k^i)$ otherwise. Notice that since the sequences $(v_k), (w_k)$ are almost surely bounded, $\sup_{k\geq k_{0,\epsilon}} |\lambda_k^i|$ is almost surely finite.

Lemma 25. *Assume A5-A6. For $i \in \{1,\dots,d\}$, and for $k \geq k_{0,\epsilon}$, $\lambda_k^i \geq 0$.*

Proof. First, notice that if $\psi_\epsilon(w_k^i) > 0$, then $\lambda_k^i = 0$ by construction.

Consider now the case where $\psi_\epsilon(w_k^i) \leq 0$. If $-\widetilde{\nabla}_i f(w_k)\psi'_\epsilon(w_k^i) \geq -\alpha\psi_\epsilon(w_k^i)$, then $v_k^i = -\widetilde{\nabla}_i f(w_k)$ and, since for $k \geq k_{0,\epsilon}$, $\psi'_\epsilon(w_k^i) \neq 0$, this implies $\lambda_k^i = 0$. Otherwise, $v_k^i = -\alpha\psi_\epsilon(w_k^i)/\psi'_\epsilon(w_k^i)$ and $0 < -\alpha\psi_\epsilon(w_k^i) + \widetilde{\nabla}_i f(w_k)\psi'(w_k^i)$. Dividing the last inequality by $\{\psi'_\epsilon(w_k^i)\}^2$, we obtain $0 < (-\alpha\psi_\epsilon(w_k^i) + \widetilde{\nabla}_i f(w_k)\psi'_\epsilon(w_k^i))/\{\psi'_\epsilon(w_k^i)\}^2 = (v_k^i + \widetilde{\nabla}_i f(w_k))/\psi'_\epsilon(w_k^i) = \lambda_k^i$. □

The rest of this section is devoted to the proof of Lemma 23.

Denote $V = \max(V_{\epsilon,c}, \sup_{1\leq j\leq|\mathcal{D}|} L_{f_j})$ and notice that for any $k \geq 0$ and $i \in \{1,\dots,d\}$, $\|\widetilde{\nabla} f(w_k)\| \leq V$ and $|v_k^i| \leq V$. Therefore, $|w_{k+1}^i - w_k^i| \leq \eta_k V$. The lemma will be proved by the following claims.

Claim 1. For $i \in \{1,\dots,d\}$, and for $2 \leq j \leq M_i - 1$ if the set $[(c_j^i + c_{j-1}^i)/2, (c_j^i + c_{j+1}^i)/2)$ is visited by w_k^i infinitely often, then there is k_0 such that for all $k > k_0$, $w_k^i \in [(c_j^i + c_{j-1}^i)/2, (c_j^i + c_{j+1}^i)/2)$.

Indeed, fix such a j and denote $[c_-, c_+]$ the set $C_\epsilon^i \cap [(c_j^i + c_{j-1}^i)/2, (c_j^i + c_{j+1}^i)/2)$, where C_ϵ^i is the projection of C_ϵ onto the i-th coordinate. Define $k_0 = \sup\{k : \eta_k V \geq \max(c_- - (c_j^i + c_{j-1}^i)/2, (c_j^i + c_{j+1}^i)/2 - c_+)\}$. Consider $k \geq k_0$, if $(c_j^i + c_{j-1}^i)/2 \leq w_k^i \leq c_-$ (we are on the left side of the interval), then the iterate is pushed to the right and $w_k^i \leq w_{k+1}^i$. Furthermore, by definition of k_0, it holds that $w_{k+1}^i \leq c_- + \eta_k V \leq (c_j^i + c_{j+1}^i)/2$. This implies, that in this case w_{k+1}^i stays in $[(c_j^i + c_{j-1}^i)/2, (c_j^i + c_{j+1}^i)/2)$. Otherwise, if $c_+ \leq w_k^i < (c_j^i + c_{j+1}^i)/2$ (we are on the right side of the interval), then, we are pushed to the left, and, by a similar reasoning, $w_{k+1}^i \in [(c_j^i + c_{j-1}^i)/2, (c_j^i + c_{j+1}^i)/2)$. Finally, if $w_k^i \in [c_-, c_+]$, then by the way k_0 was defined we obtain that $w_{k+1}^i \in [(c_j^i + c_{j-1}^i)/2, (c_j^i + c_{j+1}^i)/2)$. Thus, we have shown that for $k \geq k_0$, if w_k^i is in $[(c_j^i + c_{j-1}^i)/2, (c_j^i + c_{j+1}^i)/2)$, then for all $k' \geq k$, the same will be true for $w_{k'}^i$, which completes the proof of the claim.

The proof of the following two claims is similar to the one of Claim 1.

Claim 2. For $i \in \{1,\dots,d\}$, if the set $(-\infty, (c_1^i + c_2^i)/2)$ is visited by w_k^i infinitely often, then there is k_0 such that for all $k > k_0$, $w_k^i \in (-\infty, (c_1^i + c_2^i)/2)$.

Claim 3. For $i \in \{1,\dots,d\}$, if the set $[(c_{M_i-1}^i + c_{M_i}^i)/2, +\infty)$ is visited infinitely often, then there is k_0 such that for all $k > k_0$, $w_{k^i} \in [(c_{M_i-1}^i + c_{M_i}^i)/2, +\infty)$.

In the following, without loss of generality, we will assume that we are in the context of the first claim and that there is k_0, such that for all $k \geq k_0$, $w_k^i \in [(c_j^i + c_{j-1}^i)/2, (c_j^i + c_{j+1}^i)/2)$ (the two other cases can be treated in the exact same manner).

Denote, as previously, $[c_-, c_+]$ the set $C_\epsilon^i \cap [(c_j^i + c_{j-1}^i)/2, (c_j^i + c_{j+1}^i)/2)$, where C_ϵ^i is the projection of C_ϵ onto the i-th coordinate.

Claim 4. There is k_0, such that if there are two index $m_+ \geq m_- > k_0$ such that $w_{m_-} < c_- < c_+ < w_{m_+}$, then there is m, satisfying $m_- \leq m \leq m_+$, such that $w_m^i \in [c_-, c_+]$.

Indeed, define $k_0 = \sup\{k : \eta_k V \geq c_+ - c_-\}$. Let m_-, m_+ be as in the claim and consider $m = \inf\{k \geq m_- : w_k^i \geq c_-\}$. It holds that $w_{m-1}^i < c_- \leq w_m^i \leq w_{m-1}^i + \eta_k V$. Since $m \geq k_0$, this implies that $w_m^i \leq c_- + \eta_k V \leq c_+$, which proves the claim.

Claim 5. There is k_0, such that if there are two index $m_- \geq m_+ > k_0$, such that $w_{m_-} < c_- < c_+ < w_{m_+}$, then there is m, satisfying $m_+ \leq m \leq m_-$, such that $w_m^i \in [c_-, c_+]$. The proof is the identical to the one of the previous claim.

From the fourth and fifth claims, there are only three possible behaviors of w_k^i. Either, w_k^i visits $[c_-, c_+]$ infinitely often (this will be treated by the sixth claim), or for k large enough, w_k^i stays at the left of $[c_-, c_+]$ (this will be treated by the seventh claim), or it stays at the right of $[c_-, c_+]$ (this will be treated by the eights claim).

Claim 6. If w_k^i visits $[c_-, c_+]$ infinitely often, then $\limsup w_k^i \leq c_+$ and $\liminf w_k^i \geq c_-$.

Notice that if $w_k^i > c_+$, then $w_{k+1}^i \leq w_k^i$, and if $w_k^i \leq c_+$ and $w_{k+1}^i \leq c_+ + \eta_k V$. Thus, if k is such that $w_k^i \in [c_-, c_+]$, then $\sup_{k_1 \geq k} w_{k_1}^i \leq c_+ + \eta_k V$. Letting k tend to infinity, proves first part of the claim. Similarly, if k is such that $w_k^i \in [c_-, c_+]$, then $\inf_{k_1 \geq k} w_{k_1}^i \geq c_- - \eta_k V$. Letting k tend to infinity proves the second part of the claim.

Claim 7. If for all k large enough, $w_k^i > c_+$, then $w_k^i \to c_+$.

Indeed, in this case, for k large enough, the sequence w_k^i is decreasing and thus has a limit. Denote this limit w_+ and assume that $w_+ \neq c_+$, then for k large enough, it holds that $w_{k+1}^i = w_k^i + \eta_k v_k^i \leq w_k^i - \eta_k V_+$, where $V_+ = \inf\{\min(V_{\epsilon,c}, \alpha|\psi_\epsilon(w)|/|\psi_\epsilon'(w)|) : w \in [w_+, (c_j + c_{j+1})/2)\} > 0$. Thus, for any m, it holds that $w_{k+m+1}^i \leq w_k^i - V_+ \sum_{i=0}^m \eta_{k+i}$. Since $\sum_{j=0}^{+\infty} \eta_j = +\infty$, this shows that this case is impossible. Hence, $w_k^i \to c_+$.

Claim 8. If for all k large enough, $w_k^i < c_-$, then $w_k^i \to c_-$.

Similarly, to the previous claim, for k large enough the sequence w_k^i is increasing and thus has a limit. If $w_- \neq c_-$, then for k large enough and $m \geq 0$, it holds that $w_{k+m+1}^i \geq w_k^i + V_- \sum_{i=0}^m \eta_{k+i}$, where $V_- = \inf\{\min(V_{\epsilon,c}, \alpha|\psi_\epsilon(w)|/|\psi_\epsilon'(w)|) : w \in ((c_{j-1} + c_j)/2, w_-]\} > 0$. Since $\sum_{j=0}^{+\infty} \eta_j = +\infty$, this implies that $w_- \neq c_-$ is impossible. Hence, $w_k^i \to c_-$.

These claims show that for every $i \in \{1, \dots, d\}$, $\liminf \psi_\epsilon(w_k^i) \geq 0$. Therefore, $\limsup \mathrm{d}(w_k, C_\epsilon) = 0$.

A.2 Numerical results

In this section, we give more details about our experiments, and present results on new tasks.

A.2.1 Toy convex example

We give more results about the toy example detailed in Section 2.4. We only compare `AskewSGD` and BinaryConnect Courbariaux, Bengio, and David 2015 in a logistic regression problem, but we test several settings to highlight the strengths of `AskewSGD` : all methods are trained for a longer time (50 epochs) using the SGD optimizer, the learning rate is set to 1, and gradients are calculated on random batches of 100 or 1000 samples. Note the rest of the experimental setting is identical: we generate $n = 6000$ feature vectors $\{x_k\}_{k=1}^n$ in dimension $d = 10$ drawn independently from the uniform distribution in $[-1, 1]$. We randomly choose an optimal vector w_* on the vertices of the hypercube and generate the labels as follows: $y_k \sim \text{Bernoulli}(\{1 + \mathrm{e}^{-x_k^\top w_*}\}^{-1})$. For completeness, we study how a full precision SGD converges to the optimal point w_* of this convex problem. The same conclusions can be drawn: `AskewSGD` is very close to the full precision method while STE method suffers from oscillations. Note however that decreasing the batch size seems to have a beneficial effect for STE, the larger variance helps to reduce the gap between STE and the other methods (see down panel in Figure A.1).

A.2.2 "2 moons" example

We consider the binary classification problem "2 moons dataset" presented in Section 2.4 and inspired by Meng, Bachmann, and Khan 2020. The training dataset consists of $n = 2000$ samples and 200 test samples and is displayed in Figure A.2a. A BNN with 9 weights is trained with one-hot coding and logistic loss. This BNN uses ReLu activations and its architecture is shown in Figure A.2b. Four gradient-based approaches - a full precision NN, BinaryConnect, AdaSTE, and `AskewSGD` - are compared to exhaustive search. In the latter, all 2^9 binary configurations on

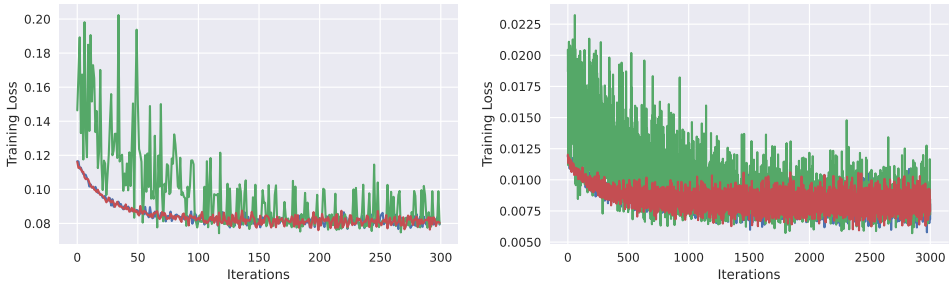
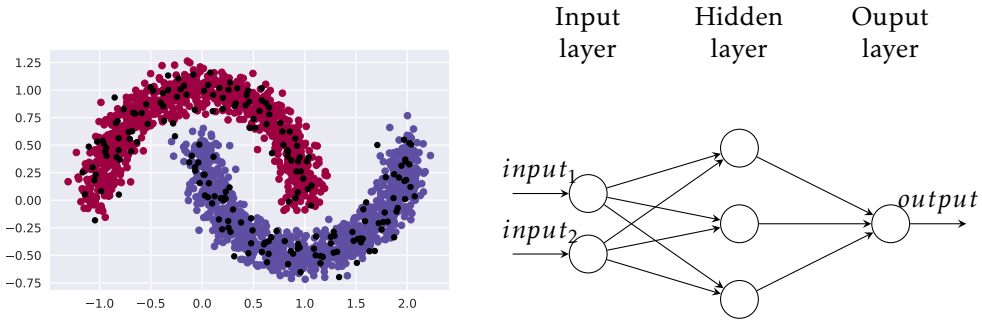


Figure A.1: Training losses for the logistic regression problem with batches of size 1000 (up panel) and 100 (down panel). BinaryConnect - green - AskewSGD - blue - full Precision methods - red -. The x-axis represents the iteration index.

the training and test sets are compared. Figure A.3 shows that different configurations lead to near-optimal performance. It is worth noting that permutation invariance implies that many solutions are equivalent in this simple example.



(a) Two moons dataset

(b) Basic BNN structure in dimension $d = 9$

Figure A.2: 2D Dataset and the associated BNN.

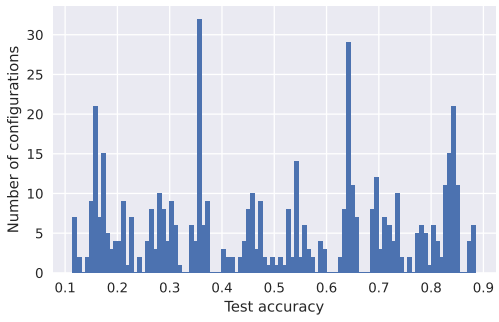


Figure A.3: Histogram of test accuracies for the exhaustive search in dimension $d = 512$

Table A.1: Logistic Loss on test samples (average over 50 random experiments) for the binary classification problem in dimension $d = 512$ after 100 steps.

Method	Loss ($\times 10^{-3}$)
Full-precision NN [W32/A32]	2.045 ± 0.005
BinaryConnect [W1/A32]	2.32 ± 0.11
AdaSTE [W1/A32]	2.24 ± 0.10
AskewSGD [W1/A32]	2.11 ± 0.01
Exhaustive search [W1/A32]	**2.1**

A.2.3 Deep learning experiments

The performances reported in Section 2.4 were obtained with the best combination of hyperparameters that we tested. Other combinations are listed in Table A.2 for the CIFAR-10 dataset. The performances reported in Table A.2 are still dependent on hyperparameter grid search and

Table A.2: Best Test accuracy after 100 training epochs on CIFAR-10.

α / lr	0.5/0.06	0.2/0.01	0.2/0.03	0.2/0.05	0.4/0.01
AskewSGD	88.51	85.60	88.42	88.32	84.50

could be further improved if more resources are available.

A.2.4 ImageNet with binary weights

In this section, we compare the performance of AskewSGD [W1/A32] on a large dataset and compare it to BinaryConnect Courbariaux, Bengio, and David 2015; Hubara et al. 2016, Mirror Descent Ajanthan, K. Gupta, et al. 2021, AdaSTE H. Le et al. 2021, a standard full-precision NN, and a hypersphere-projected full-precision NN. To ensure a fair comparison, we compare the different methods using the same NN architecture. Moreover, we do not add bias in any layer, but introduce batch normalisation (without learning parameters) after each layer. The last connected layer is kept in full precision - a standard practice in BNN -. Contrary to (Z. Liu, Shen, et al. 2020; Chmiel et al. 2021), we have kept the first convolutional layer binary. We do not use layerwise scalar contrary to Rastegari et al. 2016.

We use a training setting similar to Section 2.4. We have adapted the code of Ajanthan, K. Gupta, et al. 2021; H. Le et al. 2021 to Resnet-18 for ImageNet. The hyperparameters for AdaSTE and MD are those prescribed for TinyImageNet. We use the same default data normalizations as the methods we compare to: we resize the input images to 256×256 and then randomly crop them to 224×224 while centering them to the appropriate sizes during training. Standard multiclass cross entropy loss is used. All models are fine-tuned for 100 epochs using the Adam (Kingma and J. Ba 2014) optimizer with dynamics of 0.9 and 0.999 and a batch of size 512. The full precision NN is trained with an initial learning rate of 0.08. The projected full precision NN uses a projected gradient algorithm. The same hyperparameters are used as in the "simple" algorithm, except that a deterministic projection onto the hypersphere is performed at each iteration. The AskewSGD method is described in Algorithm 2, and we have set α to 0.5. The precision threshold ϵ is decreased from epoch to epoch: it is set to 1 at the beginning and then exponentially annealed to $.88^t$ in the last 50 epochs, where t is the epoch.

Just as with Section 2.4, we apply the function sign(·) to our NN before evaluating it on the test set. Each method was randomly initialized and independently executed once (due to ImageNet's longer training time). The learning rate at epochs [20, 40] is divided by 2 for all methods. This task is more difficult than TinyImageNet's, but we get the same result: AskewSGD

Table A.3: Best Test accuracy after 100 training epochs.

Method	ImageNet (ResNet-18)	
	Top-1	Top-5
Full-precision	66.39	95.32
BinaryConnect	45.85	71.05
MD	46.38	71.18
AdaSTE	35.37	62.22
Projected gradient	2.58	7.93
AskewSGD	**46.95**	**72.11**

outperforms all current baselines. Moreover, AskewSGD yields good results even when trained from scratch, compared to methods Bai, Y.-X. Wang, and Liberty 2018; Z. Liu, Shen, et al. 2020 that require fine-tuning using a pre-trained network.

A.2.5 BNN with binary activations

BNN with binary weights and binary activations offer significant time savings in inference. We applied our training procedure AskewSGD [W1/A1] to a VGG-small with sign(·) activations instead of ReLu activations to enable inference with only XNOR and bit-counting operations. The quantization of activations is here too extreme to apply the same procedure as in Section 2.4. The biased quantizer SAWB from Choi et al. 2018 does not work anymore (empirically). We assume the loss of neural gradient information is too important when activations are quantized on 2 levels.

During the training phase, a batch normalisation layer is inserted before each sign activation to scale the variance. In the backward pass, the derivative of sign(·) is approximated by the derivative of the function tanh(·). During inference, we can get rid of the batch normalization (only the empirical mean is conserved and added to the bias term) and compute only binary operations.

We compare test accuracy with the CIFAR-10 dataset, which consists of 50000 training images and 10000 test images (in 10 classes). BNNs are fine-tuned for 100 epochs using the Adam optimizer with a dynamic range of 0.9 and 0.999 and a batch size of 100 with a learning rate of 0.03. The best test classification accuracies of binary networks obtained with AskewSGD are listed in Table A.4 for different values of $\alpha \in [0.2, 0.5, 0.7]$. The preliminary results reported in Table A.4

Table A.4: Best Test accuracy after 100 training epochs.

α	0.2	0.5	0.7
AskewSGD [W1/A1]	81.12	**84.34**	82.92

show that AskewSGD is state-of-the art for training BNNs with binary weights and activations. Activation with the function sign(·) leads to a loss in expressive power and consequently a loss in performance. Several works introduce additional tricks such as real scaling factors Rastegari

et al. 2016 to bridge the gap between binary signals and their real counterparts. These tricks can be easily implemented in our approach AskewSGD.

Appendix B

Supplementary material of Chapter 5

B.1 Complementary comments

B.1.1 On the Importance of unbiasedness in `Dostovoq`

Randomizing codebooks is a crucial aspect of our method. While the independence of the different compression is not always highlighted in the papers, it is often crucial for the proofs:

- If the same codebooks were used by several users at the same iteration, the mutual independence of the compressions would be lost. Typically, the consequence would be to lose the good dependency on the number of users, which is one of our key focus;
- if a codebook used in the past by one user was used again at any further iteration, then the gradient would not be unbiased conditionally to the past.

The way we handle seeds in DoStoVoQ ensures that each codebook is generated in a completely independent way. Note that, in practice, this does not impair the speed of the algorithm, as sampling new codewords is extremely fast.

B.1.2 Complementary experiments results and discussions

Other experimental settings.

In this Subsection, we provide several complementary convergence results on end-to-end training on CIFAR10:

1. Under statistical heterogeneity of the workers
2. Adding Error-Feedback (EF)
3. Comparing to more compression schemes
4. Comparing on more Neural Network architectures.

These experiments corroborate the versatility of our method. However, we underline that an important message of our work is that our thorough comparison on a simpler metric (distortion)

Table B.1: Test accuracy on "heterogeneous" CIFAR10 after 100 epochs.

CIFAR, VGG16	Compression	Accuracy
Dostovoq-DIANA, $D = 8, M = 2^{12}$	20x	90.75%
SGD (baseline)	no compression	90.73%

Table B.2: Test accuracy on CIFAR10 after 100 epochs, with Error Feedback.

CIFAR, VGG16	Compression	Accuracy
Dostovoq-SGD-EF, $D = 8, M = 2^{12}$	20x	92.7%
Dostovoq-SGD, $D = 8, M = 2^{12}$	20x	92.1%

brings better insights than "aggregated" metrics (performance of an advanced algorithm, combining memory and EF, after 100 epochs on a large dataset, with a specific batch size, learning rate decay, momentum parameter, etc.), that may not reflect the actual quality of the compression technique, or highly depend on the computational power used for tuning.

Performance under heterogeneity. Our algorithm naturally extends to the statistical heterogeneous setting, in which the data hold by each worker $i \in [n]$ follows a different distribution $\mathbb{P}_i$. However, compressed stochastic gradient algorithms (e.g., QSQD) strongly suffer from heterogeneity (independently of which compressor is used). To limit this impact we use Dostovoq-DIANA (which code is given in Algorithm 12). The crucial idea is to rely on the DIANA algorithm (Mishchenko, Gorbunov, et al. 2019), that introduces a *memory*, and recovers (nearly) the same convergence in the heterogeneous setting as the one in the homogeneous setting.

We run Dostovoq-DIANA on CIFAR-10. To obtain heterogeneity, for each worker ($n = 8$), 50% of the data was selected from a unique class (different for each worker), and 50% uniformly among all classes. Results in Table B.1 highlight the robustness of our approach under heterogeneity.

Compatibility with EF. Adding EF to our pipeline is straightforward (that is, the modification is independent of the compression technique). We chose not to add this mechanism to preserve our focus on the compression technique itself, and because the importance of EF is mostly supported on biased contractive operators (Karimireddy, Rebjock, et al. 2019; Gorbunov, Kovalev, et al. 2020). In practice, EF is known to often improve convergence. We performed a supplementary experiment on CIFAR10, with the same tuning, which allowed to improve the performance by 0.6% to 92.7% (see Table B.2).

End to end empirical comparison with (biased) compression methods (Top-k, Cross-Polytope) We compared to Top-k in Tables 5.1, 5.3 and B.9 to B.11 in terms of distortion. Keeping the same setting as throughout the Section 5.5 (i.e., no tuning of parameters, we use the best parameters of SGD, no Error Feedback), we performed an experiment on Top-k (k chosen to achieve 8× compression), achieving 91.2% of accuracy, below our 92.1% for Dostovoq (see Table B.3). We also report the result of Cross-Polytope method from Gandikota et al. 2021, to which we had also compared distortion in Tables 5.1, 5.3 and B.9 to B.11.

Different architecture. We ran the code with a different architecture (ResNet18) on CIFAR-10 (with a single user simulated) instead of the VGG16 model, and present the results in Table B.4. SGD method reaches 94.5% while Dostovoq method reaches 94.4% with a ×20 compression. Note that the open-source code available as Supplementary material allows to replicate or extend these results to other architectures, without additional coding.

Table B.3: Test accuracy on CIFAR10 after 100 epochs

CIFAR10, VGG16	test accuracy	compression factor
No compression	91.9%	1x
Top-k	91.2%	8x
Cross-Polytope	90.9%	16x
`Dostovoq`$\left(M = 2^{12},\ \mathrm{d} = 8\right)$	92.1%	20x

Table B.4: Test accuracy with a Resnet18 on CIFAR10 after 100 epochs

CIFAR-10, Resnet 18	Compression	Accuracy
[Reported in Atomo, Fig 3a] Atomo,	6×	80%
DoStoVoQ-SGD, $D = 8, M = 2^{12}$	20×	94.35%

Compartison to Atomo Atomo (Hongyi Wang, Sievert, et al. 2018) is mostly related to some of the methods in Gandikota et al. 2021, where the goal is to compute an unbiased decomposition of a vector on a set of points. As underlined in Chapter 5, this corresponds to a Delaunay decomposition. This requires to solve a meta-optimization problem at each step, which can result in substantial computational overhead. Finally, the best performance reported on CIFAR-10 and a ResNet-18 is 80% on test accuracy Hongyi Wang, Sievert, et al. 2018, Fig. 3a. Comparison is summarized in Tables B.4 and B.5.

Note that the compression factor is not directly reported in Hongyi Wang, Sievert, et al. 2018. To the best of our understanding, optimal results were obtained with an SVD with $s = 3$, that would correspond to a compression factor of ~ 6.

Computational complexity.

Empirical estimation of Memory and computational cost. The primary goal of compression is to reduce the volume of the exchanged messages. Indeed, limiting the number of bits exchanged is beneficial for the bandwidth usage, energy consumption, etc. This aspect is described as (one of) the main motivation of compression in (Kairouz, H. B. McMahan, et al. 2021, e.g. at pages 13, 32). This can result in an acceleration of the overall training process (Alistarh et al. 2017). However, as our experiments are performed in a simulated environment, we do not actually benefit from the acceleration resulting from the compression. Yet, in order to ensure that the computation of the compression does not result in an important overhead (as mentioned in H. Xu et al. 2020; S. Agarwal et al. 2021, this can sometimes make a compression scheme impracticable), we have compared the end to end wall clock time in Tables B.6 and B.7, showing that our method is on par with `QSGD` in terms of computational overhead.

Discussion on the memory and time complexity.

- Memory: The only extra-memory requirement is the storage of the Codebook, of size $M \times D$,

Table B.5: Test accuracy with VGG-like networks on CIFAR10 after 100 epochs

CIFAR-10	Compression	Accuracy
[Reported in Atomo, Fig 3c] Atomo, VGG11	6×	78%
`Dostovoq`-SGD, $D = 8, M = 2^{12}$, VGG16	20×	92.1%

Table B.6: End-to-end time complexity for a VGG16 on CIFAR10

CIFAR10, VGG16	Time per epoch	Compression factor
No compression	30 s	1x
QSGD	51 s	8x
DoStoVoQ $\left(M = 2^{10}, D = 8\right)$	52 s	25x

Table B.7: End-to-end time complexity for a Resnet18 on Imagenet

ImageNet, ResNet18	Time per epoch	Compression factor
No compression	1925 s	1x
QSGD	2702 s	8x
DoStoVoQ $\left(M = 2^{10}, D = 8\right)$	2523 s	25x

with (typically) $D = 16$ and $M = 4096$. This is often marginal w.r.t. the size of the model of size d.

- Time-Complexity: At each step, on each node, the algorithm has to perform the following two steps:(i) Loading a batch of data, computing backpropagation on this batch at the current model, (ii) Performing compression.

The balance between both obviously strongly depend on the time-complexity for backpropagation, that is highly dependent on the network architecture. For compression, their are two main steps:

- Sampling the codebook. Experimentally, this is very fast.
- Finding the nearest neighbors in the codebook for each bucket.

Remark that this last step is extremely and easily parallelizable (it can be solved by performing a tensor product of two matrices).

Regarding decompression, the central server has to generate the codebook and access its relevant elements. Both these steps are very fast.

Compatibility with all-reduce. Our method is not directly compatible with all-reduce, as it is the case for QSGD, SignSGD, Atomo, Top-K (S. Agarwal et al. 2021).

Choice of hyperparameters L, M, P

There is a trade-off in between performances and compression. Hence, there is no "optimal set of parameters". But in practice, a simple choice is $D = 8$, $M \in \{1024, 4096\}$, and $P = 3$.

For example, for $D \in \{2, 8, 16\}$, we have reported the accuracies on CIFAR10 (we increased the number of points until we consistently achieved more than 92%) in Table B.8.

Note on model compression

Another line of work focuses on *model* compression, with e.g., *Pufferfish* (Hongyi Wang, S. Agarwal, and Papailiopoulos 2021), or Caldas et al. 2018. Here, our focus is on *gradient* compression. Compressing the model typically results in more randomness, and would not necessarily be very suitable with high compression factors.

Table B.8: Comparison of Test accuracy performances for several values of D, M.

CIFAR-10, VGG16	$D = \mathbf{2}$	$D = \mathbf{8}$	$D = \mathbf{16}$
$M = 32$	92.2%	91.7%	90.9%
$M = 128$	92.4%	91.8%	91.8%
$M = 256$	–	91.8%	91.9%
$M = 1024$	–	92.1%	91.9%
$M = 4096$	–	92.1%	92.4%
$M = 8192$	–	–	92.0%

B.2 Proofs

B.2.1 Classical compressors mentioned in the main text

For completeness, we here recall the formal definitions of the scalar compression operators mentioned in the main text. For $i \in [d]$, denote by e_i the i-th canonical vector. Let $H \in [d]$.

Definition 26 (Sign (Bernstein et al. 2018)). *For any $x \in \mathbb{R}^d$, $Sign(x) := \sum_{i\in[d]} \operatorname{sign}(x_i)\, e_i$.*

Definition 27 (Top-H). *For any $x \in \mathbb{R}^d$, $Top\text{-}H(x) := \sum_{i\in T_H} x_i e_i$, where T_H is the set composed of the indices of the H largest (in absolute value) coordinates of x.*

Definition 28 (Rand-H). *For any $x \in \mathbb{R}^d$, $Rand\text{-}H(x) := \frac{d}{H}\sum_{i\in R_H} x_i e_i$, where R_H is the set composed of H random indices picket uniformly without replacement.*

Definition 29 (s-quantization operator). *Let $s \geq 1$ and $p \geq 1$. Given $x \in \mathbb{R}^d$, the s-quantization operator $\mathcal{C}_s$ is defined by:*

$$\mathcal{C}_s(x) := \|x\|_p \times \sum_{i=1}^{d} \operatorname{sign}(x_i)\left\{s^{-1}\lfloor s|x_j|/\|x\|_p \rfloor + \mathbb{1}_{\{U_i \leq s|x_j|/\|x\|_p - \lfloor s|x_j|/\|x\|_p \rfloor\}}\right\} e_i\,. \tag{B.1}$$

where $\{U_i\}_{i=1}^d$ are d-independent uniform random variables on $[0,1]$.

The s-quantization scheme verifies **A**7 with $\omega_s = \min(d/s^2, \sqrt{D}/s)$. Proof can be found in Alistarh et al. 2017, see Appendix A.1.

B.2.2 Notations

For $u, v \in \mathbb{R}^d$, $\langle u, v\rangle = u^\top v$ denotes the standard scalar product. For $p \geq 1$ and $x \in \mathbb{R}^d$, $\|x\|_p = \left\{\sum_{i=1}^d |x_i|^p\right\}^{1/p}$. When $p = 2$, we sometimes drop the subscript, i.e. we write $\|x\|$ as a shorthand notation of $\|x\|_2$.

A function $\varphi : \mathbb{R}^d \to \mathbb{R}$ is said to be a *radial function* if and only if φ is invariant under unitary transforms, i.e. for all $x \in \mathbb{R}^D$ and $U \in \mathrm{U}(D)$, $\varphi(Ux) = \varphi(x)$.

We denote for $t > 0$ by $\Gamma(t) = \int_0^{+\infty} u^{t-1}\mathrm{e}^{-u}\mathrm{d}u$ the Gamma function. $\mathrm{Leb}\, d$ the Lebesgue measure on $\mathbb{R}^d$. $\mathrm{B}(x; r)$ is the (Euclidean) ball centered at $x \in \mathbb{R}^d$ with radius $r > 0$. We denote by $S_{D-1} = \{x \in \mathbb{R}^D, \|x\| = 1\}$ the unit-sphere and σ_{D-1} the uniform distribution on S_{D-1}.

B.2.3 Proof of Lemma 6

Note that, for any $U \in \mathrm{U}(d)$ and $x \in \mathbb{R}^d$,

$$\mathrm{VQ}(Ux, \mathscr{C}_M) = \operatorname*{arg\,min}_{c \in \mathscr{C}_M} \|Ux - c\| = \operatorname*{arg\,min}_{c \in \mathscr{C}_M} \|x - U^\top c\| = U\,\mathrm{VQ}(x, U^\top \mathscr{C}_M), \tag{B.2}$$

where $U^\top \mathscr{C}_M = \{U^\top C_1, \dots, U^\top C_n\}$. Using (B.2) and **A**9, we get

$$\mathbb{E}_{\mathscr{C}_M \sim p}[g(\mathrm{VQ}(Ux, \mathscr{C}_M))] = \mathbb{E}_{\mathscr{C}_M \sim p}[g(U\,\mathrm{VQ}(x, U^\top \mathscr{C}_M))] = \mathbb{E}_{\mathscr{C}_M \sim p}[g(U\,\mathrm{VQ}(x, \mathscr{C}_M))]. \tag{B.3}$$

B.2.4 Proof of Theorem 7

We preface the proof of the Theorem by stating and proving two elementary lemmas.

Lemma 30. *Let $f : \mathbb{R}^d \to \mathbb{R}^d$ be a function such that $f(Ux) = Uf(x)$ for any $x \in \mathrm{S}_{d-1}$ and $U \in \mathrm{U}(d)$. Then, there exists $r \in \mathbb{R}$ such that $f(x) = rx$ for all $x \in \mathbb{R}^d$.*

Proof. For all $x \in \mathrm{S}_{d-1}$, define $g(x) = f(x) - \langle f(x), x\rangle x$. It is easily checked that for all $x \in \mathbb{R}^d$ and $U \in \mathrm{U}(d)$, $g(Ux) = Ug(x)$. Let U_x be the reflection symmetry with axis $\mathbb{R}x$: $U_x x = x$ and for any vector $y \in \mathbb{R}^d$ orthogonal to x, $U_x y = -y$. Since $g(x) = g(U_x x) = U_x g(x) = -g(x)$, we get that $g(x) = 0$ for all $x \in \mathrm{S}_{d-1}$. Finally, denote by $U_{x \to e_1}$ (where e_1 is the first canonical vector) any unitary transform satisfying $U_{x \to e_1} x = e_1$. We get

$$\langle f(x), x\rangle = \langle U_{x \to e_1}^\top f(U_{x \to e_1} x), x\rangle = \langle f(U_{x \to e_1} x), U_{x \to e_1} x\rangle = \langle f(e_1), e_1\rangle = r\,,$$

which concludes the proof. □

Lemma 31. *Let $f : \mathbb{R}^d \to \mathbb{R}^d$ be a function such that $f(Ux) = Uf(x)$ for any $x \in \mathbb{R}^d$ and $U \in \mathrm{U}(d)$. Then, there exists a function $r : \mathbb{R} \to \mathbb{R}$ such that $f(x) = r(\|x\|)x$. Moreover, $r(x) = \|f(\|x\|e_1)\|/\|x\|$.*

Proof. Let $\lambda > 0$, and define for $x \in \mathrm{S}_{d-1}$, $f_\lambda(x) = f(\lambda x)$. Lemma 30 shows that there exists $\rho(\lambda) \in \mathbb{R}$ such that, for all $x \in \mathrm{S}_{d-1}$, $f_\lambda(x) = f(\lambda x) = \rho(\lambda)x$. Hence for $x \in \mathbb{R}^d$, $f(x) = f_{\|x\|}(x/\|x\|) = \rho(\|x\|)x/\|x\|$. Hence $|\rho(\|x\|)| = \|f(x)\| = \|f(\|x\|U_{x/\|x\| \to e_1} x/\|x\|)\| = \|f(\|x\|e_1)\|$. The proof follows. □

Proof of Theorem 7. The existence of $r_M^p(x)$ follows from Lemmas 6 and 31. It remains to prove that $r_M^p(x) \geq 0$. Let $\mathscr{C}_M = \{Y_1, \dots, Y_M\}$ where $Y_1, \dots, Y_M$ are i.i.d. random vectors with distribution p. One has

$$\begin{aligned}\langle x, \mathrm{VQ}(x, \mathscr{C}_M)\rangle = \langle \mathrm{Proj}_{\mathscr{C}_M}(x), x\rangle &= \sum_{\ell=1}^M \langle Y_\ell \mathbf{1}_{\{\|Y_\ell - x\|^2 < \min_{k \neq \ell} \|Y_k - x\|^2\}}, x\rangle \quad \text{a.s.} \\ &= \sum_{\ell=1}^M \langle Y_\ell, x\rangle \mathbf{1}_{\{\|Y_\ell - x\|^2 < \min_{k \neq \ell} \|Y_k - x\|^2\}} \quad \text{a.s.}\end{aligned}$$

Then, as the Y_ℓ are exchangeable since i.i.d.,

$$\begin{aligned}\mathbb{E}_{\mathscr{C}_M \sim p}\langle x, \mathrm{VQ}(x, \mathscr{C}_M)\rangle &= M\mathbb{E}\Big[\langle Y_1, x\rangle \mathbf{1}_{\{\|Y_1 - x\|^2 < \min_{k=2:M} \|Y_k - x\|^2\}}\Big] \\ &= M\mathbb{E}\Big[\langle Y_1, x\rangle\ \mathbf{1}_{\{\|Y_1 - \|x\|e_1\|^2 < \min_{k=2:M} \|Y_k - \|x\|e_1\|^2\}}\Big]\end{aligned}$$

where we used in the second line that the Y_ℓ are (i.i.d. and) invariant under the action of $U(d)$.

Now, setting $Y_1 = (Y_1^{1:d})$ we study for a fixed $\lambda = \|x\| \geq 0$ the quantity

$$(*) = \mathbb{E}\Big[Y_1^1 \mathbf{1}_{\{(Y_1^1-\lambda)^2+\sum_{j=2}^d (Y_1^j)^2 < \min_{k=2:M}\|Y_k-\lambda e_1\|^2\}}\Big]$$

First we analyze the quadratic inequality in Y_1^1

$$(Y_1^1)^2 - 2\,\lambda Y_1^1 + \lambda^2 + \|Y_1^{2:d}\|^2 - \min_{k=2:M}\|Y_k - \lambda e_1\|^2 < 0.$$

Its (random) reduced discriminant reads $\Delta' = \min_{k=2:M}\|Y_k - \lambda e_1\|^2 - \|Y_1^{2:d}\|^2$. Hence

$$(*) = \mathbb{E} Y_1^1 \mathbf{1}_{\{Y_1^1 \in [1-\sqrt{\Delta'}, 1+\sqrt{\Delta'}] \,\&\, \Delta' > 0\}}.$$

▷ *Gaussian case*. As Δ' and Y_1^1 are independent and $\int_{-\infty}^{x} \xi e^{-\frac{\xi^2}{2}} \frac{d\xi}{\sqrt{2\pi}} = -\frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}}$, we derive

$$(*) = \mathbb{E}\Big[\frac{e^{-\frac{(1-\sqrt{\Delta'})^2}{2}} - e^{-\frac{(1+\sqrt{\Delta'})^2}{2}}}{\sqrt{2\pi}} \mathbf{1}_{\{\Delta'>0\}}\Big] = \mathbb{E}\Big[\frac{1}{\sqrt{2\pi e}} \mathbf{1}_{\{\Delta'>0\}} e^{-\frac{\Delta'}{2}}\Big(e^{\sqrt{\Delta'}} - e^{-\sqrt{\Delta'}}\Big)\Big] \geq 0$$

▷ *General case*. The marginal Y_1^1 has an absolutely continuous distribution with a density p_1 which can be taken even since Y is itself symmetric with density p so that $\int_{-b}^{+b} y^1 p_1(dy^1) = 0$. In view of the form of $(*)$, it suffices to show that $\int_{a-b}^{a+b} y^1 p_1(dy^1) \geq 0$ for every fixed $a, b > 0$. In fact, $y^1 \mapsto y^1 p_1(y^1)$ is an odd function so that $\int_{a-b}^{0} y^1 p_1(dy^1) = -\int_0^{b-a} y^1 p_1(dy^1)$ which in turn implies

$$\int_{a-b}^{a+b} y^1 p_1(dy^1) = \int_{b-a}^{a+b} y^1 p_1(dy^1) \geq 0.$$

Finally, this proves that $\mathbb{E}_{\mathscr{C}_M \sim p}\langle x, \mathrm{VQ}(x, \mathscr{C}_M)\rangle \geq 0$ (and clearly > 0 if $x \neq 0$) which completes the proof. □

B.3 Scalar and vector Quantization

B.3.1 Unbiased random scalar quantization

A random scalar quantizer is a random map from the real line to a (scalar) codebook $\mathscr{O}_Q = \{o_1, \ldots, o_Q\} \subset \mathbb{R}$ where $Q \geq 2$. It is assumed that $-\infty < o_1 < \cdots < o_Q < \infty$. The resolution (or code rate) is $P = \log_2(Q)$ is the number of bits needed to uniquely specify a codeword. A scalar quantizer is said to be *uniform* if for all $i \in [Q-1]$, $o_{i+1} - o_i = \delta$, for some $\delta > 0$. Note that in such case $\delta = \{o_Q - o_1\}/(Q-1)$.

For $x \in \mathbb{R}$ and $u \in [0,1]$, consider a function $\mathrm{SQ}(x, \mathscr{O}_Q, u) \in \mathscr{O}_Q$. If $U \sim \mathrm{Unif}([0,1])$, then $\mathrm{SQ}(x, \mathscr{O}_Q, U)$ is a random scalar quantizer. A random scalar quantizer is said to be *unbiased* if for all $x \in [o_1, \ldots, o_Q]$, $\mathbb{E}_{U \sim \mathrm{Unif}([0,1])}[\mathrm{SQ}(x, \mathscr{O}_Q, U)] = x$.

A simple way to construct an unbiased scalar quantizer goes a follows. We first compute the index $j(x) \in [Q]$ such that $x \in \Big[o_{j(x)}, o_{j(x)+1}\Big)$. Note that $x = \lambda^*_{j(x)}(x) o_{j(x)} + (1 - \lambda^*_{j(x)}(x)) o_{j(x)+1}$ where

$$\lambda^*_{j(x)}(x) = (x - o_{j(x)})/(o_{j(x)+1} - o_{j(x)}) \in (0,1]\,.$$

For $u \in (0,1]$, we set

$$\mathrm{SQ}(x, \mathscr{O}_Q, u) = \mathbb{1}_{\{u \le \lambda^*_{j(x)}(x)\}} o_{j(x)} + \mathbb{1}_{\{u > \lambda^*_{j(x)}(x)\}} o_{j(x)+1}\,.$$

Since $\mathbb{E}_{U\sim\mathrm{Unif}([0,1])}(U \le \lambda_j(x)^*(x)) = \lambda^*_{j(x)}$ the unbiasedness follows. It is easily seen that the distortion of a scalar quantizer is directly related to the diameter of the quantizer.

Proposition 32. *For all $x \in \left[o_1, o_Q\right]$, it holds that*

$$\mathbb{E}_{U\sim\mathrm{Unif}([0,1])}[\{\mathrm{SQ}(x, \mathscr{O}_Q, U) - x\}^2] \le (1/4) \sup_{i\in[Q-1]} \{o_{i+1} - o_i\}^2.$$

If the scalar quantizer is uniform,

$$\mathbb{E}_{U\sim\mathrm{Unif}([0,1])}[\{\mathrm{SQ}(x, \mathscr{O}_Q, U) - x\}^2] \le (1/4)(Q-1)^{-2}\{o_Q - o_1\}^2.$$

Proof. For all $x \in \left[o_1, o_Q\right]$, we get

$$|\mathrm{SQ}(x, \mathscr{O}_Q, U) - x| \le (1/2)\{o_{j(x)+1} - o_{j(x)}\}$$

The proof follows. □

Unbiased random scalar quantization is a special case of dual vector quantization, introduced in the next section.

B.3.2 Dual Vector Quantization

We introduce a new notion of vector quantization, called *dual quantization* (or *Delaunay quantization*). The principle of dual quantization is to map an $\mathbb{R}^d$-valued vector x onto a codebook $\mathscr{C}_M$ using a random splitting operator Dual-VQ$(x, \mathscr{C}_M, U)$ such that, for all $x \in \mathrm{ConvHull}(\mathscr{C}_M)$,

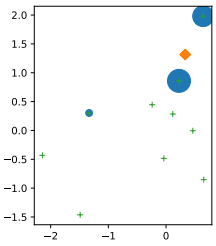
$$\mathbb{E}_{U\sim\mathrm{Unif}([0,1])}[\text{Dual-VQ}(x, \mathscr{C}_M, U)] = x. \tag{B.4}$$

We stress that in this case the unbiasedness is not due to the use of a random codebook but makes use of an external randomization. In practice, a dual quantizer procedure amounts to define a probability distribution of $\mathscr{C}_M$, with weights $(\lambda^*_1(x), \ldots, \lambda^*_M(x))$, $\lambda^*_i(x) \ge 0$, $\sum_{j=1}^M \lambda^*_j(x) = 1$. Set $\Lambda^*_0(x) = 0$ and for $i \in [M]$, $\Lambda^*_i(x) = \sum_{j=1}^i \lambda^*_j(x)$. Note that $\Lambda^*_M(x) = 1$. If $u \in \left(\Lambda^*_{j-1}(x), \Lambda^*_j(x)\right]$, $j \in [M]$, we set Dual-VQ$(x, \mathscr{C}_M, u) = c_j$. In such that, for all $x \in \mathrm{ConvHull}(\mathscr{C}_M)$, we get

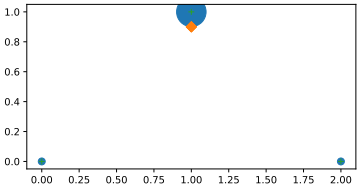
$$\mathbb{E}_{U\sim\mathrm{Unif}([0,1])}[\text{Dual-VQ}(x, \mathscr{C}_M, U)] = \sum_{i=1}^M \lambda^*_i(x) c_i = x.$$

The distortion of a dual quantizer is therefore given, for $x \in \mathbb{R}^D$, by

$$\mathbb{E}_{U\sim\mathrm{Unif}([0,1])}[\|\text{Dual-VQ}(x, \mathscr{C}_M, U) - x\|^2] = \sum_{i=1}^M \lambda^*_i(x)\|x - c_i\|^2. \tag{B.5}$$



(a) $x \sim \mathcal{N}(0, \mathrm{I}_2)$ and $M = 10$.



(b) $x \sim \mathcal{N}(0, \mathrm{I}_2)$ and $M = 3$.

Figure B.1: `Delaunay quantization` for a vector x (orange diamond), for a given set of codewords (green +),and corresponding weights (area of the blue spheres). Remark that all but three points have a 0 probability of being picked, making the quadratic error much smaller than for `HSQ-span`.

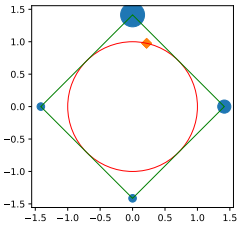


Figure B.2: `Cross-Polytope` (Gandikota et al. 2021) is a particular case of `Delaunay quantization`. The codewords are the vertices of $\mathrm{B}_1(0; \sqrt{d})$. A vector x (orange diamond) lying on the unit Ball $\mathrm{B}_2(0; 1)$ (red circle) is decomposed with weights (area of the blue spheres) of codewords on the Ball of radius $\sqrt{d}$ (green).

For $x \in \mathrm{ConvHull}(\mathscr{C}_M)$, the probability distribution $(\lambda_1^*(x), \ldots, \lambda_M^*(x))$ is obtained by solving the following convex optimization program:

$$(\lambda_1^*(x), \ldots, \lambda_M^*(x)) = \underset{(\lambda_1, \ldots, \lambda_M) \in \mathcal{S}(x, \mathscr{C}_M)}{\arg\min} \sum_{i=1}^{M} \lambda_i \|x - c_i\|^2, \tag{B.6}$$

where

$$\mathcal{S}(x, \mathscr{C}_M) = \left\{ (\lambda_1, \ldots, \lambda_M) \in \mathbb{R}_+^M, \sum_{i=1}^{M} \lambda_i = 1, \sum_{i=1}^{M} \lambda_i c_i = x \right\}. \tag{B.7}$$

The support of $(\lambda_1^*(x), \ldots, \lambda_M^*(x))$ is $M + 1$ at most. For a distribution q on $\mathbb{R}^D$, we define

$$\text{Dual-Dist}(q, \mathscr{C}_M) = \int q(x) \left\{ \sum_{i=1}^{M} \lambda_i^*(x) \|x - c_i\|^2 \right\} \mathrm{d}x. \tag{B.8}$$

For a given input distribution q, an optimal codebook $\mathscr{C}_M^*$ of cardinality M satisfies $\text{Dual-Dist}(q, \mathscr{C}_M^*) \leq \text{Dual-Dist}(q, \mathscr{C}_M)$ for all $\mathscr{C}_M$ satisfying $|\mathscr{C}_M| = M$.

Theorem 33 (Rates, see Pagès and Wilbertz 2018)**.** Asymptoptic rate. *Assume that the pdf q is compactly supported on $\mathbb{R}^d$-valued.*

$$\lim_{M \to \infty} M^{\frac{2}{d}} \inf_{|\mathscr{C}_M| = M} \text{Dual-Dist}(q, \mathscr{C}_M) =: Q_2^D(q) = Q_2^D(\mathrm{Unif}([0,1])^d) \|q\|_{\frac{D}{D+2}}. \tag{B.9}$$

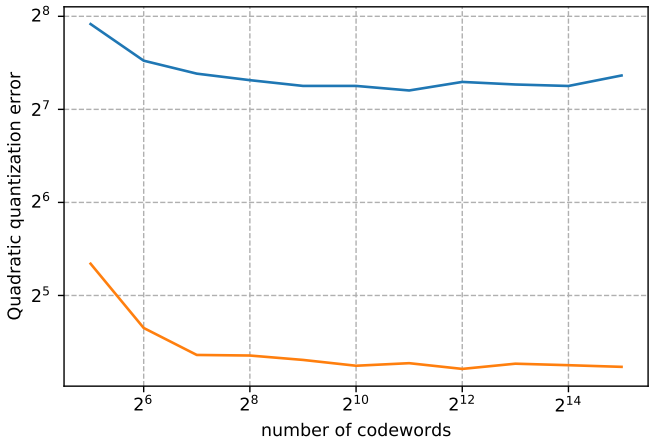


Figure B.3: HSQ-Span: Distortion as a function of M (log-scale): $n = 1$ (blue) $n = 8$ (orange).

B.3.3 HSQ methods - see Dai et al. 2019

In this Section, we provide a detailed review of the two methods proposed by Dai et al. 2019. In Appendix B.3.3, we first discuss HSQ-Span and explain why it cannot compete with approaches based on Voronoi quantization. In Appendix B.3.3, we discuss HSQ-greed.

HSQ-Span

The first method, HSQ-Span, is unbiased but suffers form a large variance. Indeed, it relies on decomposing the vector $x \in \mathbb{R}^d$ as a **linear** combination of the codewords in $\mathscr{C}_M$, assuming that $\text{Span}\{c_i, i \in [M]\} = \mathbb{R}^d$ (a codebook satisfying this property is said to be *full-rank*). Because typically $M \gg D$, there are infinitely many solutions to the linear problem $\sum_{i=1}^M \alpha_i c_i = x$, i.e. $\mathcal{A}(x, \mathscr{C}_M) = \{(\alpha_1, \dots, \alpha_M) \in \mathbb{R}^M, \sum_{i=1}^M \alpha_i c_i = x\}$ is infinite. Note that contrary to the Dual quantization approach, we do not assume that $\alpha_i \geq 0$ for $i \in [M]$ or $\sum_{i \in [M]} \alpha_i = 1$. However, for any $i \in [M]$, we pick the codeword c_i with probability $|\alpha_i|/\|\alpha\|_1$, and encode x as $\text{sign}(\alpha_i)\|\alpha\|_1 c_i$. In HSQ-Span, the **minimal norm** solution in $\mathcal{A}(x, \mathscr{C}_M)$ is chosen, i.e. solve

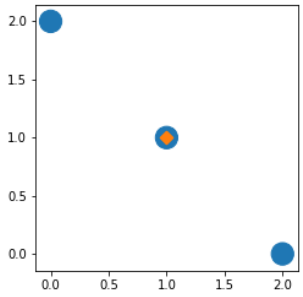
$$\alpha^*(x) := (\alpha_1^*(x), \dots, \alpha_M^*(x)) = \underset{(\alpha_1, \dots, \alpha_M) \in \mathcal{A}(x, \mathscr{C}_M)}{\arg\min} \sum_{i=1}^M \alpha_i^2, \tag{B.10}$$

The main advantage are that:

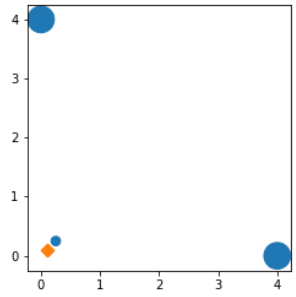
1. **Fast computation.** First, as $\alpha^*(x) = C^\dagger x$, where $C^\dagger$ is the Moore-Penrose inverse of the codewords matrix $C = [c_1, \dots, c_M]$, provided a fixed codebook $\mathscr{C}_M$, it is possible to compute only once $C^\dagger$ and to then obtain $\alpha^*(x)$ for any x by a simple matrix-vector product.

2. **Unbiased.** Second, this approach is unbiased. Its quadratic error thus linearly decays with the number of workers.

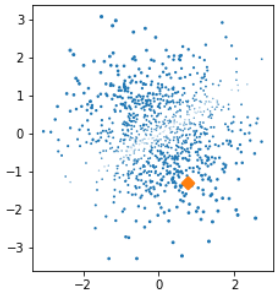
However (1) its variance is high and (2) does not decrease with M. Indeed, the minimal norm solution $\alpha^*(x)$ tends to put weight on **all** codewords. For example, we represent in Figure B.4



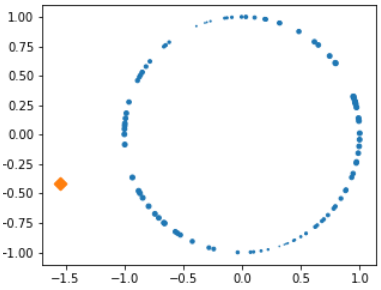
(a) $x_1 = (1,1)$,
$\mathscr{C}_M = [(2,0);(0,2);(1,1)]$.
$\alpha^*(x_1) \simeq (33\%, 33\%, 33\%)$.



(b) $x_2 = (0.1, 0.1)$,
$\mathscr{C}_M = [(4,0);(0,4);(1/4,1/4)]$
$\alpha^*(x_2) \simeq (47\%, 47\%, 6\%)$.



(c) $x \sim \mathcal{N}(0, \mathrm{I}_2)$ and $M = 1000$.



(d) $x \sim \mathcal{N}(0, \mathrm{I}_2)$ and $M = 100$, $p = \mathcal{U}(\mathcal{S}_1(\mathbb{R}^2))$.

Figure B.4: HSQ-Span: weights (size of the blue point) on each of the codewords of $\mathscr{C}_M$ when decomposing x (orange diamond) .

the weights on each vector for 3 situations in dimension $D = 2$. Intuitively, the probability of selecting c_i is not a decreasing function $\|x - c_i\|^2$ (see e.g., Figure B.4b), which results in the large variance; even if there exists i such that $c_i = x$, there is a non vanishing probability of selecting $c_j \neq c_i$ s (Figure B.4a). We illustrate the second point in Figure B.3 which gives the evolution of the distortion for $D = 16$ w.r.t. M for $n \in \{1, 8\}$ workers. The error does not decrease.

HSQ-greed

HSQ-greed is closed to Dostovoq: Dai et al. 2019 still consider a full-rank codebook $\mathscr{C}_M$, and simply encode x by $\mathrm{VQ}(x, \mathscr{C}_M)$. We list here the main differences to our approach:

1. the same codebook is used during all iterations and on all workers. This makes it impossible or cumbersome to apply the convergence result developed in the federated learning literature, which require that the **compression on each workers are independent** (at least between iterations).

2. No assumption is made on the codebook distribution (apart from the fact that it is full-rank). The importance of unitary invariance is not mentioned. In practice, authors use an

codebook generated by applying a k-means algorithm on a larger set of scaled Gaussian isotropic vectors. This pre-processing slightly improves the distribution of the codewords but is in practice of limited impact (see paragraph (e) in Section 5.2.2 in the main text.).

3. Codewords are chosen of norm 1. This means we also need to encode $\|x\|$ together with $\mathrm{VQ}(x, \mathscr{C}_M)$, which is typically done on using 6 bits per bucket.
4. The method is biased, so does not benefit from a large number of workers. No analysis of the quadratic error is provided.

Theoretical results. Dai et al. 2019 present a convergence result for `HSQ-greed`, namely in Lemma 3 and the subsequent Theorem 3. Note however that the proof of this result is **not** provided in the paper[1]. Second, the guarantee provided is almost vacuous. Indeed, authors rely on an alternative assumption[2] on the **alignment** of the compressed value $\mathrm{VQ}(x, \mathscr{C}_M)$ with x:

Definition 34 (Compression with preserved alignement)**.** *There exists $\alpha > 0$ such that for all $x \in \mathbb{R}^d$, we have $\langle \mathrm{Comp}(x), x\rangle^2 \geq (1-\alpha)\|x\|^2$.*

This assumption becomes stronger as $1-\alpha$ increases. However, Lemma 3 indicates that $1 - \alpha \geq \sigma_{\min}(C)/M$, with $\sigma_{\min}$ the minimal eigenvalue of the codebook matrix C. The bound guarantee thus worsens with M. A similar multiplicative factor $1/(1-\alpha) \propto M$ appears in their convergence rate Theorem 3. We note that without any assumption on the codebook distribution, it seems difficult to obtain any result, as the worst case codebook that satisfies the full-rank assumption could be arbitrarily bad (typically a unique codeword perturbed by a tiny amount of noise $\mathscr{C}_M = [c_1 + \eta\epsilon_i]_{i\in[M]}$, with η very small).

B.4 Algorithmic extensions

B.4.1 Spherical codebooks

In this section, we describe a spherical version of `Stovoq` and `Dostovoq`. Beyond the obvious change from the codeword distribution from Gaussian to uniform on the sphere, a key modification stems form the fact that each quantized vector has norm 1: the debiasing function does not depend on $\|x\|$, but only on the number of codewords M. Consequently, the bias correction does not need to be transmitted and can be directly performed on the central server.

On the other hand, the norm of each bucket has to be transmitted: the vector quantization is applied to the *shape*, i.e. the unitary vector $x/\|x\|$. We use a scalar quantizer for the norm, typically over 4-6 bits. For completeness, the codes of those two algorithms are given in Algorithms 10 and 11.

Algorithme 10 : Spherical-`Stovoq`

Input : $x \in \mathbb{R}^D$, D, M, P, seed s
Output : Codeword index $\mathbf{i}_c$, value $\mathbf{i}_r$

Sample $\mathscr{C}_M \sim \sigma_{D-1}$ with seed s ; `/* sample codebook with uniform distribution` σ_{D-1} `on the sphere */`
$c_l = \mathrm{VQ}(x/\|x\|, \mathscr{C}_M)$; `/* quantize (select a codeword in spherical codebook` $\mathscr{C}_M$`) */`
$\mathbf{i}_{c_l} \leftarrow$ index of c_l; `/* get index of codeword */`
$\mathbf{i}_r = \mathrm{SQ}(\|x\|)$; `/* quantize` r `on P bits */`

[1] The appendices of the paper were not available, neither on `https://arxiv.org/pdf/1911.04655.pdf`, nor on `https://paperswithcode.com/paper/hyper-sphere-quantization-communication`, on the date of the writing.

[2] See the work of Beznosikov, Horváth, et al. 2020 for a discussion between the possible assumptions.

Algorithme 11 : Spherical-Dostovoq over T iterations

Input : T nb of steps, $(\eta_t)_{t\geq 0}$ LR, w_0, D, M, P ;
Output : $(w_t)_{t\geq 0}$

for $t = 1,\dots,T$ **do**
  w_0 sends w_{t-1} and different seeds $s_{k,t}$ to each w_k;
  for $i = 1,\dots,n$ **do**
    Compute local gradient $g_{i,t}$ at w_{t-1};
    Split $g_{i,t}$ on $[b_{i,t}^1,\dots,b_{i,t}^L]$;
    for $\ell = 1,\dots,L$ *(in parallel)* **do**
      $(\mathbf{i}_c^{t,i,\ell},\mathbf{i}_r^{t,i,\ell}) = \text{Spherical}-\text{Stovoq}(b_{i,t}^{\ell},p,M,P,s_{i,t})$
    end
    Send $(\mathbf{i}_c^{t,i,\ell},\mathbf{i}_r^{t,i,\ell})_{\ell\in[L]}$ to w_0 ;
  end
  Reconstruct $(\hat{g}_{i,t})_{i\in n}$;
  Update: $w_t = w_t t - 1 - \eta_t \frac{1}{n}\sum_{i=1}^n \hat{g}_{i,t}$;
end

B.4.2 Extension to Dostovoq-DIANA and Dostovoq-VR-DIANA

In this subsection, we provide the adaptations of the Dostovoq algorithm to algorithms designed to handle heterogeneous workers, and for which the best complexities are achieved, namely DIANA Mishchenko, Gorbunov, et al. 2019 and VR-DIANA Horváth, Kovalev, et al. 2019. Those algorithms are based on the fundamental idea: relying on control variates $(h_{i,t})_{i\in[n],t\geq 0}$, updated at each iteration, that converge (in the convex case), for each worker i, to $\nabla f_i(w^*)$. Instead of compressing $g_{i,t}$, the algorithm compresses the difference between the actual gradient and the control variate $g_{i,t} - h_{i,t}$. The impact of those control variates (often referred to as *memory*) is to mitigate the discrepancy between workers' gradients that stems from the heterogeneity of the data-distribution between different workers. As explained in Appendix B.5 it is particularly relevant to reduce this discrepancy to maximize the impact of the multiple workers. The same idea can be incorporated within a variance reduced algorithm, we here focus on SVRG (R. Johnson and T. Zhang 2013) (extension to SAGA (Defazio, F. R. Bach, and Lacoste-Julien 2014) or other variants is straightforward). To incorporate variance reduction to the algorithm, we further assume that each f_i is a finite sum $\frac{1}{S}\sum_{s\in[S]} f_{i,s}$. Algorithms Dostovoq-DIANA and Dostovoq-DIANA-SVRG are provided in respectively Algorithms 12 and 13.

Algorithme 12 : `Dostovoq-DIANA` over T iterations . Lines specific to the `Diana` approach are highlighted in **blue**

Input : T nb of steps, $(\eta_t)_{t\geq 0}$ LR, w_0, p, M, P, l.r. α ;
Output : $(w_t)_{t\geq 0}$

Set $h_{i,0} = 0$ **for all** $i \in [n]$ **(or alternatively** $h_{0,i} = \nabla f_i(w_0)$**)**;
for $t = 1, \dots, T$ **do**
  $worker_0$ sends w_{t-1} and different seeds $s_{i,t}$ to each w_i;
  for $i = 1, \dots, n$ **do**
    Compute local gradient $g_{i,t}$ at w_{t-1};
    Set $\Delta_{i,t} = g_{i,t} - h_{i,t}$;
    Split $\Delta_{i,t} \times \sqrt{d}/\|\Delta_{i,t}\|$ on $[\delta_{i,t}^1, \dots, \delta_{i,t}^L]$;
    for $\ell = 1, \dots, L$ *(in parallel)* **do**
      $(\mathbf{i}_c^{t,i,\ell}, \mathbf{i}_r^{t,i,\ell}) = \texttt{Stovoq}(\delta_{i,t}^\ell, p, M, P, s_{i,t})$
    end
    Reconstruct $(\hat{\Delta}_{i,t})_{i\in n}$;
    Update memory: $h_{i,t+1} = h_{i,t} + \alpha \hat{\Delta}_{i,t}$;
    Send $(\|\Delta_{i,t}\|, (\mathbf{i}_c^{t,i,\ell}, \mathbf{i}_r^{t,i,\ell})_{\ell\in[L]})$ to w_0 ;
  end
  On the central node;
  Reconstruct $(\hat{\Delta}_{i,t})_{i\in n}$;
  Update: $w_t = w_{t-1} - \eta_t(\bar{h}_t + \frac{1}{n}\sum_{i=1}^n \hat{\Delta}_{i,t})$;
  Update averaged memory : $\bar{h}_{t+1} \left(:= \frac{1}{n}\sum_{i\in[n]} h_{i,t}\right) = \bar{h}_t + \frac{\alpha}{n}\sum_{i\in[n]} \hat{\Delta}_{i,t}$;
end

Algorithme 13 : `Dostovoq-DIANA-SVRG` over T iterations . Lines specific to the variance reduction approach are highlighted in **green**

Input : T nb of steps, $(\eta_t)_{t\geq 0}$ LR, w_0, p, M, P, l.r. α ;
Output : $(w_t)_{t\geq 0}$

Set $h_{i,0} = 0$ **for all** $i \in [n]$ **(or alternatively** $h_{0,i} = \nabla f_i(w_0)$**)**;
for $t = 1, \ldots, T$ **do**
  Sample $u_t \sim \mathcal{B}(S^{-1})$;
  $worker_0$ sends w_{t-1}, u_t and different seeds $s_{i,t}$ to each w_i;
  for $i = 1, \ldots, n$ **do**
    if $u_t = 1$ **then**
      Set $\rho_{i,s,t} = w_t$ **for all** $s \in [S]$;
    Sample $s_{i,t} \sim \mathrm{Unif}[S]$;
    Set $\mu_{t,i} = S^{-1} \sum_{s\in S} \nabla f_{i,s}(\rho_{i,s,t})$;
    Set $g_{i,t} = \nabla f_{i,s_{i,t}}(w_{t-1}) - \nabla f_{i,s_{i,t}}(\rho_{i,s_{i,t},t}) + \mu_{i,t}$;
    Set $\Delta_{i,t} = g_{i,t} - h_{i,t}$;
    Split $\Delta_{i,t} \times \sqrt{d}/\|\Delta_{i,t}\|$ on $[\delta_{i,t}^1, \ldots, \delta_{i,t}^L]$;
    for $\ell = 1, \ldots, L$ *(in parallel)* **do**
      $(\mathbf{i}_c^{t,i,\ell}, \mathbf{i}_r^{t,i,\ell}) = \mathtt{Stovoq}(\delta_{i,t}^\ell, p, M, P, s_{i,t})$
    end
    Reconstruct $(\hat{\Delta}_{i,t})_{i\in n}$;
    Update memory: $h_{i,t+1} = h_{i,t} + \alpha \hat{\Delta}_{i,t}$;
    Send $(\|\Delta_{i,t}\|, (\mathbf{i}_c^{t,i,\ell}, \mathbf{i}_r^{t,i,\ell})_{\ell\in[L]})$ to CS ;
  end
  On the central node (CS);
  Reconstruct $(\hat{\Delta}_{i,t})_{i\in n}$;
  Update: $w_t = w_{t-1} - \eta_t(\bar{h}_t + \frac{1}{n}\sum_{i=1}^K \hat{\Delta}_{i,t})$;
  Update averaged memory : $\bar{h}_{t+1} \left(:= \frac{1}{n}\sum_{i\in[n]} h_{i,t}\right) = \bar{h}_t + \frac{\alpha}{n}\sum_{i\in[n]} \hat{\Delta}_{i,t}$;
end

B.4.3 Extension to `Dostovoq-FedAvg`

Following the implementation of B. McMahan et al. 2017, we propose the natural extension to `Dostovoq`, namely the `Dostovoq-FedAvg` in Algorithm 14.

Algorithme 14 : Dostovoq-FedAvg over T iterations . Lines specific to the FedAvg approach are highlighted in **purple**

Input : T nb of steps, $(\eta_t)_{t\geq 0}$ LR, w_0, p, M, P, **client fraction C, E is the number of local epochs** ;
Output : $(w_t)_{t\geq 0}$

for $t = 1, \ldots, T$ **do**
  $m = \max(C \times n, 1)$;
  $[n_t] =$ **(random set of m clients)**;
  $u_k =$ **number of samples available in k**;
  $u =$ **total number of samples**;
  $worker_0$ sends w_{t-1} and different seeds $s_{i,t}$ to each w_i;
  for $i \in [n_t]$ **do**
    for *each local epoch j in E* **do**
      Compute local gradient $g_{i,t}^j$ at w_{t-1};
    end
    Average the gradients: $g_{i,t} = \frac{1}{E}\sum_j g_{i,t}^j$;
    Split $g_{i,t} \times \sqrt{d}/\|g_{i,t}\|$ on $[\delta_{i,t}^1, \ldots, \delta_{i,t}^L]$;
    for $\ell = 1, \ldots, L$ *(in parallel)* **do**
      $(\mathbf{i}_c^{t,i,\ell}, \mathbf{i}_r^{t,i,\ell}) =$ Stovoq$(\delta_{i,t}^\ell, p, M, P, s_{i,t})$
    end
    Reconstruct $(\hat{g}_{i,t})_{i\in n}$;
    Send $(\|g_{i,t}\|, (\mathbf{i}_c^{t,i,\ell}, \mathbf{i}_r^{t,i,\ell})_{\ell\in[L]})$ to CS ;
  end
  On the central node;
  Reconstruct $(\hat{g}_{i,t})_{i\in n}$;
  Update: $w_t = w_{t-1} - \eta_t(\frac{u_k}{u}\sum_{i=1}^n \hat{g}_{i,t})$;
end

B.5 Additional experiments

In this Section, we compare by Monte Carlo the distortions achieved by different compression schemes for 3 types of input x. For a given (random) compressor generically denoted $Q(\cdot)$ and $x \in \mathbb{R}^d$, we decompose $Q(x) = Q_{\|}(x) + Q_{\perp}(x)$, where $Q_{\|}(x) = \|x\|^{-2}xx^\top Q(x)$. With these notations, $Q_{\|}(x)$ and $Q_{\perp}(x)$ are the components of the quantization error which are colinear and orthogonal to x, respectively. By construction, $\|x - Q(x)\|^2 = \|x - Q_{\|}(x)\|^2 + \|Q_{\perp}(x)\|^2$. The distortion is computed for $n = 1$ and $n \in \{8, 20\}$ workers (depending on the experiments). We compare 10 compression schemes, corresponding to 7 algorithms (some with several variants): the signed algorithm (Sign) (see Definition 26), Top-H with $H = 2$ (see Definition 27), Rand-H with $H = 2$ (see Definition 28), HSQ-Span (see Appendix B.3.3) with $M = 2^{10}$ and a 6 bits scalar quantizer for the norm, HSQ-greed (see Appendix B.3.3) with $M = 2^{10}$ and a 6 bits scalar quantizer for the norm, Polytope (see Appendix B.3.2) with and without quantization of the norm, three variants of Stovoq with a Gaussian random codebook with $M = 2^{13}$ and $p = \mathcal{N}(0, (1 + 2/D)\,\mathrm{I}_D)$: GRVQ which is Stovoq without the radial debiasing step, Stovoq without quantization of r_M^p, and Stovoq with an unbiased scalar quantization of $(r_M^p)^{-1}$ over $P = 3$ bits (strictly speaking, only this last column corresponds to the algorithm Stovoq, the two previous versions have been

added to assess the influence of the debiasing by $\{r_M^p\}^{-1}$ and the quantization of $\{r_M^p\}^{-1}$).

We compare those algorithms over three tasks:

1. **Task 1:** Compression of a random vector from a standard Gaussian input distribution in dimension $D = 16$. We compare $n = 1$ to $n = 20$. Results are given in Appendix B.5.1.

2. **Task 2:** Compression of "real" gradients, extracted from a training performed with a VGG16 on CIFAR10 with SGD, extracted at epoch 10, on which a pre-processing similar to `Dostovoq` is applied. The minibatch gradients on a given layer are divided into buckets of dimension $D = 16$. A normalisation is applied to sets of $L = 32$ buckets (the normalisation for the blocks of $D \times L = 512$ coefficients are scalar quantized with a high-resolution scalar quantizer and sent to the parameter server). Results are given in Appendix B.5.2. We compare the performance with 1 and 8 workers, when *all workers compress the same gradient*. The goal of this task is to assess the impact of the actual distribution of the normalised minibatch gradients values w.r.t. a Gaussian distribution.

3. **Task 3:** Compression of "real" gradients, with multiple workers, each worker compresses a different minibatch stochastic gradient, computed at the same parameter (as described in Section 5.5): this is the most practical setting, and we explain the resulting trade-offs, especially in terms of the distribution of stochastic gradient noise (see Panigrahi et al. 2019) and the inhomogeneity between workers. We perform this task on (i) the same setting as for task 2, and (ii) the gradients from the LS experiment introduced in Section 5.5. Results are given in Appendix B.5.2.

4. **Task 4:** Compression of "real" gradients, with multiple workers, each worker compresses a different minibatch stochastic gradient, similarly to **Task 3**. But here the normalization is considered with respect to the best estimatation of the gradient. Results are given in Appendix B.5.2.

B.5.1 Distortion for Gaussian input

Setup: We here compare all methods on a Gaussian input $x \sim \mathcal{N}(0, \mathrm{I}_D)$ for $D = 16$. Monte Carlo is performed over 10^4 repetitions. Standard deviation is negligible.

Observations. Results are provided in Table B.9. We make the following observations:

1. We first observe that in the single worker case, `Sign`, `Top-2`, `HSQ-greed` and `Stovoq-GRVQ` achieve a global error or respectively 6.4, 8.7, 9.1 and 6.8. (These errors are obtained by summing the radial and orthogonal numbers). `Stovoq` achieves an error of 11 which is slightly higher, `Rand-2`, `Polytope`, `HSQ-span` suffer a much higher errors of 110, 121, 147. This confirms the theoretical predictions.

2. We observe in practice here the fundamental differences between biased / unbiased compression methods and also methods that ensure the independence of the compression on each individual worker: while all biased methods do not benefit from the multiplicity of workers, for unbiased and independent compression, the distortion for $n = 1$ is divided by n. Here, the quadratic errors, both radial and orthogonal, are reduced by a factor 20. Overall, over 20 workers, the error obtained by `Stovoq`, with debiasing and scalar quantization is 0.5. This is by far the best method in terms of global distortion for 20 workers.

3. `Stovoq-GRVQ` **vs** `Stovoq-Unbiased`. For `Stovoq`, the application of debiasing increases the non-radial quantization distortion, by a factor of nearly 2 (from 5 to 10), while simultaneously reducing the radial distortion, form 2 to 0.5. This increase is unavoidable to

Table B.9: **Task 1:** Distortion for Gaussian inputs

Method	Sign	Top-2	Rand-2	Polytope	
Variant					norm-quant.
$n = 1$	1.0 || 5.4	4.8 || 3.9	12 || 98	5.8 || 115	5.8 || 115
$n = 20$	1.0 || 5.4	4.7 || 3.8	0.6 || 4.8	0.3 || 5.6	0.3 || 5.6

Method	HSQ-span	HSQ-greed	Stovoq		
Variant	norm-quant.	norm-quant.	GRVQ	Unbiased	Unbiased+quant.
$n = 1$	3.8 || 143	1.3 || 7.8	1.8 || 5.0	0.5 || 10.5	0.5 || 10.5
$n = 20$	0.2 || 7.0	1.3 || 7.5	1.7 || 0.25	0.03 || 0.5	0.03 || 0.5

obtain the unbiased character, that is necessary to reduce the error beyond 1. Indeed, it is important to remark the radial bias for Stovoq-GVRQ and HSQ is not negligible (1.3 and 1.8 respectively): in fact , this radial distortion is also of order $M^{-2/d}$ thus using an even larger codebook would not reduce it significantly.

4. Stovoq-Unbiased vs Stovoq-Unbiased+Scalar-Quantization. We observe that the impact of the scalar quantization is negligible here, which indicates that the impact of the scalar quantization of the norm is limited.

5. HSQ vs Stovoq-GRVQ: These two methods are somehow similar for a single worker: HSQ relies on a gain-shape decomposition with the a scalar quantization of the norm and a vector quantization of the normalized vector using spherical codebooks whereas GRVQ uses random Gaussian codebooks with a variance matched to the input variance. We observe that overall Stovoq-GRVQ slightly outperforms HSQ for $n = 1$. This is in favor of using Gaussian codebooks.

B.5.2 Distortion for neural networks gradients

Task 2: Impact of the distribution **Setup:** We compare the distortion for $n = 1$ and 8, on stochastic gradients sampled from the training of a VGG16, with SGD, at epoch 10. The gradients are partitioned into blocks of size 2^9; then each block is scaled and split into buckets of dimension $D = 16$. Those buckets are then compressed using each of the possible methods in dimension 16. The results presented are obtained using 1000 stochastic gradient .

The main objective is to compare the impact of the distribution of the gradients on the distortion of the different compressors. For example, if the stochastic gradient noise is heavy tailed (leading equivalently to sparse gradients), methods relying on sparsification, e.g., *Top-2*, is expected to perform significantly better than Gaussian random codebook.

For $n = 8$, we assume that each workers compresses the same gradient : we compute the error of $n^{-1} \sum_{i \in [n]} \hat{g}_{t,i} - g_t$, where $\hat{g}_{t,i}$ stands for the output of the i-th compressor on g_t.

In Figure B.5, we represent simultaneously the histogram of the bucket norms, and the histogram of the norms of the Gaussian vectors used in **Task 1**. This suggests a departure from the Gaussian distribution for the stochastic gradient noise.

Observations Results are provided in Table B.10. We highlight the following points.

1. Again, the unbiased version of Stovoq achieves the best distortion.

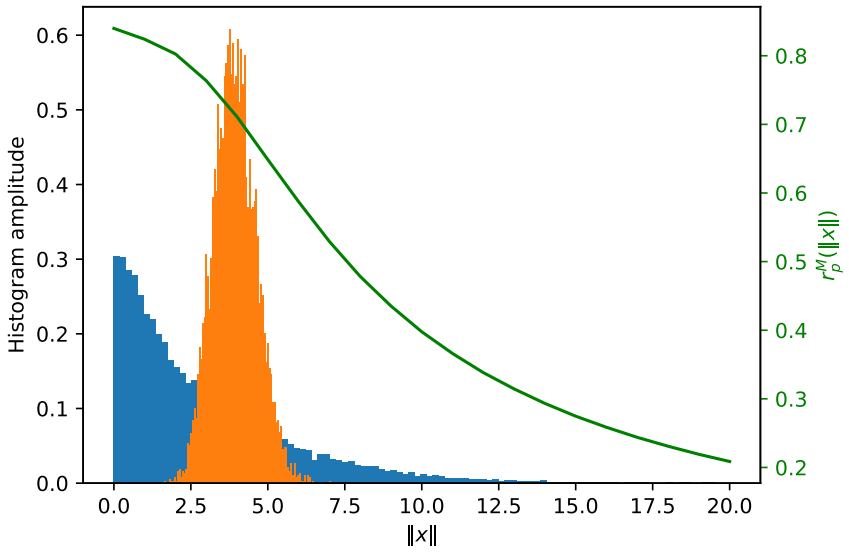


Figure B.5: Histograms of the VGG16 gradient buckets (blue), of Gaussian vectors (orange), and the radial bias for the associated dimension $D = 16$ (green).

2. Even though the distribution of the norms is very different from the norm of the Gaussian vectors (as illustrated in Figure B.5), the distortion of Stovoq is not severely impaired. Especially, the error for Stovoq for a single worker is 0.4 vs 0.7 for Top-2, while for Gaussian inputs it was 11 vs 8.7 for Top-2.

Task 3: Signal-Noise ratio on the various gradients **Setup:** We now consider that each worker computes and compresses a different stochastic gradient. More precisely, we collect samples of the stochastic gradients during an epoch: $[g_{t,1}^\top, \ldots, g_{t,n}^\top]$, where $g_{t,i}$ is computed by the worker i on distinct minibatch of size b (all the gradients are $\{g_{t,i}\}_{i=1}^n$ are evaluated for the same value of the parameters). The compressed version is denoted $\{\hat{g}_{t,i}\}_{i=1}^n$.

Table B.10: **Task 2:** empirical distortion from a sample of gradients sampled from a VGG-16 at epoch 10, (same gradients on each worker).

Method	Sign	Top-2	Rand-2	Polytope	
Variant					norm-quant.
$n = 1$	0.46‖0.35	0.35‖0.16	0.86‖4.8	0.50‖7.0	0.50‖7.1
$n = 8$	0.46‖0.35	0.35‖0.16	0.15‖0.61	0.07‖0.9	0.07‖0.9

Method	HSQ-span	HSQ-greed	Stovoq		
Variant	norm-quant.	norm-quant.	GRVQ	Unbiased	Unbiased+quant.
$n = 1$	0.24 ‖ 7.5	0.09 ‖ 0.5	0.05 ‖ 0.2	0.02 ‖ 0.36	0.02 ‖ 0.4
$n = 8$	0.07 ‖ 0.9	0.09 ‖ 0.4	0.04 ‖ 0.03	0.002 ‖ 0.05	0.003 ‖ 0.05

In the homogeneous setting, for all $i \in [n]$, $g_{t,i}(\theta_t) = \nabla F(w_t) + \epsilon_{t,i}$, with $(\epsilon_{t,i})_{t,i}$ is the stochastic gradient noise $\mathbb{E}\left[\epsilon_{t,i} \middle| \mathcal{F}_{t-1}\right] = 0$, where $\mathcal{F}_{t-1}$ collects the past observations.

The central node averages the quantized stochastic gradient sent by the workers: $\tilde{g}_t := n^{-1} \sum_{i=1}^n \hat{g}_{t,i}$. We report in Tables 5.3 and B.12 the normalized averaged error defined as

$$T^{-1} \sum_{t \in [T]} \frac{\| \frac{1}{n} \sum_{i=1}^n \hat{g}_{t,i} - g_{t,i} \|^2}{\| \frac{1}{n} \sum_{i=1}^n g_{t,i} \|^2}. \tag{B.11}$$

We here discuss in which settings we expect the multiple workers to improve w.r.t. a single worker. More precisely, we show that the impact of enforcing unbiased independent compression for the different workers increases with the "dependence" of stochastic gradients. Consider the following two cases. **Example 1: (large noise, low correlation between stochastic gradients)** each worker computes a stochastic gradient that is nearly independent of the other workers. The error made is **not** reduced by the multiplicity of workers. **Example 2: (low or no noise, strong consensus between stochastic gradients)** if each worker computes the same gradient, we recover **task 2**. The variance reduction obtained by using multiple workers and independent compressors is proportional to the number of workers. More generally, this is true when the noise is small w.r.t. the gradient of the function.

This signal/noise ratio fundamentally impacts the performance of algorithms using compressions operators: in **example 2**, it is crucial to use unbiased and independent workers, while in **example 1**, it is more important to reduce the distortion for a single worker.

Many factors impact this "consensus" between workers: first of all the mini-batch size. The noise variance is inversely proportional to b: as b increases, each stochastic gradient becomes closer to $\nabla F(w)$. More generally, all variance reduction techniques tend to increase the "consensus". On the other hand, heterogeneity between workers increases the discrepancy between gradients (but memory techniques as in `Dostovoq-DIANA` mitigate this discrepancy). Finally, performing several steps (Sebastian U. Stich 2019), as in `Local-SGD` also has a similar impact of averaging the noise over several iterations, and thus increases the consensus.

We thus evaluate all algorithms on two tasks:

1. First, on gradients extracted from the LSR task: in this task, data is distributed on all workers, that compute a batch gradient. The gradients obviously depend on the workers (each worker has access to a different subset of the data), but because the workers are homogeneous, these gradients have a strong consensus. We give the results in Table B.12. We observe that the reduction by a factor 8 is preserved when using $n = 8$. This explains why our method outperforms HSQ in Figure 5.2.

2. Second, on gradients from the VGG16 trained with SGD on CIFAR. On this task, the noise level is much higher and the consensus much weaker. This is expected in very high dimensional models and non convex objective (roughly speaking, the gradients on different workers nearly point in random descent directions). We thus do not see any strong effect of the number of workers on the distortion for $b \leq 512$. Increasing further the batch size, to $b = 4096$, we recover the gain of multiple workers. Results are given in Table B.11. The distortion is twice smaller with `Stovoq`-unbiased than with any other method. While a batch of 4096 is very high, very large batch were used in a successful training of CIFAR and ImageNet. More generally, when communication cost is a major concern, increasing the batch size and the number of local iterations is natural, to increase the quality of updates transmitted.

Table B.11: **Task 3:** normalized distortion for a mini-batch of size 4096 of a VGG-16 at epoch 10.

Method	Sign	Top-2	Rand-2	Polytope	
Variant					norm-quant.
$n = 1$	0.3‖0.2	0.5‖0.2	0.5‖6.2	0.2‖7.3	0.2‖7.3
$n = 8$	0.3‖0.1	0.5‖0.1	0.09‖1.8	0.06‖2.0	0.06‖2.0

Method	HSQ-span	HSQ-greed	Stovoq		
Variant	norm-quant.	norm-quant.	GRVQ	Unbiased	Unbiased+quant.
$n = 1$	0.2 ‖ 8.5	0.09 ‖ 0.5	0.06 ‖ 0.2	0.02 ‖ 0.4	0.02 ‖ 0.4
$n = 8$	0.09 ‖ 2.3	0.09 ‖ 0.4	0.1 ‖ 0.07	0.01 ‖ 0.1	0.01 ‖ 0.1

Table B.12: **Task 3:** normalized distortion for LSR (see Section 5.5).

Method	Sign	Top-2	Rand-2	Polytope	
Variant					norm-quant.
$K = 1$	0.05‖0.3	0.4‖0.2	0.6‖6.3	0.3‖7.3	0.3‖7.3
$K = 8$	0.04‖0.09	0.4‖0.08	0.1‖1.3	0.07‖1.4	0.07‖1.4

Method	HSQ-span	HSQ-greed	Stovoq		
Variant	norm-quant.	norm-quant.	GRVQ	Unbiased	Unbiased+quant.
$K = 1$	0.3 ‖ 9.4	0.09 ‖ 0.5	0.1 ‖ 0.3	0.03 ‖ 0.6	0.03 ‖ 0.6
$K = 8$	0.08 ‖ 1.9	0.09 ‖ 0.1	0.1 ‖ 0.06	0.008 ‖ 0.1	0.008 ‖ 0.1

Task 4: "fair" Signal-Noise ratio on the various gradients

Setup: In this last setup, we still focus on the setting in which each worker computes and compresses a different stochastic gradient. More precisely, we collect samples of the stochastic gradients $[g_{t,1}^\top, \ldots, g_{t,n}^\top]$, where $g_{t,i}$ is computed by the worker i on distinct minibatch of size b (all the gradients are $\{g_{t,i}\}_{i=1}^n$ are evaluated for the same value of the parameters, chosen at an iteration t). The corresponding compressed gradients are denoted $\{\hat{g}_{t,i}\}_{i=1}^n$.

In the homogeneous setting, for all $i \in [n]$, $g_{t,i}(\theta_t) = \nabla F(\theta_t) + \epsilon_{t,i}$, with $(\epsilon_{t,i})_{t,i}$ is the stochastic gradient noise $\mathbb{E}\left[\epsilon_{t,i} \middle| \mathcal{F}_{t-1}\right] = 0$, where $\mathcal{F}_{t-1}$ collects the past observations.

The central node averages the quantized stochastic gradient sent by the workers: $\tilde{g}_t := n^{-1} \sum_{i=1}^n \hat{g}_{t,i}$. We report in Table B.13 the "fair" normalized averaged error defined as

$$T^{-1} \sum_{t \in [T]} \frac{\| \frac{1}{n} \sum_{i=1}^n \hat{g}_{t,i} - g_{t,i} \|^2}{\| \nabla F(w_t) \|^2}. \tag{B.12}$$

W.r.t. **Task 3**, only the denominator changes. For both $n = 1$ and $n = 64$, we use $\frac{1}{64} \sum_{i=1}^{64} g_{t,i}$ as a surrogate of the gradient $\nabla F(w_t)$ (each $g_{t,k}$ being obtained with batch size of 256 samples, this corresponds to a mini-batch of size 16384).

Table B.13: **Task 4:** "fair" normalized distortion for a mini-batch of size 256 (for each worker) of a VGG16 at epoch 10.

Method	Sign	Top-2	Rand-2	Polytope	
Variant					norm-quant.
$n = 1$	6.8‖3.3	11.3‖3.8	6.1‖133.9	3.8‖157.3	3.8‖157.3
$n = 64$	0.38‖0.26	0.46‖0.19	0.37‖6.2	0.30‖7.0	0.30‖7.0

Method	HSQ-span	HSQ-greed	Stovoq		
Variant	norm-quant.	norm-quant.	GRVQ	Unbiased	Unbiased+quant.
$n = 1$	4.7‖174.5	1.8‖10.3	1.1‖3.3	0.35‖6.7	0.37‖6.9
$n = 64$	0.35 ‖ 7.5	0.17 ‖ 2.9	0.24 ‖ 0.27	0.07 ‖ 0.59	0.07 ‖ 0.59

Appendix C

Supplementary material of Chapter 6

C.1 Proofs

The complete analysis of Theorem 10 is fully provided in the following pages, and is heavily inspired by Zakerinia et al. 2022's analysis. We ask the reader to refer to Section 6.4.1 for a detailed definition of the random variables used in the analysis.

C.1.1 Preliminaries

Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space. We assume that all the random variables defined in this proof are defined on this space. Consider $I = \mathbb{N}^* \times \mathbb{N}$, together with the lexicographical ordering in I. We recall that $(a, b) \leq (c, d)$ in lexicographical ordering if and only if $a < c$ or $a = c$ and $b \leq d$. We define two families of σ-algebras $\left(\mathcal{F}_{(t,q)}\right)_{(t,q)\in I}$ and $(\mathcal{F}_t)_{t\in\mathbb{N}}$. These are defined by the relations $\mathcal{F}_0 = \{\emptyset, \Omega\}$ and, for $t \geq 1$ and $q \geq 0$,

$$\mathcal{F}_{(t,q)} = \sigma\left(\mathcal{F}_{t-1} \cup \sigma\left(\widetilde{h}^i_{t,q'} : q' \leq q, i \in [1, n]\right)\right) \tag{C.1}$$

$$\mathcal{F}_t = \sigma\left(\sigma\left(\mathcal{S}_t, E^i_t : i \in [1, n]\right) \cup \bigcup_{q\geq 0} \mathcal{F}_{(t,q)}\right). \tag{C.2}$$

Therefore, $\mathcal{F}_t$ contains all information up to the end of time step t. Additionally, $\mathcal{F}_{(t,q)}$ contains all information up to the local step q of time step t. Notice that at this point we do not have information about $\mathcal{S}_t$ and E^i_t. We define $\mathbb{E}_t$ to be the conditional expectation with respect to $\mathcal{F}_t$.

For all time steps t, local steps q, and client i, we define:

$$h^i_{t,q} = \nabla f_i\left(w^i_t - \eta \sum_{s=1}^{q-1} \widetilde{h}^i_{t,s}\right) = \mathbb{E}\left[\widetilde{h}^i_{t,q} | \mathcal{F}_{(t,q-1)}\right] \tag{C.3}$$

where $\widetilde{h}^i_{t,s}$ is defined in (6.2).

Contrary to Zakerinia et al. 2022, we do *not* assume that all clients have computed the same number of epochs upon being contacted by the server.

We start by establishing a basic, yet useful, algebraic equality in the following Lemma.

Lemma 35. *Let $s \in \mathbb{N}^*$, $a_i, b \in \mathbb{R}^d$ for $i \in [|1,s|]$ be vectors. It holds that:*

$$\left\| \frac{\sum_{i=1}^s a_i + b}{s+1} \right\|^2 - \frac{1}{s+1}\sum_{i=1}^s \|a_i\|^2 - \frac{1}{s+1}\|b\|^2 = \frac{-1}{(s+1)^2}\sum_{i=1}^s \|a_i - b\|^2 - \frac{1}{(s+1)^2}\sum_{i,j=1}^s \|a_i - a_j\|^2 \quad \text{(C.4)}$$

Proof.

$$I = \left\| \frac{\sum_{i=1}^s a_i + b}{s+1} \right\|^2 - \frac{1}{s+1}\sum_{i=1}^s \|a_i\|^2 - \frac{1}{s+1}\|b\|^2 \quad \text{(C.5)}$$

$$= (s+1)^{-2}\left(\left\langle \sum_{i=1}^s a_i + b, \sum_{i=1}^s a_i + b \right\rangle - (s+1)\sum_{i=1}^s \|a_i\|^2 - (s+1)\|b\|^2 \right). \quad \text{(C.6)}$$

We expand the inner product to obtain:

$$I = (s+1)^{-2}\left(-s\|b\|^2 + 2\sum_{i=1}^s \langle a_i, b\rangle + \sum_{i,j=1}^s \langle a_i, a_j\rangle - (s+1)\sum_{i=1}^s \|a_i\|^2 \right) \quad \text{(C.7)}$$

$$= \frac{-1}{(s+1)^2}\left(\left(\sum_{i=1}^s \|a_i\|^2 + s\|b\|^2 - 2\sum_{i=1}^s \langle a_i, b\rangle \right) + \left(s\sum_{i=1}^s \|a_i\|^2 - \sum_{i,j=1}^s \langle a_i, a_j\rangle \right) \right) \quad \text{(C.8)}$$

$$= \frac{-1}{(s+1)^2}\left(\left(\sum_{i=1}^s \|a_i - b\|^2 \right) + \left(\sum_{i,j=1}^s \|a_i - a_j\|^2 \right) \right). \quad \text{(C.9)}$$

□

Lemma 36. *Let n, d be positive integers, $a_1, a_2, ..., a_n, b \in \mathbb{R}^d$ be vectors, and $g = \frac{a_1 + ... + a_n}{n}$ be the center of mass of $a_1, ..., a_n$. Then the following identity holds:*

$$\sum_{i=1}^n \|b - a_i\|^2 = n\|b - g\|^2 + \sum_{i=1}^n \|g - a_i\|^2. \quad \text{(C.10)}$$

Proof. We may compute:

$$\sum_{i=1}^n \|b - a_i\|^2 = \sum_{i=1}^n \|b - g + g - a_i\|^2 = \sum_{i=1}^n \left(\|b - g\|^2 + \|g - a_i\|^2 + 2\langle b - g, g - a_i\rangle \right) \quad \text{(C.11)}$$

We can easily see that $\sum_{i=1}^n \langle b - g, g - a_i\rangle = 0$. The identity then follows. □

Lemma 37. *Let $X_1, \ldots, X_n$ be random variables. Moreover, let $\mathcal{S}$ be a subset of $\{1, 2, ..., n\}$ containing s elements chosen uniformly at random. Assume that $\mathcal{S}$ is independent from X_i for $i = 1, ..., n$. Then, we have:*

$$\mathbb{E}\left[\sum\nolimits_{i \in \mathcal{S}} X_i \right] = \sum_{i=1}^n \frac{s}{n}\mathbb{E}[X_i]. \quad \text{(C.12)}$$

Proof. We introduce indicator functions in the sum above and apply linearity of expectation:

$$\mathbb{E}\left[\sum_{i\in\mathcal{S}} X_i\right] = \mathbb{E}\left[\sum_{i=1}^{n} X_i \mathbb{1}_{\mathcal{S}}(i)\right] = \sum_{i=1}^{n} \mathbb{E}[X_i \mathbb{1}_{\mathcal{S}}(i)]. \tag{C.13}$$

Using that $\mathcal{S}$ is independent of each X_i we get:

$$\mathbb{E}\left[\sum_{i\in\mathcal{S}} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i]\mathbb{E}[\mathbb{1}_{\mathcal{S}}(i)] = \sum_{i=1}^{n} \frac{s}{n}\mathbb{E}[X_i]. \tag{C.14}$$

□

The following preliminary lemmas allow one to recover unbiased gradient estimates in FAVANO, and bound their variance.

Lemma 38. *Let $\{Y^q\}_{q>0}$ a collection of independent random variables such that $\mathbb{E}[Y_q] = \mu$. Let S be a positive random variable independent from the collection $\{Y^q\}_{q>0}$, and with expected value $\mathbb{E}[S] = m$. Consider $M_1 = \mathbb{E}[\frac{\mathbb{1}[S>0]}{S\mathbf{P}(S>0)}\sum_{q=1}^{S} Y^q]$, and $M_2 = \mathbb{E}[\frac{\mathbb{1}[S>0]}{\mathbb{E}[S]}\sum_{q=1}^{S} Y^q]$. M_1, M_2 are unbiased estimate of μ, i.e. $M = \mu$.*

Proof. M_1 : One can note this setting corresponds to $\alpha^i = \mathbf{P}(E_t^i > 0)(E_t^i \wedge K)$. We have

$$\begin{aligned} M_1 =& \frac{1}{\mathbf{P}(S>0)}\mathbb{E}[\mathbb{E}[\frac{\mathbb{1}[S>0]}{S}\sum_{q=1}^{S} Y^q|S]] && \text{(C.15)}\\ =& \frac{1}{\mathbf{P}(S>0)}\mathbb{E}[S\frac{\mathbb{1}[S>0]}{S}\mu] && \text{(C.16)}\\ =& \frac{1}{\mathbf{P}(S>0)}\mathbb{E}[\mathbf{1}[S>0]\mu] && \text{(C.17)}\\ =& \mu. && \text{(C.18)} \end{aligned}$$

Thus when reweighting with the (random) number of additions, one need to also take into account the $\mathbf{P}(s>0)$ term to obtain an unbiased estimate.

M_2 : This setting corresponds to $\alpha^i = \mathbb{E}[E_t^i \wedge K]$. We have

$$\begin{aligned} M_2 =& \mathbb{E}[\mathbb{E}[\frac{\mathbb{1}[S>0]}{\mathbb{E}[S]}\sum_{q=1}^{S} Y^q|S]] && \text{(C.19)}\\ =& \mathbb{E}[S\frac{\mathbb{1}[S>0]}{\mathbb{E}[S]}\mu] && \text{(C.20)}\\ =& \frac{\mathbb{E}[S\mathbb{1}[S>0]]}{\mathbb{E}[S]}\mu && \text{(C.21)}\\ =& \mu. && \text{(C.22)} \end{aligned}$$

Thus reweighting the sum with $\mathbb{E}[s]$ allows us to obtain an unbiased estimate $M_2 = \mu$. □

Lemma 39. *Let $\{Y^q\}_{q>0}$ a collection of independent random variables such that $\mathbb{E}[Y_q] = \mu$, $\mathrm{Var}(Y^q) = \mathrm{Var}(Y) < \infty$. Let S be a positive random variable independent from the collection $\{Y^q\}_{q>0}$, and with*

expected value $\mathbb{E}[S] = m$. *Consider* $M_1 = \mathbb{E}[\frac{\mathbb{1}[S>0]}{S\mathbf{P}(S>0)}\sum_{q=1}^{S} Y^q]$, *and* $M_2 = \mathbb{E}[\frac{\mathbb{1}[S>0]}{\mathbb{E}[S]}\sum_{q=1}^{S} Y^q]$. *We compute the variance :*

$$\begin{cases} \text{Var}(M_1) = \frac{1}{\mathbf{P}(s>0)^2}(\mu^2\mathbf{P}(S>0)(1-\mathbf{P}(S>0)) + \text{Var}(Y)\mathbb{E}[\frac{\mathbb{1}[S>0]}{S}]) \\ \text{Var}(M_2) = \frac{\mu^2\,\text{Var}(S)}{\mathbb{E}[S]^2} + \frac{\text{Var}(Y)}{\mathbb{E}[S]}. \end{cases} \tag{C.23}$$

Proof. M_2*:* This setting corresponds to $\alpha^i = \mathbb{E}[E_t^i \wedge K]$. We have

$$\text{Var}(M_2) = \frac{1}{\mathbb{E}[S]^2}\mathbb{E}[(\sum_{q=1}^{S} Y^q - \mu m)^2] \tag{C.24}$$

$$= \frac{1}{\mathbb{E}[S]^2}\mathbb{E}[(\sum_{q=1}^{S} (Y^q - \mu) + \mu(S-m))^2] \tag{C.25}$$

The cross products reduce to 0 in expectation, hence

$$\text{Var}(M_2) = \frac{1}{\mathbb{E}[S]^2}\mathbb{E}[\mathbb{E}[(S(Y^q-\mu)|S] + \mu^2(S-m)^2] \tag{C.26}$$

$$= \frac{1}{\mathbb{E}[S]^2}(\mathbb{E}[S]Var(Y) + \mu^2\,\text{Var}(S)) \tag{C.27}$$

M_1*:* This setting corresponds to $\alpha^i = \mathbf{P}(E_t^i > 0)(E_t^i \wedge K)$ or QuAFL (Zakerinia et al. 2022) when the same number of local epochs is done by clients. First note that $\mathbb{E}[M_1|S] = \mu\frac{\mathbb{1}(S>0)}{\mathbf{P}(S>0)}$. We have

$$\text{Var}(M_1) = \mathbb{E}[\mathbb{E}[(M_1 - \mathbb{E}[M_1|S])^2|S]] + \mathbb{E}[(\mathbb{E}[M_1|s] - \mathbb{E}[M_1])^2] \tag{C.28}$$

$$= \frac{1}{\mathbf{P}(S>0)^2}\mathbb{E}[(\frac{1}{S}\sum_{q}^{S}(Y^q-\mu)\mathbb{1}[S>0])^2] + \mathbb{E}[(\mu\mathbb{1}(S>0) - \mu\mathbf{P}(S>0))^2] \tag{C.29}$$

$$= \frac{1}{\mathbf{P}(S>0)^2}\mathbb{E}[\frac{1}{S^2}(S\,\text{Var}(Y)\mathbb{1}[S>0])^2] + \frac{1}{\mathbf{P}(S>0)^2}\mu^2\mathbb{E}[(\mathbb{1}(S>0) - \mathbf{P}(S>0))^2] \tag{C.30}$$

$$= \frac{1}{\mathbf{P}(S>0)^2}\,\text{Var}(Y)\mathbb{E}[\frac{\mathbb{1}[S>0]}{S}] + \frac{1}{\mathbf{P}(S>0)^2}\mu^2\mathbb{E}[(\mathbb{1}(S>0) - \mathbf{P}(S>0))^2] \tag{C.31}$$

$$= \frac{1}{\mathbf{P}(S>0)^2}\,\text{Var}(Y)\mathbb{E}[\frac{\mathbb{1}[S>0]}{S}] + \frac{1}{\mathbf{P}(S>0)^2}\mu^2\mathbf{P}(S>0)(1-\mathbf{P}(S>0)). \tag{C.32}$$

□

Last, but not least, we will make use of a result from Koloskova, Sebastian U Stich, and Jaggi 2022 to optimize learning rates and obtain sharp complexity bounds:

Lemma 40. *Assume* ***A****10 to* ***A****13 and consider problem* (6.1). *If the output of an optimization algorithm with step size* η *has an expected error upper bounded by* $\frac{r_0}{\eta(T+1)} + b\eta + e\eta^2$, *and if* η *satisfies the constraints* $\eta \leq \min((\frac{r_0}{b(T+1)})^{\frac{1}{2}}, (\frac{r_0}{c(T+1)})^{\frac{1}{3}}, \frac{1}{d})$ *for some non-negative values* r_0, b, d, e, *then*

the number of communication rounds required to reach ϵ accuracy is lower bounded by

$$\frac{36br_0}{\epsilon^2} + \frac{15r_0\sqrt{e}}{\epsilon^{\frac{3}{2}}} + \frac{3dr_0}{\epsilon}. \tag{C.33}$$

Proof. Consider ψ_T a positive variable such that

$$\psi_T \leq \frac{r_0}{\eta(T+1)} + b\eta + e\eta^2 \tag{C.34}$$

for any positive step size verifying $\eta \leq \min((\frac{r_0}{b(T+1)})^{\frac{1}{2}}, (\frac{r_0}{e(T+1)})^{\frac{1}{3}}, \frac{1}{d})$. Then Koloskova, Sebastian U Stich, and Jaggi 2022 shows the following inequality:

$$\psi_T \leq 2(\frac{br_0}{T+1})^{\frac{1}{2}} + e^{\frac{1}{3}}(\frac{r_0}{T+1})^{\frac{2}{3}} + \frac{dr_0}{T+1}. \tag{C.35}$$

In order to reach an ϵ precision, we bound each term by $\frac{\epsilon}{3}$, and deduce the following lower bound

$$T \geq \frac{36br_0}{\epsilon^2} + \frac{15r_0\sqrt{e}}{\epsilon^{\frac{3}{2}}} + \frac{3dr_0}{\epsilon}. \tag{C.36}$$

□

C.1.2 Useful Lemmas

A key result of our analysis is the upper bound on the change (in expected value) of the potential function Φ_t. Recall that Φ_t is defined by equation:

$$\Phi_t = \|w_t - \mu_t\|^2 + \sum_{i=1}^{n} \|w_t^i - \mu_t\|^2. \tag{C.37}$$

In the next lemma, we show that Φ_t exhibits a contractive property, which allows us to bound its value through the execution of the optimization algorithm.

Proof of Lemma 9

Proof. Consider the following quantities:

$$\mu_t = \left(w_t + \sum_{i=1}^{n} w_t^i\right)/(n+1) \tag{C.38}$$

$$G_{t+1} = -\frac{1}{n+1}\eta \sum_{i \in \mathcal{S}_{t+1}} \check{h}_{t+1}^i \tag{C.39}$$

where $\check{h}_{t+1}^i = \frac{1}{\mathbf{P}(E_{t+1}^i > 0)(E_{t+1}^i \wedge K)} \widetilde{h}_{t+1}^i$ or $\check{h}_{t+1}^i = \frac{1}{\mathbb{E}[E_{t+1}^i \wedge K]} \widetilde{h}_{t+1}^i$. And recall the updates rules:

$$\begin{cases} w_{t+1} & = \frac{1}{s+1}\left(w_t + \sum_{i \in \mathcal{S}_{t+1}} w_t^i\right) + \frac{n+1}{s+1} G_{t+1}, \\ w_t^i & = w_t; \text{ for } i \in \mathcal{S}_t \\ w_t^i & = w_{t-1}^i; \text{ for } i \notin \mathcal{S}_t. \end{cases} \tag{C.40}$$

With these definitions, we get

$$\mu_{t+1} = \frac{s+1}{n+1} w_{t+1} + \frac{1}{n+1} \sum_{i \notin \mathcal{S}_{t+1}} w_t^i = \mu_t + G_{t+1}\,. \tag{C.41}$$

We can now compute the difference of potential:

$$\Phi_{t+1} - \Phi_t = \sum_{i \in \mathcal{S}_{t+1}} \left(\left\| w_{t+1} - \mu_{t+1} \right\|^2 - \left\| w_t^i - \mu_t \right\|^2 \right) + \left\| w_{t+1} - \mu_{t+1} \right\|^2 - \left\| w_t - \mu_t \right\|^2 \tag{C.42}$$

$$+ \sum_{i \notin \mathcal{S}_{t+1}} \left(\left\| w_t^i - \mu_{t+1} \right\|^2 - \left\| w_t^i - \mu_t \right\|^2 \right). \tag{C.43}$$

We can rewrite this equation into a more convenient form:

$$\Phi_{t+1} - \Phi_t = (s+1) \left\| w_{t+1} - \mu_{t+1} \right\|^2 - \sum_{i \in \mathcal{S}_{t+1}} \|w_t^i - \mu_t\|^2 - \|w_t - \mu_t\|^2 + \sum_{i \notin \mathcal{S}_{t+1}} \left(\left\| w_t^i - \mu_t - G_{t+1} \right\|^2 - \left\| w_t^i - \mu_t \right\|^2 \right). \tag{C.44}$$

Step 1. First, notice that:

$$\sum_{i \notin \mathcal{S}_{t+1}} \left(\left\| w_t^i - \mu_t - G_{t+1} \right\|^2 - \left\| w_t^i - \mu_t \right\|^2 \right) = \sum_{i \notin \mathcal{S}_{t+1}} \left(\|G_{t+1}\|^2 - 2\langle w_t^i - \mu_t, G_{t+1} \rangle \right)$$

$$= (n-s)\|G_{t+1}\|^2 - 2\langle \sum_{i \notin \mathcal{S}_{t+1}} (w_t^i - \mu_t), G_{t+1} \rangle. \tag{C.45}$$

Step 2. Next, we compute the first term of Equation (C.44):

$$(s+1) \left\| w_{t+1} - \mu_{t+1} \right\|^2 = (s+1) \left\| \frac{(w_t - \mu_t) + \sum_{i \in \mathcal{S}_{t+1}} (w_t^i - \mu_t)}{s+1} + \frac{n+1}{s+1} G_{t+1} - G_{t+1} \right\|^2 \tag{C.46}$$

$$= (s+1) \left\| \frac{(w_t - \mu_t) + \sum_{i \in \mathcal{S}_{t+1}} (w_t^i - \mu_t)}{s+1} \right\|^2 + 2 \left\langle (w_t - \mu_t) + \sum_{i \in \mathcal{S}_{t+1}} (w_t^i - \mu_t), \frac{n+1}{s+1} G_{t+1} - G_{t+1} \right\rangle \tag{C.47}$$

$$+ \frac{(n-s)^2}{s+1} \|G_{t+1}\|^2. \tag{C.48}$$

We apply Young's inequality ($\langle x, y \rangle \leq \beta \|x\|^2 + 1/(4\beta)\|y\|^2$) and Jensen inequality to get:

$$\left\langle (w_t - \mu_t) + \sum_{i \in \mathcal{S}_{t+1}} (w_t^i - \mu_t), \frac{n+1}{s+1} G_{t+1} \right\rangle$$

$$\leq \frac{\alpha(n+1)}{s+1} \|w_t - \mu_t\|^2 + \frac{\alpha(n+1)}{s+1} \sum_{i \in \mathcal{S}_{t+1}} \|w_t^i - \mu_t\|^2 + \frac{n+1}{4\alpha} \|G_{t+1}\|^2. \tag{C.49}$$

Applying Lemma 35, we get:

$$(s+1)\left\|\frac{(w_t-\mu_t)+\sum_{i\in\mathcal{S}_{t+1}}(w_t^i-\mu_t)}{s+1}\right\|^2$$
$$=\left(\sum_{i\in\mathcal{S}_{t+1}}\|w_t^i-\mu_t\|^2\right)+\|w_t-\mu_t\|^2-\frac{1}{s+1}\sum_{i\in\mathcal{S}_{t+1}}\|w_t^i-w_t\|^2-\frac{1}{s+1}\sum_{i,j\in\mathcal{S}_{t+1}}\|w_t^i-w_t^j\|^2 \quad \text{(C.50)}$$

Combining the results above, we get:

$$(s+1)\left\|w_{t+1}-\mu_{t+1}\right\|^2\le\left\|w_t-\mu_t\right\|^2+\sum_{i\in\mathcal{S}_{t+1}}\left\|w_t^i-\mu_t\right\|^2-\frac{1}{s+1}\sum_{i\in\mathcal{S}_{t+1}}\|w_t^i-w_t\|^2 \quad \text{(C.51)}$$
$$+\frac{2\alpha(n+1)}{s+1}\|w_t-\mu_t\|^2+\frac{2\alpha(n+1)}{s+1}\sum_{i\in\mathcal{S}_{t+1}}\|w_t^i-\mu_t\|^2+\left(\frac{(n-s)^2}{s+1}+\frac{n+1}{2\alpha}\right)\|G_{t+1}\|^2 \quad \text{(C.52)}$$
$$-2\langle(w_t-\mu_t)+\sum_{i\in\mathcal{S}_{t+1}}(w_t^i-\mu_t),G_{t+1}\rangle. \quad \text{(C.53)}$$

Using simple algebra, this relation can be rewritten as :

$$(s+1)\left\|w_{t+1}-\mu_{t+1}\right\|^2\le\left(1+\frac{2\alpha(n+1)}{s+1}\right)\left\|w_t-\mu_t\right\|^2 \quad \text{(C.54)}$$
$$+\left(1+\frac{2\alpha(n+1)}{s+1}\right)\sum_{i\in\mathcal{S}_{t+1}}\|w_t^i-\mu_t\|^2+\left(\frac{n+1}{2\alpha}+\frac{(n-s)^2}{s+1}\right)\|G_{t+1}\|^2 \quad \text{(C.55)}$$
$$-2\left\langle(w_t-\mu_t)+\sum_{i\in\mathcal{S}_{t+1}}(w_t^i-\mu_t),G_{t+1}\right\rangle-\frac{1}{s+1}\sum_{i\in\mathcal{S}_{t+1}}\|w_t^i-w_t\|^2. \quad \text{(C.56)}$$

Step 3. We combine the results above to get bound $\Phi_{t+1}-\Phi_t$. Plugging (C.45) into (C.44), (C.56), and using (see (6.5))

$$(w_t-\mu_t)+\sum_{i\in\mathcal{S}_{t+1}}(w_t^i-\mu_t)+\sum_{i\notin\mathcal{S}_{t+1}}(w_t^i-\mu_t)=0,$$

we get that

$$\Phi_{t+1}-\Phi_t\le\frac{2\alpha(n+1)}{s+1}\left\|w_t-\mu_t\right\|^2+\frac{2\alpha(n+1)}{s+1}\sum_{i\in\mathcal{S}_{t+1}}\|w_t^i-\mu_t\|^2+\left((n-s)+\frac{n+1}{2\alpha}+\frac{(n-s)^2}{s+1}\right)\|G_{t+1}\|^2 \quad \text{(C.57)}$$
$$-\frac{1}{s+1}\sum_{i\in\mathcal{S}_{t+1}}\|w_t^i-w_t\|^2. \quad \text{(C.58)}$$

Step 4. We now apply Lemma 37 to take expectations in the inequality above. We have:

$$\mathbb{E}\left[\sum\nolimits_{i\in\mathcal{S}_{t+1}}\|w_t^i-\mu_t\|^2\right]=\sum_{i=1}^{n}\frac{s}{n}\mathbb{E}[\|w_t^i-\mu_t\|^2]. \quad \text{(C.59)}$$

Moreover, we have:

$$\|G_{t+1}\|^2 \leq \frac{s}{(n+1)^2}\eta^2 \sum_{i\in\mathcal{S}_{t+1}} \left\|\check{h}^i_{t+1}\right\|^2. \tag{C.60}$$

Therefore, also by Lemma 37, we have:

$$\mathbb{E}[\|G_{t+1}\|^2] \leq \frac{s^2}{n(n+1)^2}\eta^2 \sum_{i=1}^{n} \mathbb{E}[\left\|\check{h}^i_{t+1}\right\|^2]. \tag{C.61}$$

Step 5. Now we derive the final inequality. We have:

$$\mathbb{E}[\Phi_{t+1}] - \mathbb{E}[\Phi_t] \leq \frac{2\alpha(n+1)}{s+1}\mathbb{E}\left\|w_t - \mu_t\right\|^2 + \frac{2\alpha(n+1)s}{(s+1)n}\sum_{i=1}^{n}\mathbb{E}\|w^i_t - \mu_t\|^2 + \tag{C.62}$$

$$+\left((n-s) + \frac{n+1}{2\alpha} + \frac{(n-s)^2}{s+1}\right)\frac{s^2}{n(n+1)^2}\eta^2\sum_{i=1}^{n}\mathbb{E}[\|\check{h}^i_{t+1}\|^2] \tag{C.63}$$

$$-\frac{s}{n(s+1)}\sum_{i=1}^{n}\mathbb{E}\|w^i_t - w_t\|^2. \tag{C.64}$$

We can apply Lemma 36 to the above inequality's last line with $a_i = w^i_t$ for $i = 1,...,n$, and $a_{n+1} = w_t$, and $b = w_t$:

$$\sum_{i=1}^{n}\mathbb{E}\|w^i_t - w_t\|^2 = (n+2)\mathbb{E}\|w_t - \mu_t\|^2 + \sum_{i=1}^{n}\mathbb{E}\|w^i_t - \mu_t\|^2. \tag{C.65}$$

This allows us to simplify:

$$\mathbb{E}[\Phi_{t+1}] - \mathbb{E}[\Phi_t] \leq \left(\frac{2\alpha(n+1)}{s+1} - \frac{s(n+2)}{n(s+1)}\right)\mathbb{E}\left\|w_t - \mu_t\right\|^2 \tag{C.66}$$

$$+\left(\frac{2\alpha(n+1)s}{(s+1)n} - \frac{s}{n(s+1)}\right)\sum_{i=1}^{n}\mathbb{E}\|w^i_t - \mu_t\|^2 \tag{C.67}$$

$$+\left((n-s) + \frac{n+1}{2\alpha} + \frac{(n-s)^2}{s+1}\right)\frac{s^2}{n(n+1)^2}\eta^2\sum_{i=1}^{n}\mathbb{E}[\|\check{h}^i_{t+1}\|^2]. \tag{C.68}$$

We let $\alpha = \frac{1}{4(n+1)}$ and define $\kappa = \frac{1}{n}\left(\frac{s(n-s)}{2(n+1)(s+1)}\right)$ to simply as following:

$$\mathbb{E}[\Phi_{t+1}] - \mathbb{E}[\Phi_t] \leq -\kappa\mathbb{E}\left\|w_t - \mu_t\right\|^2 - \kappa\sum_{i=1}^{n}\mathbb{E}\|w^i_t - \mu_t\|^2 \tag{C.69}$$

$$+\left((n-s) + 2(n+1)^2 + \frac{(n-s)^2}{s+1}\right)\frac{s^2}{n(n+1)^2}\eta^2\sum_{i=1}^{n}\mathbb{E}[\|\check{h}^i_{t+1}\|^2]. \tag{C.70}$$

We now introduce Φ_t on the right-hand side of the inequality above:

$$\mathbb{E}[\Phi_{t+1}] - \mathbb{E}[\Phi_t] \leq -\kappa \mathbb{E}[\Phi_t] + \left((n-s) + 2(n+1)^2 + \frac{(n-s)^2}{s+1} \right) \frac{s^2}{n(n+1)^2} \eta^2 \sum_{i=1}^{n} \mathbb{E}[\|\check{h}_{t+1}^i\|^2]. \quad \text{(C.71)}$$

We reorganize the terms to make the final statement:

$$\mathbb{E}[\Phi_{t+1}] \leq (1-\kappa)\mathbb{E}[\Phi_t] + 3\frac{s^2}{n}\eta^2 \sum_{i=1}^{n} \mathbb{E}[\|\check{h}_{t+1}^i\|^2]. \quad \text{(C.72)}$$

□

Bound expected local gradient Variance

In the next lemma we show that an analogous version of **A**12 holds in expectation.

Lemma 41. *Assume **A**12. Let $t \geq 1$ be a time step, q a local step, and i a client. We have:*

$$\mathbb{E}[\|\widetilde{h}_{t,q}^i\|^2] \leq \mathbb{E}[\|h_{t,q}^i\|^2] + \sigma^2. \quad \text{(C.73)}$$

Proof. We refer the reader to the filtrations $\left(\mathcal{F}_{(t,q)}\right)_{(t,q)\in I}$ defined in (C.1). By the tower property of conditional expectation, we have:

$$\mathbb{E}[\|\widetilde{h}_{t,q}^i\|^2] = \mathbb{E}\left[\mathbb{E}\left[\|\widetilde{h}_{t,q}^i\|^2 \,\middle|\, \mathcal{F}_{(t,q-1)}\right]\right]. \quad \text{(C.74)}$$

We denote by $h_{t,q}^i$ the gradient of f_i at $w_{t-1}^i - \sum_{s=1}^{q-1} \eta \widetilde{h}_{t,s}^i$. By construction, $\mathbb{E}\left[\widetilde{h}_{t,q}^i \,\middle|\, \mathcal{F}_{(t,q-1)}\right] = h_{t,q}^i$. By **A**12, we conclude that:

$$\mathbb{E}\left[\mathbb{E}\left[\|\widetilde{h}_{t,q}^i\|^2 \,\middle|\, \mathcal{F}_{(t,q-1)}\right]\right] \leq \mathbb{E}[\sigma^2 + \|h_{t,q}^i\|^2] = \mathbb{E}[\|h_{t,q}^i\|^2] + \sigma^2. \quad \text{(C.75)}$$

□

In the following we define $w_{t,q}^i = w_{t-1}^i - \sum_{s=1}^{q} \eta \widetilde{h}_{t,s}^i$. This is the model of client i, at time step t and at local step q. Therefore, $\widetilde{h}_{t,q}^i$ is a stochastic gradient of f_i computed at the point $w_{t,q-1}^i$. The next lemma sets an upper bound on the gradients of quantized weights for each client. We show that such quantities can be upper bounded by an expression containing the true gradient at the "average model" μ_t. For any agent i, and time step $t \geq 0$, define the quantity:

$$B_t^i = \frac{\sigma^2}{K^2} + 16L^2\mathbb{E}\left\|w_t^i - \mu_t\right\|^2 + 8\mathbb{E}\left\|\nabla f_i(\mu_t)\right\|^2. \quad \text{(C.76)}$$

Lemma 42. *Assume **A**12, and that the learning rate η satisfies $\eta < \frac{1}{4LK^2}$. Under the assumptions of Lemma 41, then, for any agent i, time step $t \geq 0$ and local step q, the following inequality holds:*

$$\mathbb{E}[\|h_{t+1,q}^i\|^2] \leq B_t^i. \quad \text{(C.77)}$$

Proof. We will prove the result by induction on q. Initially, we show inequalities that are

necessary for both the base case $q = 1$ and for the general case.

$$\mathbb{E}[\|h^i_{t+1,q}\|^2] = \mathbb{E}\left\|\nabla f_i\left(w^i_{t+1,q-1}\right)\right\|^2 \tag{C.78}$$

We introduce the gradient on the virtual point μ_t:

$$\mathbb{E}\left\|\nabla f_i(w^i_{t+1,q-1})\right\|^2 \leq \mathbb{E}\left\|\left(\nabla f_i\left(w^i_t - \sum_{s=1}^{q-1}\eta\widetilde{h}^i_{t+1,s}\right) - \nabla f_i(\mu_t)\right) + \nabla f_i(\mu_t)\right\|^2 \tag{C.79}$$

$$\leq 2\mathbb{E}\left\|\nabla f_i\left(w^i_t - \sum_{s=1}^{q-1}\eta\widetilde{h}^i_{t+1,s}\right) - \nabla f_i(\mu_t)\right\|^2 + 2\mathbb{E}\left\|\nabla f_i(\mu_t)\right\|^2 \tag{C.80}$$

$$\leq 2L^2\mathbb{E}\left\|w^i_t - \sum_{s=1}^{q-1}\eta\widetilde{h}^i_{t+1,s} - \mu_t\right\|^2 + 2\mathbb{E}\left\|\nabla f_i(\mu_t)\right\|^2 \tag{C.81}$$

$$\leq 4L^2\mathbb{E}\left\|w^i_t - \mu_t\right\|^2 + 4\eta^2L^2(q-1)\sum_{s=1}^{q-1}\mathbb{E}\left\|\widetilde{h}^i_{t+1,s}\right\|^2 + 2\mathbb{E}\left\|\nabla f_i(\mu_t)\right\|^2. \tag{C.82}$$

Applying this result with $q = 1$ shows that (C.77) holds. For $q \geq 1$, we proceed by induction. First, we apply Lemma 41:

$$\mathbb{E}\left\|\nabla f_i(w^i_{t+1,q-1})\right\|^2 \leq 4L^2\mathbb{E}\left\|w^i_t - \mu_t\right\|^2 + 4\eta^2L^2(q-1)\sum_{s=1}^{q-1}\left(\mathbb{E}\left\|h^i_{t+1,s}\right\|^2 + \sigma^2\right) + 2\mathbb{E}\left\|\nabla f_i(\mu_t)\right\|^2. \tag{C.83}$$

Using the induction hypothesis, we have:

$$\mathbb{E}\left\|\nabla f_i(w^i_{t+1,q-1})\right\|^2 \leq 4L^2\mathbb{E}\left\|w^i_t - \mu_t\right\|^2 + 4\eta^2L^2(q-1)^2\left(B^i_t + \sigma^2\right) + 2\mathbb{E}\left\|\nabla f_i(\mu_t)\right\|^2. \tag{C.84}$$

Now we use that $\eta < \frac{1}{4LK^2}$ and $q \leq K$.

$$\mathbb{E}\left\|\nabla f_i(w^i_{t+1,q-1})\right\|^2 \leq 4L^2\mathbb{E}\left\|w^i_t - \mu_t\right\|^2 + \frac{4}{16K^2}\left(B^i_t + \sigma^2\right) + 2\mathbb{E}\left\|\nabla f_i(\mu_t)\right\|^2 \tag{C.85}$$

$$\leq 4L^2\mathbb{E}\left\|w^i_t - \mu_t\right\|^2 + \frac{\sigma^2}{4K^2} + 2\mathbb{E}\left\|\nabla f_i(\mu_t)\right\|^2 + \frac{B^i_t}{4} \tag{C.86}$$

Finally, we get:

$$\mathbb{E}\left\|h^i_{t+1,q}\right\|^2 \leq 8L^2\mathbb{E}\left\|w^i_t - \mu_t\right\|^2 + \frac{\sigma^2}{2K^2} + 4\mathbb{E}\left\|\nabla f_i(\mu_t)\right\|^2 + \frac{B^i_t}{2} \tag{C.87}$$

It then suffices to see that the last term above is upper bounded by B^i_t. □

In Lemma 42 we have found a way to bound $h^i_{t+1,q}$. The goal of the next lemma is to use this result to find an upper bound for the stochastic gradients $\widetilde{h}^i_{t+1,q}$.

Corollary 43. *Under the assumptions of Lemma 42, for any local step q, agent i, and step $t \geq 0$, the following holds:*

$$\mathbb{E}\|\widetilde{h}^i_{t+1,q}\|^2 \leq \left(\sigma^2 + B^i_t\right). \tag{C.88}$$

The next lemma gives an upper bound on the difference of the gradient at the average model μ_t and the expected value of the updates computed by the clients. In particular, the next lemma shows how well the $h^i_{t+1,q}$ approximate the true gradients $\nabla f_i(\mu_t)$. For any $i \in \{1,\dots,n\}$ and $t \geq 0$, define

$$C^i_t = 4L^2\eta^2K^2\sigma^2 + 20L^2\mathbb{E}\left\|w^i_t - \mu_t\right\|^2 + 16L^2\eta^2K^2\mathbb{E}\left\|\nabla f_i(\mu_t)\right\|^2. \tag{C.89}$$

Lemma 44. *Assume the learning rate η satisfies $\eta < \frac{1}{2LK}$. Under the assumptions of Corollary 43, for any $i \in \{1,\dots,n\}$, $t \geq 0$ and $q \in \{1,\dots,K\}$, it holds that :*

$$\mathbb{E}\left\|\nabla f_i(\mu_t) - h^i_{t+1,q}\right\|^2 \leq C^i_t. \tag{C.90}$$

Proof.

$$\mathbb{E}\left\|\nabla f_i(\mu_t) - h^i_{t+1,q}\right\|^2 = \mathbb{E}\left\|\nabla f_i(\mu_t) - \nabla f_i\left(w^i_{t+1,q-1}\right)\right\|^2 \tag{C.91}$$

$$\leq L^2\mathbb{E}\left\|\mu_t - w^i_{t+1,q-1}\right\|^2. \tag{C.92}$$

We can now decompose the client drift as:

$$\mathbb{E}\|\mu_t - w^i_{t+1,q-1}\|^2 = \mathbb{E}\left\|\mu_t - w^i_t + \sum_{s=1}^{q}\eta\widetilde{h}^i_{t+1,s}\right\|^2 \tag{C.93}$$

$$\leq 2\mathbb{E}\left\|w^i_t - \mu_t\right\|^2 + 2\eta^2\mathbb{E}\left\|\sum_{s=1}^{q}\widetilde{h}^i_{t+1,s}\right\|^2 \tag{C.94}$$

$$\leq 2\mathbb{E}\left\|w^i_t - \mu_t\right\|^2 + 2\eta^2 q\sum_{s=1}^{q}\mathbb{E}\left\|\widetilde{h}^i_{t+1,s}\right\|^2. \tag{C.95}$$

By using Corollary 43, we get

$$\mathbb{E}\|\mu_t - w^i_{t+1,q-1}\|^2 \leq 2\mathbb{E}\left\|w^i_t - \mu_t\right\|^2 + 2\eta^2K^2(\sigma^2 + B^i_t), \tag{C.96}$$

where B^i_t is defined in (C.76). Combining the two bounds, we get:

$$\mathbb{E}\left\|\nabla f_i(\mu_t) - h^i_{t+1,q}\right\|^2 \leq 2L^2\mathbb{E}\left\|w^i_t - \mu_t\right\|^2 + 2L^2\eta^2K^2(\sigma^2 + B^i_t). \tag{C.97}$$

Expanding the above inequality:

$$\mathbb{E}\left\|\nabla f_i(\mu_t) - h^i_{t+1,q}\right\|^2 \leq 2L^2\mathbb{E}\left\|w^i_t - \mu_t\right\|^2 \tag{C.98}$$

$$+ 2L^2\eta^2K^2(\sigma^2 + \frac{\sigma^2}{K^2} + 16L^2\mathbb{E}\left\|w^i_t - \mu_t\right\|^2 + 8\mathbb{E}\left\|\nabla f_i(\mu_t)\right\|^2) \tag{C.99}$$

$$\leq 4L^2\eta^2K^2\sigma^2 + 20L^2\mathbb{E}\left\|w^i_t - \mu_t\right\|^2 + 16L^2\eta^2K^2\mathbb{E}\left\|\nabla f_i(\mu_t)\right\|^2. \tag{C.100}$$

As claimed. □

Lemma 45. *Under assumptions of Lemma 44, we have*

$$\mathbb{E}\Big\langle \nabla f\left(\mu_t\right), -h_{t+1,q}^i \Big\rangle \leq \frac{\mathbb{E}[\|\nabla f(\mu_t)\|^2]}{4} + C_t^i - \mathbb{E}[\langle \nabla f(\mu_t), \nabla f_i(\mu_t)\rangle]. \tag{C.101}$$

Proof. We may manipulate the equation above to get:

$$\mathbb{E}\Big\langle \nabla f\left(\mu_t\right), -h_{t+1}^i \Big\rangle = \mathbb{E}\Big\langle \nabla f\left(\mu_t\right), \nabla f_i\left(\mu_t\right) - h_{t+1,q}^i \Big\rangle - \mathbb{E}\left\langle \nabla f\left(\mu_t\right), \nabla f_i\left(\mu_t\right)\right\rangle. \tag{C.102}$$

Using Young's inequality together with Lemma 44 we can upper bound $\mathbb{E}\Big\langle \nabla f\left(\mu_t\right), \nabla f_i\left(\mu_t\right) - h_{t+1,q}^i \Big\rangle$ by

$$\frac{\mathbb{E}\left\|\nabla f\left(\mu_t\right)\right\|^2}{4} + \mathbb{E}\left\|\nabla f_i\left(\mu_t\right) - h_{t+1,q}^i\right\|^2 \leq \frac{\mathbb{E}\left\|\nabla f\left(\mu_t\right)\right\|^2}{4} + C_t^i. \tag{C.103}$$

This concludes the proof. □

The next lemma incorporates the idea behind gradient descent. We find an upper bound for the expected value of the inner product between the true gradients $\nabla f(\mu_t)$ and the client updates $-\widetilde{h}_{t+1}^i$. In particular, we seek to show that, in expectation $-\widetilde{h}_{t+1}^i$ is a descent direction for the function f. In other words, that the updates proposed by the clients contribute to getting μ_t closer to a local minimum.

Lemma 46. *Assume **A**13. We denote by E_{t+1}^i the effective number of locals steps done by a client i while being called by the central server. We clip to K and consider the random variable $E_{t+1}^i \wedge K$. Under assumptions of Lemma 44, and for any time step $t > 0$, we have:*

$$\sum_{i=1}^{n} \mathbb{E}\langle \nabla f\left(\mu_t\right), -\frac{1}{\alpha^i}\widetilde{h}_{t+1}^i \rangle \leq 20L^2\mathbb{E}\left[\Phi_t\right] + 4nL^2\eta^2K^2(\sigma^2 + 4G^2) + n\Big(16L^2\eta^2K^2B^2 - \frac{3}{4}\Big)\mathbb{E}\left\|\nabla f\left(\mu_t\right)\right\|^2. \tag{C.104}$$

for

$$\alpha^i = \begin{cases} \mathbf{P}(E_{t+1}^i > 0) E_{t+1}^i \wedge K \\ \mathbb{E}[E_{t+1}^i \wedge K]. \end{cases} \tag{C.105}$$

Proof. Initially, we introduce indicator random variables in order to work with the E_{t+1}^i terms. We also introduce Z^i the following random variable as :

$$Z^i = \begin{cases} 0 \text{ if } E_{t+1}^i < 1 \\ \langle \nabla f(\mu_t), -\frac{1}{\alpha^i}\sum_q^{E_{t+1}^i} \widetilde{h}_{t+1,q} \rangle \text{ if } 1 \leq E_{t+1}^i \leq K \\ \langle \nabla f(\mu_t), -\frac{1}{\alpha^i}\sum_q^{K} \widetilde{h}_{t+1,q} \text{ if } E_{t+1}^i > K. \end{cases} \tag{C.106}$$

We first claim that $\mathbb{E}\langle \nabla f\left(\mu_t\right), \widetilde{h}_{t+1,q}^i \rangle = \mathbb{E}\langle \nabla f\left(\mu_t\right), h_{t+1,q}^i \rangle$. This result follows from the following algebraic manipulations. First, notice that:

$$\mathbb{E}\langle \nabla f\left(\mu_t\right), \widetilde{h}_{t+1,q}^i \rangle = \mathbb{E}\langle \nabla f\left(\mu_t\right), \widetilde{h}_{t+1,q}^i - h_{t+1,q}^i \rangle + \mathbb{E}\langle \nabla f\left(\mu_t\right), h_{t+1,q}^i \rangle. \tag{C.107}$$

Now, we recall that $\mathbb{E}[\langle \nabla f(\mu_t), \widetilde{h}^i_{t+1,q} - h^i_{t+1,q}\rangle | \mathcal{F}_{(t+1,q-1)}] = 0$. Therefore:

$$\mathbb{E}\langle \nabla f(\mu_t), \widetilde{h}^i_{t+1,q} - h^i_{t+1,q}\rangle = \mathbb{E}[\mathbb{E}[\langle \nabla f(\mu_t), \widetilde{h}^i_{t+1,q} - h^i_{t+1,q}\rangle | \mathcal{F}_{(t+1,q-1)}]] = 0. \tag{C.108}$$

Now notice that E^i_{t+1} is independent of the random variables $\nabla f(\mu_t)$ and $\widetilde{h}^i_{t+1,q}$. Therefore:

$$\sum_{i=1}^{n} \mathbb{E}\left\langle \nabla f(\mu_t), -\frac{1}{\alpha^i}\widetilde{h}^i_{t+1}\right\rangle = \sum_{i}^{n} \mathbb{E}[Z^i] \tag{C.109}$$

$$= \sum_{i=1}^{n} \mathbb{E}[\mathbb{1}[E^i_{t+1} > K]\langle \nabla f(\mu_t), -\frac{1}{\alpha^i}\sum_{q}^{K} \widetilde{h}^i_{t+1,q}\rangle \tag{C.110}$$

$$+ \mathbb{1}[1 \le E^i_{t+1} \le K]\langle \nabla f(\mu_t), -\frac{1}{\alpha^i}\sum_{q}^{K} \mathbb{1}[q \le E^i_{t+1}]\widetilde{h}^i_{t+1,q}] \tag{C.111}$$

$$= \sum_{i}^{n}\sum_{q}^{K} \mathbb{E}[\frac{\mathbb{1}[E^i_{t+1} \ge 1]\mathbb{1}[q \le (E^i_{t+1} \wedge K)]}{\alpha^i}\langle \nabla f(\mu_t), -\widetilde{h}^i_{t+1,q}\rangle] \tag{C.112}$$

$$= \sum_{i}^{n} \mathbb{E}[\frac{\mathbb{1}[E^i_{t+1} \ge 1]}{\alpha^i}\sum_{q}^{E^i_{t+1}\wedge K}\langle \nabla f(\mu_t), -\widetilde{h}^i_{t+1,q}\rangle]. \tag{C.113}$$

We now apply Lemma 45 to obtain the following:

$$\mathbb{E}\left\langle \nabla f(\mu_t), -\frac{1}{\alpha^i}\widetilde{h}^i_{t+1}\right\rangle \le \mathbb{E}[\frac{\mathbb{1}[E^i_{t+1} \ge 1]}{\alpha^i}\sum_{q}^{E^i_{t+1}\wedge K} \mathbb{E}[\langle \nabla f(\mu_t), -\widetilde{h}^i_{t+1,q}\rangle | E^i_{t+1}]] \tag{C.114}$$

$$\le \mathbb{E}[\frac{\mathbb{1}[E^i_{t+1} \ge 1]}{\alpha^i}\sum_{q}^{E^i_{t+1}\wedge K}(\frac{\mathbb{E}[\|\nabla f(\mu_t)\|^2]}{4} + C^i_t - \mathbb{E}[\langle \nabla f(\mu_t), \nabla f_i(\mu_t)\rangle])] \tag{C.115}$$

$$\tag{C.116}$$

We can make use of Lemma 38 with $\alpha^i = \mathbf{P}(E^i_{t+1} > 0)E^i_{t+1} \wedge K$ or $\alpha^i = \mathbb{E}[E^i_{t+1} \wedge K]$, and $S = E^i_{t+1} \wedge K$, $Y_q = \mathbb{E}[\langle \nabla f(\mu_t), -\widetilde{h}^i_{t+1,q}\rangle | E^i_{t+1}]$ to achieve the following:

$$\sum_{i=1}^{n} \mathbb{E}\left\langle \nabla f(\mu_t), -\frac{1}{\alpha^i}\widetilde{h}^i_{t+1}\right\rangle \le \sum_{i=1}^{n} \frac{\mathbb{E}[\|\nabla f(\mu_t)\|^2]}{4} + C^i_t - \mathbb{E}[\langle \nabla f(\mu_t), \nabla f_i(\mu_t)\rangle]. \tag{C.117}$$

Now we use that $\sum_{i=1}^{n} \frac{f_i(w)}{n} = f(w)$, for any vector $w \in \mathbb{R}^d$.

$$\sum_{i=1}^{n} \mathbb{E}\left\langle \nabla f(\mu_t), -\frac{1}{\alpha^i}\widetilde{h}^i_{t+1}\right\rangle \le \frac{n\mathbb{E}\left\|\nabla f(\mu_t)\right\|^2}{4} - n\mathbb{E}\left\|\nabla f(\mu_t)\right\|^2 + \sum_{i=1}^{n} C^i_t. \tag{C.118}$$

Finally, we compute:

$$\sum_{i=1}^{n} C_t^i = \sum_{i=1}^{n} \left(4L^2\eta^2K^2\sigma^2 + 20L^2\mathbb{E}\left\|w_t^i - \mu_t\right\|^2 + 16L^2\eta^2K^2\mathbb{E}\left\|\nabla f_i(\mu_t)\right\|^2 \right) \tag{C.119}$$

$$= 4nL^2\eta^2K^2\sigma^2 + 20L^2\sum_{i=1}^{n}\mathbb{E}\left\|w_t^i - \mu_t\right\|^2 + 16L^2\eta^2K^2\sum_{i=1}^{n}\mathbb{E}\left\|\nabla f_i(\mu_t)\right\|^2. \tag{C.120}$$

We can then use assumption **A**13:

$$\sum_{i=1}^{n} C_t^i \leq 4nL^2\eta^2K^2\sigma^2 + 20L^2\mathbb{E}[\Phi_t] + 16nL^2\eta^2K^2\left(G^2 + B^2\mathbb{E}\left\|\nabla f(\mu_t)\right\|^2\right). \tag{C.121}$$

In conclusion, we get the following upper bound for $\sum_{i=1}^{n}\mathbb{E}\left\langle \nabla f(\mu_t), -\frac{1}{\alpha^i}\widetilde{h}_{t+1}^i \right\rangle$:

$$20L^2\mathbb{E}[\Phi_t] + 4nL^2\eta^2K^2(\sigma^2 + 4G^2) + n\left(16L^2\eta^2K^2B^2 - \frac{3}{4}\right)\mathbb{E}\left\|\nabla f(\mu_t)\right\|^2. \tag{C.122}$$

□

C.1.3 Bound Sum of expected local gradient variance

The next lemma gives a bound on the expected update computed by clients at time step t. The result is useful, for example, in setting an upper bound on how much the average model μ_t changes between time steps. The proof follows a similar reasoning as the proof of the previous lemma.

Lemma 47. *Assume **A**12 and **A**13. Under assumptions of Lemma 42, and for any step t, we have that:*

$$\sum_{i}^{n}\mathbb{E}\left[\left\|\frac{1}{\mathbf{P}(E_{t+1}^i>0)E_{t+1}^i\wedge K}\widetilde{h}_{t+1}^i\right\|^2\right] \leq \sigma^2\sum_{i}^{n}\left(\frac{1}{K^2\mathbf{P}(E_{t+1}^i>0)} + \frac{1}{\mathbf{P}(E_{t+1}^i>0)^2}\mathbb{E}\left[\frac{\mathbb{1}(E_{t+1}^i>0)}{E_{t+1}^i\wedge K}\right]\right) + 16L^2\mathbb{E}[\Phi_t]\max_i\left(\frac{1}{\mathbf{P}(E_{t+1}^i>0)}\right) \tag{C.123}$$

$$+ 8n\max_i\left(\frac{1}{\mathbf{P}(E_{t+1}^i>0)}\right)B^2\mathbb{E}[\|\nabla f(\mu_t)\|^2] + 8n\max_i\left(\frac{1}{\mathbf{P}(E_{t+1}^i>0)}\right)G^2 \tag{C.124}$$

$$\sum_{i}^{n}\mathbb{E}\left[\left\|\frac{1}{\mathbb{E}[E_{t+1}^i\wedge K]}\widetilde{h}_{t+1}^i\right\|^2\right] \leq \sigma^2\sum_{i}^{n}\left(\frac{1}{\mathbb{E}[E_{t+1}^i\wedge K]} + \frac{\mathbb{E}[(E_{+1}^i\wedge K)^2]}{K^2\mathbb{E}[E_{t+1}^i\wedge K]}\right) + 16L^2\mathbb{E}[\Phi_t]\max_i\left(\frac{\mathbb{E}[(E_{t+1}^i\wedge K)^2]}{\mathbb{E}[E_{t+1}^i\wedge K]}\right) \tag{C.125}$$

$$+ 8n\max_i\left(\frac{\mathbb{E}[(E_{t+1}^i\wedge K)^2]}{\mathbb{E}[E_{t+1}^i\wedge K]}\right)B^2\mathbb{E}[\|\nabla f(\mu_t)\|^2] + 8n\max_i\left(\frac{\mathbb{E}[(E_{t+1}^i\wedge K)^2]}{\mathbb{E}[E_{t+1}^i\wedge K]}\right)G^2. \tag{C.126}$$

Proof. For $\alpha^i = \mathbf{P}(E_{t+1}^i>0)E_{t+1}^i\wedge K$ or $\alpha^i = \mathbb{E}[E_{t+1}^i\wedge K]$, we have:

$$\sum_{i=1}^{n}\mathbb{E}[\left\|\frac{1}{\alpha^i}\widetilde{h}_{t+1}^i\right\|^2] = \sum_{i=1}^{n}\mathrm{Var}\left(\frac{1}{\alpha^i}\sum_{q=1}^{E_{t+1}^i\wedge K}\widetilde{h}_{t+1,q}^i\right) + \sum_{i}^{n}\|\mathbb{E}[\check{h}_{t+1}^i]\|^2. \tag{C.127}$$

Recall that for any $q \in \{1,...,K\}$, the random variables E^i_{t+1} and $\widetilde{h}^i_{t+1,q}$ are independent. For clarity we reuse the notation where $\check{h}^i_{t+1} = \frac{1}{\mathbf{P}(E^i_{t+1}>0)(E^i_{t+1}\wedge K)}\widetilde{h}^i_{t+1}$ or $\check{h}^i_{t+1} = \frac{1}{\mathbb{E}[E^i_{t+1}\wedge K]}\widetilde{h}^i_{t+1}$. Therefore we can apply Lemma 39 with $S = E^i_{t+1}\wedge K$ and $Y_q = \widetilde{h}^i_{t+1,q}$:

$$\begin{cases} \mathbb{E}[\|\frac{1}{\mathbf{P}(E^i_{t+1}>0)E^i_{t+1}\wedge K}\widetilde{h}^i_{t+1}\|^2] \le \|\mathbb{E}[\check{h}^i_{t+1}]\|^2 + \frac{\|\mathbb{E}[\check{h}^i_{t+1}]\|^2}{\mathbf{P}(E^i_{t+1}>0)^2}(\mathbf{P}(E^i_{t+1}>0)(1-\mathbf{P}(E^i_{t+1}>0))) + \frac{\mathrm{Var}(\widetilde{h}^i_{t+1,q})}{\mathbf{P}(E^i_{t+1}>0)^2}\mathbb{E}[\frac{\mathbb{1}(E^i_{t+1}>0)}{E^i_{t+1}\wedge K}] \\ \mathbb{E}[\left\|\frac{1}{\mathbb{E}[E^i_{t+1}\wedge K]}\widetilde{h}^i_{t+1}\right\|^2] \le \|\mathbb{E}[\check{h}^i_{t+1}]\|^2 + \frac{\mathrm{Var}(\widetilde{h}^i_{t+1,q})}{\mathbb{E}[E^i_{t+1}\wedge K]} + \frac{\|\mathbb{E}[\widetilde{h}^i_{t+1,q}]\|^2\,\mathrm{Var}(E^i_{t+1}\wedge K)}{\mathbb{E}[E^i_{t+1}\wedge K]^2}. \end{cases} \tag{C.128}$$

We now use Corollary 43 to get:

$$\begin{cases} \mathbb{E}\left[\|\frac{1}{\mathbf{P}(E^i_{t+1}>0)E^i_{t+1}\wedge K}\widetilde{h}^i_{t+1}\|^2\right] \le \|\mathbb{E}[\check{h}^i_{t+1}]\|^2\frac{1}{\mathbf{P}(E^i_{t+1}>0)} + \frac{\sigma^2+B^i_t-\|\mathbb{E}[\widetilde{h}^i_{t+1,q}]\|^2}{\mathbf{P}(E^i_{t+1}>0)^2}\mathbb{E}\left[\frac{\mathbb{1}[E^i_{t+1}>0]}{E^i_{t+1}\wedge K}\right] \\ \mathbb{E}\left[\|\frac{1}{\mathbb{E}[E^i_{t+1}\wedge K]}\widetilde{h}^i_{t+1}\|^2\right] \le \|\mathbb{E}[\check{h}^i_{t+1}]\|^2(1+\frac{\mathrm{Var}(E^i_{t+1}\wedge K)}{\mathbb{E}\left[E^i_{t+1}\wedge K\right]^2}) + \frac{\sigma^2+B^i_t-\|\mathbb{E}[\widetilde{h}^i_{t+1,q}]\|^2}{\mathbb{E}\left[E^i_{t+1}\wedge K\right]}. \end{cases} \tag{C.129}$$

Hence we can use Lemma 38 with $S = E^i_{t+1}\wedge K$ and $Y_q = \widetilde{h}^i_{t+1,q}$ to simplify as following:

$$\begin{cases} \mathbb{E}\left[\|\frac{1}{\mathbf{P}(E^i_{t+1}>0)E^i_{t+1}\wedge K}\widetilde{h}^i_{t+1}\|^2\right] \le \|\mathbb{E}[\widetilde{h}^i_{t+1,q}]\|^2\frac{1}{\mathbf{P}(E^i_{t+1}>0)} + \frac{\sigma^2+B^i_t-\|\mathbb{E}[\widetilde{h}^i_{t+1,q}]\|^2}{\mathbf{P}(E^i_{t+1}>0)^2}\mathbb{E}\left[\frac{\mathbb{1}[E^i_{t+1}>0]}{E^i_{t+1}\wedge K}\right] \\ \mathbb{E}\left[\|\frac{1}{\mathbb{E}[E^i_{t+1}\wedge K]}\widetilde{h}^i_{t+1}\|^2\right] \le \|\mathbb{E}[\widetilde{h}^i_{t+1,q}]\|^2(1+\frac{\mathrm{Var}(E^i_{t+1}\wedge K)}{\mathbb{E}\left[E^i_{t+1}\wedge K\right]^2}) + \frac{\sigma^2+B^i_t-\|\mathbb{E}[\widetilde{h}^i_{t+1,q}]\|^2}{\mathbb{E}\left[E^i_{t+1}\wedge K\right]}. \end{cases} \tag{C.130}$$

We refer the reader to the filtrations $\left(\mathcal{F}_{(t,q)}\right)_{(t,q)\in I}$ defined in (C.1). Now we can express the expected value of $\widetilde{h}^i_{t+1,q}$ (as μ following the notations from Lemma 39), and upper bound its square norm by Lemma 42 :

$$\begin{aligned} \|\mu\|^2 &= \|\mathbb{E}[\mathbb{E}\left[\widetilde{h}^i_{t+1,q}\middle|\mathcal{F}_{(t,q-1)}\right]]\|^2 && \text{(C.131)} \\ &\le \mathbb{E}[\|\mathbb{E}\left[\widetilde{h}^i_{t+1,q}\middle|\mathcal{F}_{(t,q-1)}\right]\|^2] && \text{(C.132)} \\ &\le \mathbb{E}[\|h^i_{t+1,q}\|^2] && \text{(C.133)} \\ &\le B^i_t && \text{(C.134)} \end{aligned}$$

We can insert this bound in the above inequations:

$$\begin{cases} \mathbb{E}\left[\|\frac{1}{\mathbf{P}(E^i_{t+1}>0)E^i_{t+1}\wedge K}\widetilde{h}^i_{t+1}\|^2\right] \le B^i_t\frac{\mathbf{P}(E^i_{t+1}>0)-\mathbb{E}\left[\frac{\mathbb{1}[E^i_{t+1}>0]}{E^i_{t+1}\wedge K}\right]}{\mathbf{P}(E^i_{t+1}>0)^2} + \frac{\sigma^2+B^i_t}{\mathbf{P}(E^i_{t+1}>0)^2}\mathbb{E}\left[\frac{\mathbb{1}[E^i_{t+1}>0]}{E^i_{t+1}\wedge K}\right] \\ \mathbb{E}\left[\|\frac{1}{\mathbb{E}[E^i_{t+1}\wedge K]}\widetilde{h}^i_{t+1}\|^2\right] \le B^i_t\frac{\mathbb{E}[(E^i_{t+1}\wedge K)^2]-\mathbb{E}[E^i_{t+1}\wedge K]}{\mathbb{E}\left[E^i_{t+1}\wedge K\right]^2} + \frac{\sigma^2+B^i_t}{\mathbb{E}\left[E^i_{t+1}\wedge K\right]}. \end{cases} \tag{C.135}$$

This simplifies as:

$$\begin{cases} \mathbb{E}\left[\|\frac{1}{\mathbf{P}(E^i_{t+1}>0)E^i_{t+1}\wedge K}\widetilde{h}^i_{t+1}\|^2\right] \le \frac{B^i_t}{\mathbf{P}(E^i_{t+1}>0)} + \frac{\sigma^2}{\mathbf{P}(E^i_{t+1}>0)^2}\mathbb{E}\left[\frac{\mathbb{1}[E^i_{t+1}>0]}{E^i_{t+1}\wedge K}\right] \\ \mathbb{E}\left[\|\frac{1}{\mathbb{E}[E^i_{t+1}\wedge K]}\widetilde{h}^i_{t+1}\|^2\right] \le B^i_t\frac{\mathbb{E}[(E^i_{t+1}\wedge K)^2]}{\mathbb{E}\left[E^i_{t+1}\wedge K\right]^2} + \frac{\sigma^2}{\mathbb{E}\left[E^i_{t+1}\wedge K\right]}. \end{cases} \tag{C.136}$$

Expanding B_t^i and summing from i to n, we get:

$$\sum_i^n \mathbb{E}\left[\|\frac{1}{\mathbf{P}(E_{t+1}^i>0)E_{t+1}^i \wedge K}\widetilde{h}_{t+1}^i\|^2\right] \leq \sigma^2 \sum_i^n (\frac{1}{K^2\mathbf{P}(E_{t+1}^i>0)} + \frac{1}{\mathbf{P}(E_{t+1}^i>0)^2}\mathbb{E}[\frac{\mathbb{1}(E_{t+1}^i>0)}{E_{t+1}^i \wedge K}]) + 16L^2\mathbb{E}[\Phi_t]\max_i(\frac{1}{\mathbf{P}(E_{t+1}^i>0)}) \tag{C.137}$$

$$+ 8\sum_i^n \max_j(\frac{1}{\mathbf{P}(E_{t+1}^j>0)})\mathbb{E}[\|\nabla f_i(\mu_t)\|^2] \tag{C.138}$$

$$\sum_i^n \mathbb{E}\left[\|\frac{1}{\mathbb{E}[E_{t+1}^i \wedge K]}\widetilde{h}_{t+1}^i\|^2\right] \leq \sigma^2 \sum_i^n (\frac{1}{\mathbb{E}[E_{t+1}^i \wedge K]} + \frac{\mathbb{E}[(E_{+1}^i \wedge K)^2]}{K^2\mathbb{E}[E_{t+1}^i \wedge K]}) + 16L^2\mathbb{E}[\Phi_t]\max_i(\frac{\mathbb{E}[(E_{t+1}^i \wedge K)^2]}{\mathbb{E}[E_{t+1}^i \wedge K]}) \tag{C.139}$$

$$+ 8\sum_i^n \max_j(\frac{\mathbb{E}[(E_{t+1}^j \wedge K)^2]}{\mathbb{E}[E_{t+1}^j \wedge K]})\mathbb{E}[\|\nabla f(\mu_t)\|^2]. \tag{C.140}$$

Remark 48. *Here we loose a lot: we have upper bounded the term* $\sum_i^n \frac{1}{\mathbf{P}(E_{t+1}^i>0)}\|w_t^i-\mu_t\|^2 \leq \sum_i^n \max_i(\frac{1}{\mathbf{P}(E_{t+1}^i>0)})\|w_t^i-\mu_t\|^2$. *But still, our bounds stay better than Zakerinia et al. 2022's ones.*

In order to complete the proof, we combine assumption **A**13 together with the above inequalities

$$\sum_i^n \mathbb{E}\left[\|\frac{1}{\mathbf{P}(E_{t+1}^i>0)E_{t+1}^i \wedge K}\widetilde{h}_{t+1}^i\|^2\right] \leq \sigma^2 \sum_i^n (\frac{1}{K^2\mathbf{P}(E_{t+1}^i>0)} + \frac{1}{\mathbf{P}(E_{t+1}^i>0)^2}\mathbb{E}[\frac{\mathbb{1}(E_{t+1}^i>0)}{E_{t+1}^i \wedge K}]) + 16L^2\mathbb{E}[\Phi_t]\max_i(\frac{1}{\mathbf{P}(E_{t+1}^i>0)}) \tag{C.141}$$

$$+ \max_i(\frac{1}{\mathbf{P}(E_{t+1}^i>0)})\left(8nB^2\mathbb{E}[\|\nabla f(\mu_t)\|^2] + 8nG^2\right) \tag{C.142}$$

$$\sum_i^n \mathbb{E}\left[\|\frac{1}{\mathbb{E}[E_{t+1}^i \wedge K]}\widetilde{h}_{t+1}^i\|^2\right] \leq \sigma^2 \sum_i^n (\frac{1}{\mathbb{E}[E_{t+1}^i \wedge K]} + \frac{\mathbb{E}[(E_{+1}^i \wedge K)^2]}{K^2\mathbb{E}[E_{t+1}^i \wedge K]}) + 16L^2\mathbb{E}[\Phi_t]\max_i(\frac{\mathbb{E}[(E_{t+1}^i \wedge K)^2]}{\mathbb{E}[E_{t+1}^i \wedge K]}) \tag{C.143}$$

$$+ 8n\max_i(\frac{\mathbb{E}[(E_{t+1}^i \wedge K)^2]}{\mathbb{E}[E_{t+1}^i \wedge K]})B^2\mathbb{E}[\|\nabla f(\mu_t)\|^2] + 8n\max_i(\frac{\mathbb{E}[(E_{t+1}^i \wedge K)^2]}{\mathbb{E}[E_{t+1}^i \wedge K]})B^2. \tag{C.144}$$

□

C.1.4 Bound the sum (over time) of expected potential

Lemma 49. *Assume that* $\eta \leq \frac{1}{20sL\max_i(\frac{1}{\mathbf{P}(E^i_{t+1}>0)})}, \frac{1}{20sL\max_i(\frac{\mathbb{E}[(E^i_{t+1}\wedge K)^2]}{\mathbb{E}[E^i_{t+1}\wedge K]})}$. *Under the assumptions of Lemmas 9 and 47, and for any time step t we have:*

$$\mathbb{E}[\Phi_{t+1}] \leq \left(1-\frac{1}{5n}\right)\mathbb{E}[\Phi_t] + 3s^2\eta^2\left(\frac{1}{n}\sum_i^n\left(\frac{1}{K^2\mathbf{P}(E^i_{t+1}>0)} + \frac{1}{\mathbf{P}(E^i_{t+1}>0)^2}\mathbb{E}[\frac{\mathbb{1}(E^i_{t+1}>0)}{E^i_{t+1}\wedge K}]\right) + 8\max_i\left(\frac{1}{\mathbf{P}(E^i_{t+1}>0)}\right)G^2\right) \tag{C.145}$$

$$+ 24B^2s^2\eta^2\max_i\left(\frac{1}{\mathbf{P}(E^i_{t+1}>0)}\right)\mathbb{E}\left\|\nabla f(\mu_t)\right\|^2 \tag{C.146}$$

$$\tag{C.147}$$

$$\mathbb{E}[\Phi_{t+1}] \leq \left(1-\frac{1}{5n}\right)\mathbb{E}[\Phi_t] + 3s^2\eta^2\left(\frac{1}{n}\sum_i^n\left(\frac{1}{\mathbb{E}[E^i_{t+1}\wedge K]} + \frac{\mathbb{E}[(E^i_{+1}\wedge K)^2]}{K^2\mathbb{E}[E^i_{t+1}\wedge K]}\right) + 8\max_i\left(\frac{\mathbb{E}[(E^i_{t+1}\wedge K)^2]}{\mathbb{E}[E^i_{t+1}\wedge K]}\right)G^2\right) \tag{C.148}$$

$$+ 24B^2s^2\eta^2\max_i\left(\frac{\mathbb{E}[(E^i_{t+1}\wedge K)^2]}{\mathbb{E}[E^i_{t+1}\wedge K]}\right)\mathbb{E}\left\|\nabla f(\mu_t)\right\|^2 \tag{C.149}$$

$$\tag{C.150}$$

Proof. We first use Lemma 9:

$$\mathbb{E}[\Phi_{t+1}] \leq (1-\kappa)\mathbb{E}[\Phi_t] + 3\frac{s^2}{n}\eta^2\sum_{i=1}^n\mathbb{E}[\frac{1}{{\alpha^i}^2}\left\|\widetilde{h}^i_{t+1}\right\|^2]. \tag{C.151}$$

, with $\alpha^i = \mathbf{P}(E^i_{t+1}>0)(E^i_{t+1}\wedge K)$ or $\alpha^i = \mathbb{E}[E^i_{t+1}\wedge K]$. Now we expand the quantity above using the inequality in Lemma 47:

$$\mathbb{E}[\Phi_{t+1}] \leq (1-\kappa)\mathbb{E}[\Phi_t] + 3\frac{s^2}{n}\eta^2\left(\sigma^2\sum_i^n a^i + b(16L^2\mathbb{E}[\Phi_t] + 8nB^2\mathbb{E}[\|\nabla f(\mu_t)\|^2]) + 8nG^2\right) \tag{C.152}$$

$$\leq \left(1-\frac{1}{n}\left(\frac{s(n-s)}{2(n+1)(s+1)}\right) + 48\frac{s^2}{n}L^2b\eta^2\right)\mathbb{E}[\Phi_t] + 3\frac{s^2}{n}(\sigma^2\sum_i^n a^i + 8nbG^2)\eta^2 + 24B^2s^2b\eta^2\mathbb{E}\left\|\nabla f(\mu_t)\right\|^2. \tag{C.153}$$

With

$$\begin{cases} a^i, b = \frac{1}{K^2\mathbf{P}(E^i_{t+1}>0)} + \frac{1}{\mathbf{P}(E^i_{t+1}>0)^2}\mathbb{E}[\frac{\mathbb{1}(E^i_{t+1}>0)}{E^i_{t+1}\wedge K}], \max_i(\frac{1}{\mathbf{P}(E^i_{t+1}>0)}) \\ a^i, b = \frac{1}{\mathbb{E}[E^i_{t+1}\wedge K]} + \frac{\mathbb{E}[(E^i_{+1}\wedge K)^2]}{K^2\mathbb{E}[E^i_{t+1}\wedge K]}, \max_i(\frac{\mathbb{E}[(E^i_{t+1}\wedge K)^2]}{\mathbb{E}[E^i_{t+1}\wedge K]}). \end{cases} \tag{C.154}$$

To complete, we use $\eta \leq \frac{1}{20sLb}$:

$$\mathbb{E}[\Phi_{t+1}] \leq \left(1-\frac{1}{5n}\right)\mathbb{E}[\Phi_t] + 3s^2\eta^2\left(\sigma^2\frac{1}{n}\sum_i^n a^i + 8bG^2\right) + 24B^2s^2\eta^2b\mathbb{E}\left\|\nabla f(\mu_t)\right\|^2. \tag{C.155}$$

□

In the next lemma, we bound the cumulative sum of potential functions.

Lemma 50. *Let T be a positive integer. Under the assumptions of Lemma 49, the following inequality holds:*

$$\sum_{t=0}^{T}\mathbb{E}\left[\Phi_t\right] \leq 120nB^2s^2\max_i\left(\frac{1}{\mathbf{P}(E_{t+1}^i>0)}\right)\eta^2\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla f\left(\mu_t\right)\right\|^2 \tag{C.156}$$

$$+15Ts^2\eta^2\left(\sigma^2\sum_i^n\left(\frac{1}{K^2\mathbf{P}(E_{t+1}^i>0)}+\frac{1}{\mathbf{P}(E_{t+1}^i>0)^2}\mathbb{E}\left[\frac{\mathbb{1}(E_{t+1}^i>0)}{E_{t+1}^i\wedge K}\right]\right)+8n\max_i\left(\frac{1}{\mathbf{P}(E_{t+1}^i>0)}\right)G^2\right). \tag{C.157}$$

$$\sum_{t=0}^{T}\mathbb{E}\left[\Phi_t\right] \leq 120nB^2s^2\max_i\left(\frac{\mathbb{E}[(E_{t+1}^i\wedge K)^2]}{\mathbb{E}[E_{t+1}^i\wedge K]}\right)\eta^2\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla f\left(\mu_t\right)\right\|^2 \tag{C.158}$$

$$+15Ts^2\eta^2\left(\sigma^2\sum_i^n\left(\frac{1}{\mathbb{E}[E_{t+1}^i\wedge K]}+\frac{\mathbb{E}[(E_{+1}^i\wedge K)^2]}{K^2\mathbb{E}[E_{t+1}^i\wedge K]}\right)+8n\max_i\left(\frac{\mathbb{E}[(E_{t+1}^i\wedge K)^2]}{\mathbb{E}[E_{t+1}^i\wedge K]}\right)G^2\right). \tag{C.159}$$

Proof. From Lemma 49, we get that there exist α,β not depending on t such that:

$$\mathbb{E}\left[\Phi_{t+1}\right] \leq \left(1-\frac{1}{5n}\right)\mathbb{E}\left[\Phi_t\right]+\alpha\mathbb{E}\left\|\nabla f\left(\mu_t\right)\right\|^2+\beta. \tag{C.160}$$

Therefore:

$$\sum_{t=0}^{T-1}\mathbb{E}\left[\Phi_{t+1}\right] \leq \sum_{t=0}^{T-1}\left(\left(1-\frac{1}{5n}\right)\mathbb{E}\left[\Phi_t\right]+\alpha\mathbb{E}\left\|\nabla f\left(\mu_t\right)\right\|^2+\beta\right) \tag{C.161}$$

$$\leq\left(1-\frac{1}{5n}\right)\sum_{t=0}^{T-1}\mathbb{E}\left[\Phi_t\right]+T\beta+\alpha\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla f\left(\mu_t\right)\right\|^2. \tag{C.162}$$

Rearranging the terms in the sum we obtain the following:

$$\left(1-\frac{1}{5n}\right)\mathbb{E}\left[\Phi_0\right]+\frac{1}{5n}\sum_{t=1}^{T-1}\mathbb{E}\left[\Phi_t\right]+\mathbb{E}\left[\Phi_T\right] \leq T\beta+\alpha\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla f\left(\mu_t\right)\right\|^2. \tag{C.163}$$

From this inequality, we get:

$$\sum_{t=0}^{T}\mathbb{E}\left[\Phi_t\right] \leq 5n\left(T\beta+\alpha\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla f\left(\mu_t\right)\right\|^2\right). \tag{C.164}$$

Expanding on the values of α,β obtained by Lemma 49, we get the desired result:

$$\sum_{t=0}^{T}\mathbb{E}\left[\Phi_t\right] \leq 5nT\frac{3s^2}{n}\eta^2\left(\sigma^2\sum_i^n a^i+8nbG^2\right)+120nB^2s^2b\eta^2\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla f\left(\mu_t\right)\right\|^2. \tag{C.165}$$

□

C.1.5 Bound the change in the average model

The next lemma upper bounds the expected change in the average model μ_t.

Lemma 51. *For any time step $t \geq 0$,*

$$\mathbb{E}\left\|\mu_{t+1} - \mu_t\right\|^2 \leq \frac{s^2\eta^2}{n(n+1)^2} \sum_{i=1}^{n} \mathbb{E}\left\|\check{h}_{t+1}^i\right\|^2. \tag{C.166}$$

Proof. Recall that

$$\mu_{t+1} - \mu_t = -\frac{\eta}{n+1} \sum_{i \in \mathcal{S}_{t+1}} \check{h}_{t+1}^i. \tag{C.167}$$

Therefore we may compute an upper bound:

$$\|\mu_{t+1} - \mu_t\|^2 = \frac{\eta^2}{(n+1)^2} \left\| \sum_{i \in \mathcal{S}_{t+1}} \check{h}_{t+1}^i \right\|^2 \tag{C.168}$$

$$\leq \frac{s\eta^2}{(n+1)^2} \sum_{i \in \mathcal{S}_{t+1}} \left\|\check{h}_{t+1}^i\right\|^2 \tag{C.169}$$

We may then apply Lemma 37 to get the desired result:

$$\mathbb{E}\left\|\mu_{t+1} - \mu_t\right\|^2 \leq \frac{s^2\eta^2}{n(n+1)^2} \sum_{i=1}^{n} \mathbb{E}\left\|\check{h}_{t+1}^i\right\|^2. \tag{C.170}$$

□

We now give another upper bound on how the average model μ_t changes at time step t.

Lemma 52. *Under the assumptions of Lemmas 47 and 51, and for any step t:*

$$\mathbb{E}\|\mu_{t+1} - \mu_t\|^2 \leq \frac{s^2\eta^2\sigma^2}{n(n+1)^2} \sum_{i}^{n} \left(\frac{1}{K^2 \mathbf{P}(E_{t+1}^i > 0)} + \frac{1}{\mathbf{P}(E_{t+1}^i > 0)^2} \mathbb{E}\left[\frac{\mathbb{1}(E_{t+1}^i > 0)}{E_{t+1}^i \wedge K}\right]\right) + \frac{16L^2s^2\eta^2}{n(n+1)^2} \mathbb{E}[\Phi_t] \max_i \left(\frac{1}{\mathbf{P}(E_{t+1}^i > 0)}\right) \tag{C.171}$$

$$+ \frac{8s^2B^2\eta^2}{(n+1)^2} \max_i \left(\frac{1}{\mathbf{P}(E_{t+1}^i > 0)}\right) \mathbb{E}[\|\nabla f(\mu_t)\|^2] + \frac{8s^2G^2\eta^2}{(n+1)^2} \max_i \left(\frac{1}{\mathbf{P}(E_{t+1}^i > 0)}\right) \tag{C.172}$$

$$\mathbb{E}\|\mu_{t+1} - \mu_t\|^2 \leq \frac{s^2\eta^2\sigma^2}{n(n+1)^2} \sum_{i}^{n} \left(\frac{1}{\mathbb{E}[E_{t+1}^i \wedge K]} + \frac{\mathbb{E}[(E_{+1}^i \wedge K)^2]}{K^2 \mathbb{E}[E_{t+1}^i \wedge K]}\right) + \frac{16L^2s^2\eta^2}{n(n+1)^2} \mathbb{E}[\Phi_t] \max_i \left(\frac{\mathbb{E}[(E_{t+1}^i \wedge K)^2]}{\mathbb{E}[E_{t+1}^i \wedge K]}\right) \tag{C.173}$$

$$+ \frac{8s^2B^2\eta^2}{(n+1)^2} \max_i \left(\frac{\mathbb{E}[(E_{t+1}^i \wedge K)^2]}{\mathbb{E}[E_{t+1}^i \wedge K]}\right) \mathbb{E}[\|\nabla f(\mu_t)\|^2] + \frac{8s^2G^2\eta^2}{(n+1)^2} \max_i \left(\frac{\mathbb{E}[(E_{t+1}^i \wedge K)^2]}{\mathbb{E}[E_{t+1}^i \wedge K]}\right). \tag{C.174}$$

Proof. For this proof, we will combine the inequality obtained from Lemma 51 to the one from

Lemma 47. This will be enough to obtain the desired result.

$$\mathbb{E}\|\mu_{t+1}-\mu_t\|^2 \leq \sum_{i=1}^{n} \frac{s^2\eta^2}{n(n+1)^2}\mathbb{E}\|\check{h}_{t+1}^i\|^2. \tag{C.175}$$

Simplifying the above quantity, we get the desired inequality:

$$\mathbb{E}\|\mu_{t+1}-\mu_t\|^2 \leq \frac{s^2\eta^2\sigma^2}{n(n+1)^2}\sum_i^n\Big(\frac{1}{K^2\mathbf{P}(E_{t+1}^i>0)}+\frac{1}{\mathbf{P}(E_{t+1}^i>0)^2}\mathbb{E}\Big[\frac{\mathbb{1}(E_{t+1}^i>0)}{E_{t+1}^i\wedge K}\Big]\Big)+\frac{16L^2s^2\eta^2}{n(n+1)^2}\mathbb{E}[\Phi_t]\max_i\Big(\frac{1}{\mathbf{P}(E_{t+1}^i>0)}\Big) \tag{C.176}$$

$$+\frac{8s^2B^2\eta^2}{(n+1)^2}\max_i\Big(\frac{1}{\mathbf{P}(E_{t+1}^i>0)}\Big)\mathbb{E}[\|\nabla f(\mu_t)\|^2]+\frac{8s^2G^2\eta^2}{(n+1)^2}\max_i\Big(\frac{1}{\mathbf{P}(E_{t+1}^i>0)}\Big)G^2 \tag{C.177}$$

$$\mathbb{E}\|\mu_{t+1}-\mu_t\|^2 \leq \frac{s^2\eta^2\sigma^2}{n(n+1)^2}\sum_i^n\Big(\frac{1}{\mathbb{E}[E_{t+1}^i\wedge K]}+\frac{\mathbb{E}[(E_{+1}^i\wedge K)^2]}{K^2\mathbb{E}[E_{t+1}^i\wedge K]}\Big)+\frac{16L^2s^2\eta^2}{n(n+1)^2}\mathbb{E}[\Phi_t]\max_i\Big(\frac{\mathbb{E}[(E_{t+1}^i\wedge K)^2]}{\mathbb{E}[E_{t+1}^i\wedge K]}\Big) \tag{C.178}$$

$$+\frac{8s^2B^2\eta^2}{(n+1)^2}\max_i\Big(\frac{\mathbb{E}[(E_{t+1}^i\wedge K)^2]}{\mathbb{E}[E_{t+1}^i\wedge K]}\Big)\mathbb{E}[\|\nabla f(\mu_t)\|^2]+\frac{8s^2G^2\eta^2}{(n+1)^2}\max_i\Big(\frac{\mathbb{E}[(E_{t+1}^i\wedge K)^2]}{\mathbb{E}[E_{t+1}^i\wedge K]}\Big)G^2. \tag{C.179}$$

□

C.1.6 Convergence result

In this section, we use the lemmas proved so far to demonstrate Theorem 10. Following the proof, we establish the learning rate η that results in the best overall rate of convergence.

Proof. Using L-smoothness, we have:

$$f(\mu_{t+1}) \leq f(\mu_t)+\langle\nabla f(\mu_t),\mu_{t+1}-\mu_t\rangle+\frac{L}{2}\left\|\mu_{t+1}-\mu_t\right\|^2. \tag{C.180}$$

First we look at the term $\langle\nabla f(\mu_t),\mu_{t+1}-\mu_t\rangle$. Recall that:

$$\mu_{t+1}-\mu_t = -\frac{\eta}{n+1}\sum_{i\in\mathcal{S}_{t+1}}\check{h}_{t+1}^i \tag{C.181}$$

by Lemma 37, we have:

$$\mathbb{E}_t[\mu_{t+1}-\mu_t] = -\frac{s\eta}{n(n+1)}\sum_{i=1}^{n}\check{h}_{t+1}^i \tag{C.182}$$

and subsequently

$$\mathbb{E}_t\langle\nabla f(\mu_t),\mu_{t+1}-\mu_t\rangle = \sum_{i=1}^{n}\frac{s\eta}{n(n+1)}\mathbb{E}_t\left\langle\nabla f(\mu_t),-\check{h}_{t+1}^i\right\rangle.$$

Hence, we can rewrite (C.180) as:

$$\mathbb{E}_t\left[f\left(\mu_{t+1}\right)\right] \leq f\left(\mu_t\right) + \sum_{i=1}^{n} \frac{s\eta}{n(n+1)} \mathbb{E}_t \left\langle \nabla f\left(\mu_t\right), -\check{h}_{t+1}^i \right\rangle + \frac{L}{2} \mathbb{E}_t \left\| \mu_{t+1} - \mu_t \right\|^2. \tag{C.183}$$

Next, we remove the conditioning with the tower law of expectation:

$$\mathbb{E}\left[f\left(\mu_{t+1}\right)\right] \leq \mathbb{E}\left[f\left(\mu_t\right)\right] + \sum_{i=1}^{n} \frac{s\eta}{n(n+1)} \mathbb{E} \left\langle \nabla f\left(\mu_t\right), -\check{h}_{t+1}^i \right\rangle + \frac{L}{2} \mathbb{E} \left\| \mu_{t+1} - \mu_t \right\|^2. \tag{C.184}$$

We now define some notation to simplify the computations. By Lemma 46, there exist a_1, a_2, a_3 not depending on t such that

$$\sum_{i=1}^{n} \mathbb{E} \left\langle \nabla f\left(\mu_t\right), -\check{h}_{t+1}^i \right\rangle \leq a_1 \mathbb{E}\left[\Phi_t\right] + a_2 \mathbb{E} \left\| \nabla f\left(\mu_t\right) \right\|^2 + a_3. \tag{C.185}$$

Similarly, by Lemma 52, there exist b_1, b_2, b_3 not depending on t such that:

$$\mathbb{E} \left\| \mu_{t+1} - \mu_t \right\|^2 \leq b_1 \mathbb{E}\left[\Phi_t\right] + b_2 \mathbb{E} \left\| \nabla f\left(\mu_t\right) \right\|^2 + b_3. \tag{C.186}$$

Defining $c_i = a_i \frac{s\eta}{n(n+1)} + b_i \frac{L}{2}$, we have

$$\mathbb{E}\left[f\left(\mu_{t+1}\right)\right] - \mathbb{E}\left[f\left(\mu_t\right)\right] \leq c_1 \mathbb{E}\left[\Phi_t\right] + c_2 \mathbb{E} \left\| \nabla f\left(\mu_t\right) \right\|^2 + c_3. \tag{C.187}$$

Summing the above inequality for $t = 0, 1, \ldots, T-1$ we get that:

$$\mathbb{E}\left[f\left(\mu_T\right)\right] - f(\mu_0) \leq c_1 \sum_{t=0}^{T-1} \mathbb{E}\left[\Phi_t\right] + c_2 \sum_{t=0}^{T-1} \mathbb{E} \left\| \nabla f\left(\mu_t\right) \right\|^2 + c_3 T. \tag{C.188}$$

By Lemma 50 there exist d_1, d_2 independent of T such that:

$$\sum_{t=0}^{T} \mathbb{E}\left[\Phi_t\right] \leq d_1 \sum_{t=0}^{T-1} \mathbb{E} \left\| \nabla f\left(\mu_t\right) \right\|^2 + T d_2. \tag{C.189}$$

We then get:

$$\mathbb{E}\left[f\left(\mu_T\right)\right] - f(\mu_0) \leq (c_1 d_1 + c_2) \sum_{t=0}^{T-1} \mathbb{E} \left\| \nabla f\left(\mu_t\right) \right\|^2 + T(c_1 d_2 + c_3). \tag{C.190}$$

We now assume that $c_1 d_1 + c_2 < 0$. Later in the proof, we will show that this is true for small enough η. Using the fact that $f(\mu_T) \geq f_*$ and rearranging the terms, we get:

$$\frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E} \left\| \nabla f\left(\mu_t\right) \right\|^2 \leq \frac{f(\mu_0) - f_*}{T(-c_1 d_1 - c_2)} + \frac{c_1 d_2 + c_3}{-c_1 d_1 - c_2}. \tag{C.191}$$

Of course, now we expand each of these terms. Refer to Lemma 46, Lemma 52, and Lemma 50

for the specific values of the defined quantities a_i, b_i, and d_i. We have:

$$c_1 = \frac{s\eta}{n(n+1)}a_1 + \frac{L}{2}b_1 \tag{C.192}$$

$$= 20L^2\frac{s\eta}{n(n+1)} + \frac{16s^2\eta^2L^2b}{n(n+1)^2}\frac{L}{2} \tag{C.193}$$

$$= \frac{4L^2s\eta}{n(n+1)^2}(5(n+1) + 2s\eta Lb). \tag{C.194}$$

We recall here the definition from Lemma 49

$$\begin{cases} a^i, b = \frac{1}{K^2\mathbf{P}(E^i_{t+1}>0)} + \frac{1}{\mathbf{P}(E^i_{t+1}>0)^2}\mathbb{E}[\frac{\mathbb{1}(E^i_{t+1}>0)}{E^i_{t+1}\wedge K}], \max_i(\frac{1}{\mathbf{P}(E^i_{t+1}>0)}) \\ a^i, b = \frac{1}{\mathbb{E}[E^i_{t+1}\wedge K]} + \frac{\mathbb{E}[(E^i_{+1}\wedge K)^2]}{K^2\mathbb{E}[E^i_{t+1}\wedge K]}, \max_i(\frac{\mathbb{E}[(E^i_{t+1}\wedge K)^2]}{\mathbb{E}[E^i_{t+1}\wedge K]}). \end{cases} \tag{C.195}$$

By using $\eta \leq \frac{n+1}{2sLb}$ we get:

$$c_1 \leq \frac{24L^2s\eta}{n(n+1)}. \tag{C.196}$$

Therefore:

$$c_1d_1 \leq \frac{24L^2s\eta}{n(n+1)}120nB^2s^2b\eta^2 \tag{C.197}$$

$$\leq \frac{2880L^2s^3\eta^3B^2b}{n+1}. \tag{C.198}$$

Moreover:

$$c_2 = \frac{s\eta}{n(n+1)}a_2 + \frac{L}{2}b_2 \tag{C.199}$$

$$= \frac{s\eta}{n(n+1)}\left(n\left(16L^2\eta^2K^2B^2 - \frac{3}{4}\right)\right) + \frac{L}{2}\left(\frac{8s^2\eta^2bB^2}{(n+1)^2}\right) \tag{C.200}$$

$$= \frac{s\eta}{(n+1)}\left(16L^2\eta^2K^2B^2 - \frac{3}{4}\right) + \frac{4Ls^2\eta^2bB^2}{(n+1)^2}. \tag{C.201}$$

By using $\eta \leq \frac{1}{20B^2bKLs}$ we get:

$$c_1d_1 \leq \frac{2880}{8000(n+1)LB^4K^3b^2} \tag{C.202}$$

$$c_2 \leq \frac{s\eta}{n+1}\left(\frac{16}{400s^2b^2B^2} - \frac{3}{4}\right) + \frac{4}{400(n+1)^2LbB^2K^2} \tag{C.203}$$

$$\leq \frac{-s\eta}{2(n+1)} - c_1d_1. \tag{C.204}$$

Therefore we conclude that $-c_2 - c_1 d_1 \geq \frac{s\eta}{2(n+1)}$, which is greater than 0. Now we compute:

$$c_1 d_2 \leq \frac{24L^2 s\eta}{n(n+1)}\left(15s^2\eta^2\left(\sigma^2\sum_i^n a^i + 8nbG^2\right)\right) \tag{C.205}$$

$$\leq \frac{360L^2 s^3\eta^3}{(n+1)}\left(\sigma^2\frac{1}{n}\sum_i^n a^i + 8bG^2\right). \tag{C.206}$$

And additionally:

$$c_3 = \frac{s\eta}{n(n+1)}a_3 + \frac{L}{2}b_3 \tag{C.207}$$

$$= \frac{s\eta}{n(n+1)}\left(4nL^2\eta^2K^2(\sigma^2+4G^2)\right) + \frac{L}{2}\left(\frac{s^2\eta^2\sigma^2}{n(n+1)^2}\sum_i^n a^i + \frac{8s^2G^2\eta^2}{(n+1)^2}b\right). \tag{C.208}$$

Thus

$$c_3 + c_1 d_2 \leq \left(\frac{4L^2\eta^2K^2 s\eta}{(n+1)} + \frac{Ls^2\eta^2}{2n(n+1)^2}\sum_i^n a^i + \frac{360L^2s^3\eta^3}{n(n+1)}\sum_i^n a^i\right)\sigma^2 \tag{C.209}$$

$$+\left(\frac{16L^2\eta^2K^2 s\eta}{(n+1)} + \frac{4Ls^2\eta^2}{(n+1)^2}b + \frac{2880bL^2s^3\eta^3}{(n+1)}\right)G^2. \tag{C.210}$$

And therefore:

$$\frac{c_3 + c_1 d_2}{-c_1 d_1 - c_2} \leq \left(8L^2\eta^2K^2 + \frac{Ls\eta}{n(n+1)}\sum_i^n a^i + \frac{720L^2s^2\eta^2}{n}\sum_i^n a^i\right)\sigma^2 \tag{C.211}$$

$$+\left(32L^2\eta^2K^2 + \frac{8Ls\eta}{(n+1)}b + 5600bL^2s^2\eta^2\right)G^2. \tag{C.212}$$

Finally:

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla f(\mu_t)\right\|^2 \leq 2(n+1)\frac{f(\mu_0) - f_*}{Ts\eta} \tag{C.213}$$

$$+\left(8L^2\eta^2K^2 + \frac{Ls\eta}{n(n+1)}\sum_i^n a^i + \frac{720L^2s^2\eta^2}{n}\sum_i^n a^i\right)\sigma^2 \tag{C.214}$$

$$+\left(32L^2\eta^2K^2 + \frac{8Ls\eta}{(n+1)}b + 5600bL^2s^2\eta^2\right)G^2. \tag{C.215}$$

□

In particular for stochastic reweighting:

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla f(\mu_t)\right\|^2 \leq 2(n+1)\frac{f(\mu_0)-f_*}{Ts\eta} \tag{C.216}$$

$$+\left(8L^2\eta^2K^2+\frac{Ls\eta}{n(n+1)}\sum_i^n\left(\frac{1}{K^2\mathbf{P}(E_{t+1}^i>0)}+\frac{1}{\mathbf{P}(E_{t+1}^i>0)^2}\mathbb{E}\left[\frac{\mathbb{1}(E_{t+1}^i>0)}{E_{t+1}^i\wedge K}\right]\right)\right)\sigma^2 \tag{C.217}$$

$$+\left(\frac{720L^2s^2\eta^2}{n}\sum_i^n\left(\frac{1}{K^2\mathbf{P}(E_{t+1}^i>0)}+\frac{1}{\mathbf{P}(E_{t+1}^i>0)^2}\mathbb{E}\left[\frac{\mathbb{1}(E_{t+1}^i>0)}{E_{t+1}^i\wedge K}\right]\right)\right)\sigma^2 \tag{C.218}$$

$$+\left(32L^2\eta^2K^2+\frac{8Ls\eta}{(n+1)}\max_i\left(\frac{1}{\mathbf{P}(E_{t+1}^i>0)}\right)+5600L^2s^2\eta^2\max_i\left(\frac{1}{\mathbf{P}(E_{t+1}^i>0)}\right)\right)G^2. \tag{C.219}$$

And particular for expectation reweighting:

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla f(\mu_t)\right\|^2 \leq 2(n+1)\frac{f(\mu_0)-f_*}{Ts\eta} \tag{C.220}$$

$$+\left(8L^2\eta^2K^2+\frac{Ls\eta}{n(n+1)}\sum_i^n\left(\frac{1}{\mathbb{E}[E_{t+1}^i\wedge K]}+\frac{\mathbb{E}[(E_{t+1}^i\wedge K)^2]}{K^2\mathbb{E}[E_{t+1}^i\wedge K]}\right)+\frac{720L^2s^2\eta^2}{n}\sum_i^n\left(\frac{1}{\mathbb{E}[E_{t+1}^i\wedge K]}+\frac{\mathbb{E}[(E_{t+1}^i\wedge K)^2]}{K^2\mathbb{E}[E_{t+1}^i\wedge K]}\right)\right)\sigma^2 \tag{C.221}$$

$$+\left(32L^2\eta^2K^2+\frac{8Ls\eta}{(n+1)}\max_i\left(\frac{\mathbb{E}[(E_{t+1}^i\wedge K)^2]}{\mathbb{E}[E_{t+1}^i\wedge K]}\right)+5600L^2s^2\eta^2\max_i\left(\frac{\mathbb{E}[(E_{t+1}^i\wedge K)^2]}{\mathbb{E}[E_{t+1}^i\wedge K]}\right)\right)G^2. \tag{C.222}$$

C.2 Detailed simulation environment

From Algorithm 8, one must note that local weights are reset with the central model only when being contacted by the central server. Hence initially we have $w_0^i = w_0$, but at time t we may have $w_t^i \neq w_t$, see Figure C.1.

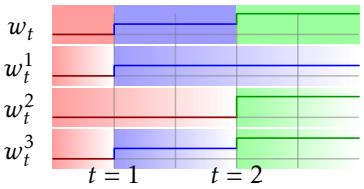


Figure C.1: Example of asynchronous updates with $n = 3$ nodes and selection size $s = 2$. At $t = 0$, all clients are initialized withe the same value. At time $t = 1$, clients $\{1, 3\}$ are selected, and at time $t = 2$, clients $\{2, 3\}$. At time $t = 2$, client 2 is reporting updates computed on outdated parameter.

C.2.1 Implementation of concurrent works

In Section 6.5 we have simulated experiments and run the code for the concurrent approaches FedAvg, QuAFL, and FedBuff. FedAvg is a standard synchronous method. At the beginning of each round, the central node s selects clients uniformly at random and broadcast its current model. Each of these clients take the central server value and then performs exactly K local

steps, and then sends the resulting model progress back to the server. The server then computes the average of the s received models and updates its model. In this synchronous structure, the server must wait in each round for the slowest client to complete its update. QuAFL is an asynchronous method that randomly selects s clients at each server invocation. The server then replaces its model with a convex combination of the received models and its current model. Also, the s receiving clients replace their local model with a convex combination between their current model and the model of the receiving server. In FedBuff, clients compute local training asynchronously as well, with the help of a buffer. Once the buffer is filled with Z different client updates, the server averages the buffer updates and performs a gradient step on the computed average. Then the buffer is reset to zero and the available clients get the server model as a new starting point.

C.2.2 Discussion on simulated runtime

We based our simulations mainly on the code developed by J. Nguyen et al. 2022: we assume a server and n clients, each of which initially has a model copy. We assume that, at each time step t (for the central server), a batch of s clients are sampled at random without replacement. For the client i, the inter-arrival time of two successive requests are therefore independent and distributed according to a geometric distributions of parameter s/n. The time elapsed from the last renewal is distributed according to the stationary distribution of the age process (assuming that the renewal is stationary), which is also distributed according to a geometric random variable with the same parameter s/n. We assume that the clients have different computational speeds. For this purpose E_t^i is distributed according to a geometrical distribution of parameter λ^i: $E_t^i \sim \text{Geom}(\lambda^i)$. The parameter λ^i is 1/2 for fast clients and 1/16 for slow clients; the expected running time $\mathbb{E}[E_t^i]$ is 2 and 16, respectively. The training dataset is distributed among the clients so that each of them has access to an equal portion of the training data (whether it is IID or non-IIID). We track the performance of each algorithm by evaluating the server's model against an unseen validation dataset. We measure the loss and accuracy of the model in terms of simulation time, server steps, and total local steps taken by clients.

To adequately capture the time spent on the server side for computations and orchestration of centralized learning, two quantities are implemented: the server waiting time (the time the server waits between two consecutive calls) and the server interaction time (the time the server takes to send and receive the required data). In all experiments, they are set to 4 and 3, respectively.

For each global step, the FedAvg runtime is the sum of the server interaction time (see above) and the local step runtime of the slowest selected client times the number of maximum epochs K (we wait until all clients have computed all their local epochs in this synchronous setting). For QuAFL and `FAVANO`, the duration of a global step is simply the sum of the server interaction time and the server waiting time. For FedBuff, the runtime is the sum of the server interaction time and the time spent feeding the buffer of size Z. The waiting time for feeding the buffer depends on the respective local runtimes of the slow and fast clients, as well as on the ratio between slow and fast clients: in the code, we reset a counter at the beginning of each global step and read the runtime when the Z^{th} local update arrives.

C.2.3 Detailed results

Below we provide further insight into the experiments described in Section 6.5. We present figures for loss, variance ($\sum_{i=1}^{n} \|w_t^i - w_t\|^2$), but also for accuracy (evaluated on the held-out test set on the server side) in terms of time, but also in terms of total local steps and total server

steps.

We find that FAVANO and other asynchronous methods, when time rather than the number of server steps FAVANO - and more generally asynchronous methods - can achieve significant speedups on these metrics compared to FedAvg. This is due to asynchronous communication allowing rounds to complete faster without always having to wait for slower nodes to complete their local computations. Although this behaviour is simulated, we believe it reflects the practical potential of FAVANO.

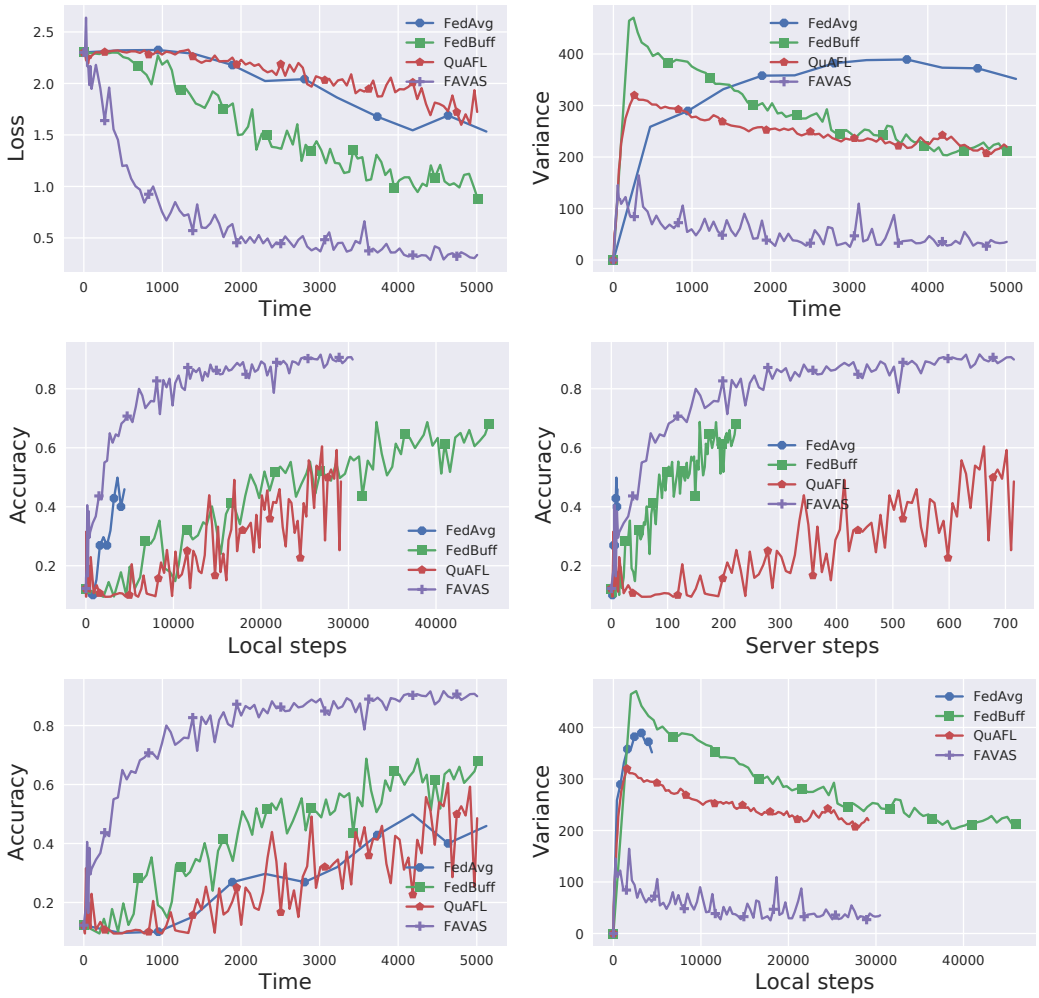


Figure C.2: Validation loss/accuracy and variance on the MNIST dataset with a non-iid split in between $n = 100$ total nodes. In this particular experiment, one ninth of the clients are defined as fast.

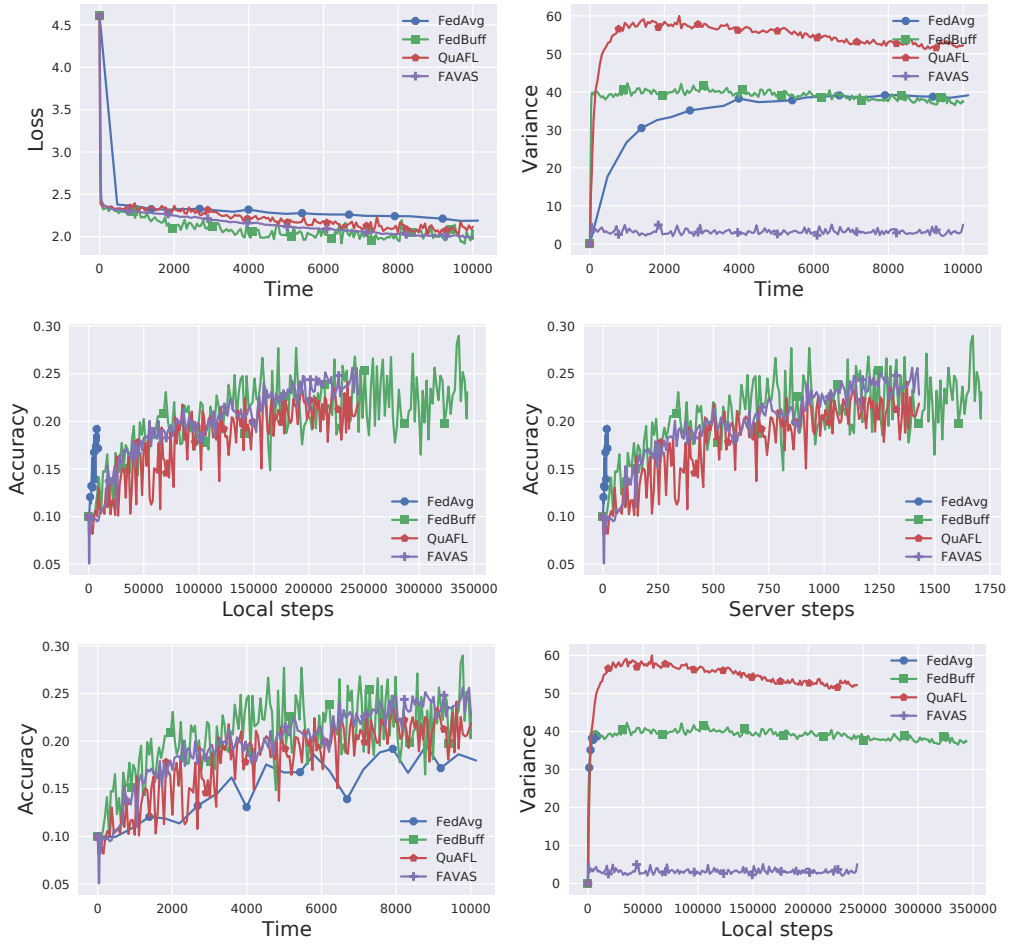


Figure C.3: Validation loss/accuracy and variance on the CIFAR-10 dataset with a non-iid split in between $n = 100$ total nodes.

We refer to FAVANO[QNN] when training a neural network with low bit precision arithmetic. We ran the code [1] from LUQ (Chmiel et al. 2021) and adapted it to our datasets and the FL framework. During FAVANO[QNN] training, 3-bits quantization for weights and activation are used, 4 bits quantization for neural gradients is used.

[1] https://openreview.net/forum?id=clwYez4n8e8

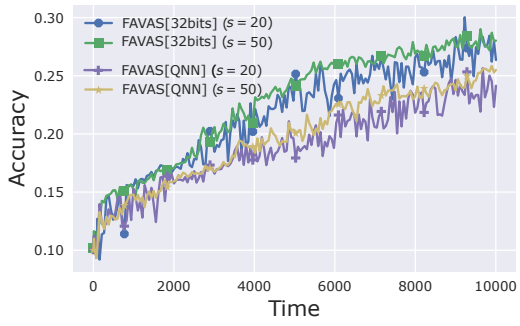


Figure C.4: Validation accuracy on the CIFAR-10 dataset with a non-iid split in between $n = 100$ total nodes. The amount s of selected clients at each round is varied. FAVANO[QNN] is the quantized version of FAVANO[32bits].

In Figure C.4 we analyse the effects of quantization and the influence of the number of randomly selected clients s on the convergence behaviour. As expected, we find that higher s improve the performance of FAVANO. Quantizing the neural network degrades the convergence behaviour of the algorithm, but, even if the weights and activation functions are highly quantized - as in the scenario we are considering-, the results are close to their full-precision counterpart.

Appendix D

Supplementary material of Chapter 7

D.1 Notations and definitions

In the proof section below we will refer on the following notations. For $0 \leq k \leq T$ consider the filtration $\mathcal{F}_k$ defined as $\mathcal{F}_k = \sigma(\{w_\ell, \ell \leq k, K_m, m < k\})$. We define the *virtual iterates* μ_k as follows:

$$\begin{cases} \mu_0 = w_0, \\ \mu_1 = \mu_0 - \eta \sum_{i \in \mathcal{S}_0} \frac{1}{np_i} \tilde{g}_i(w_0), \\ \mu_{k+1} = \mu_k - \frac{\eta}{np_{K_k}} \tilde{g}_{K_k}(w_k), \quad k \geq 1\,. \end{cases} \tag{D.1}$$

D.2 Proofs of Section 7.4

We split the proof of Theorem 14 into several steps. First we bound the quantity of interest $\sum_{k=0}^{T} \mathbb{E}[\|\nabla f(w_k)\|^2]$ in terms of norms of difference between exact and virtual iterations $\left\|\mu_k - w_k\right\|^2$ defined in (D.1). More precisely, the following statement holds:

Lemma 53. *Assume A14 to A17 and let the learning rate η satisfy $\eta \leq \frac{n^2}{8L\sum_{i=1}^{n} \frac{1}{p_i}}$. Then for the iterates $(w_k)_{k\geq 0}$ of* `Generalized AsyncSGD` *it holds that*

$$\frac{\eta}{4(T+1)} \sum_{k=0}^{T} \mathbb{E}[\|\nabla f(w_k)\|^2] \leq \frac{f(\mu_0) - \mathbb{E}[f(\mu_{T+1})]}{T+1} + \frac{\eta L^2}{2} \frac{1}{T+1} \sum_{k=0}^{T} \mathbb{E}[\|\mu_k - w_k\|^2] + \eta^2 L \sum_{i=1}^{n} \frac{2G^2 + \sigma^2}{n^2 p_i}. \tag{D.2}$$

Proof. Using the smoothness assumption **A**15 and the definition of μ_{k+1} from (D.1), we obtain the following descent inequality:

$$\mathbb{E}\left[f(\mu_{k+1}) \middle| \mathcal{F}_k\right] - f(\mu_k) \leq -\eta \mathbb{E}\left[\langle \nabla f(\mu_k), \frac{1}{np_{K_k}} \widetilde{g}_{K_k}(w_k)\rangle \middle| \mathcal{F}_k\right] + \frac{\eta^2 L}{2} \mathbb{E}\left[\|\frac{1}{np_{K_k}} \widetilde{g}_{K_k}(w_k)\|^2 \middle| \mathcal{F}_k\right]. \tag{D.3}$$

With the unbiasedness property of stochastic gradients and **A**16, we get

$$\mathbb{E}\Big[f(\mu_{k+1})\Big|\mathcal{F}_k\Big]-f(\mu_k)\leq-\eta\mathbb{E}\left[\langle\nabla f(\mu_k),\frac{1}{np_{K_k}}\nabla f_{K_k}(w_k)\rangle\middle|\mathcal{F}_k\right]+\eta^2L\mathbb{E}\left[\|\frac{1}{np_{K_k}}\nabla f_{K_k}(w_k)\|^2\middle|\mathcal{F}_k\right]+\eta^2L\sigma^2\sum_{i=1}^n\frac{1}{n^2p_i} \tag{D.4}$$

$$=-\eta\langle\nabla f(\mu_k),\nabla f(w_k)\rangle+\eta^2L\mathbb{E}\left[\|\frac{1}{np_{K_k}}\nabla f_{K_k}(w_k)\|^2\middle|\mathcal{F}_k\right]+\eta^2L\sigma^2\sum_{i=1}^n\frac{1}{n^2p_i}. \tag{D.5}$$

In the last equality we used that $\mathbb{E}\left[\langle\nabla f(\mu_k),\frac{1}{np_{K_k}}\nabla f_{K_k}(w_k)\rangle\middle|\mathcal{F}_k\right]=\langle\nabla f(\mu_k),\nabla f(w_k)\rangle$. Now we introduce a notation

$$\Delta_k=\mathbb{E}\Big[f(\mu_{k+1})\Big|\mathcal{F}_k\Big]-f(\mu_k).$$

Since $\langle a,b\rangle=1/2(\|a\|^2+\|b\|^2-\|a-b\|^2)$ for any $a,b\in\mathbb{R}^d$, we get that

$$\Delta_k\leq-\frac{\eta}{2}(\|\nabla f(\mu_k)\|^2+\|\nabla f(w_k)\|^2-\|\nabla f(w_k)-\nabla f(\mu_k)\|^2)+\eta^2L\mathbb{E}\left[\|\frac{1}{np_{K_k}}\nabla f_{K_k}(w_k)\|^2\middle|\mathcal{F}_k\right]+\eta^2L\sigma^2\sum_{i=1}^n\frac{1}{n^2p_i} \tag{D.6}$$

$$\leq-\frac{\eta}{2}\|\nabla f(w_k)\|^2+\frac{\eta}{2}L^2\|\mu_k-w_k\|^2+\eta^2L\mathbb{E}\left[\|\frac{1}{np_{K_k}}(\nabla f_{K_k}(w_k)-\nabla f(w_k)+\nabla f(w_k))\|^2\middle|\mathcal{F}_k\right]+\eta^2L\sigma^2\sum_{i=1}^n\frac{1}{n^2p_i} \tag{D.7}$$

$$\leq-\frac{\eta}{2}\|\nabla f(w_k)\|^2+\frac{\eta}{2}L^2\|\mu_k-w_k\|^2+2\eta^2L(\mathbb{E}\left[\|\frac{1}{np_{K_k}}(\nabla f_{K_k}(w_k)-\nabla f(w_k))\|^2\middle|\mathcal{F}_k\right]+\|\nabla f(w_k)\|^2\sum_{i=1}^n\frac{1}{n^2p_i}) \tag{D.8}$$

$$+\eta^2L\sigma^2\sum_{i=1}^n\frac{1}{n^2p_i}. \tag{D.9}$$

Applying the bounded gradient dissimilarity assumption **A**17, we get

$$\Delta_k\leq-\frac{\eta}{2}\|\nabla f(w_k)\|^2+\frac{\eta}{2}L^2\|\mu_k-w_k\|^2+2\eta^2L(\sum_{i=1}^n\frac{G^2}{n^2p_i}+\|\nabla f(w_k)\|^2\sum_{i=1}^n\frac{1}{n^2p_i})+\eta^2L\sigma^2\sum_{i=1}^n\frac{1}{n^2p_i} \tag{D.10}$$

$$\leq(-\frac{\eta}{2}+2\eta^2L\sum_{i=1}^n\frac{1}{n^2p_i})\|\nabla f(w_k)\|^2+\frac{\eta}{2}L^2\|\mu_k-w_k\|^2+2\eta^2L\sum_{i=1}^n\frac{G^2}{n^2p_i}+\eta^2L\sigma^2\sum_{i=1}^n\frac{1}{n^2p_i}. \tag{D.11}$$

As a consequence by taking $\eta\leq\frac{n^2}{8L\sum_{i=1}^n\frac{1}{p_i}}$ and substituting for Δ_k, we get

$$\frac{\eta}{4}\|\nabla f(w_k)\|^2\leq f(\mu_k)-\mathbb{E}\Big[f(\mu_{k+1})\Big|\mathcal{F}_k\Big]+\frac{\eta L^2}{2}\|\mu_k-w_k\|^2+2\eta^2L\sum_{i=1}^n\frac{G^2}{n^2p_i}+\eta^2L\sigma^2\sum_{i=1}^n\frac{1}{n^2p_i}. \tag{D.12}$$

Now taking sum for $k\in\{0,\dots,T\}$, we get

$$\frac{\eta}{4(T+1)}\sum_{k=0}^T\mathbb{E}[\|\nabla f(w_k)\|^2]\leq\frac{f(\mu_0)-\mathbb{E}[f(\mu_{K+1})]}{T+1}+\frac{\eta L^2}{2}\frac{1}{T+1}\sum_{k=0}^T\mathbb{E}[\|\mu_k-w_k\|^2] \tag{D.13}$$

$$+\eta^2L\sum_{i=1}^n\frac{2G^2+\sigma^2}{n^2p_i}. \tag{D.14}$$

□

In order to apply the result of Lemma 53, one needs to provide an upper bound on the correction term $\mathbb{E}[\left\|\mu_k - w_k\right\|^2]$. As explained in Section 7.4, the virtual iterates deviation from the true $\{w_k\}_{k>0}$ is made of all the gradients computed (on potentially outdated w's) and not applied yet. We can introduce the sets $\{\mathcal{I}_k\}_{k>0}$, as the sets of time and client indexes whose gradients are still on fly at time k. With $\mathcal{S}_0$ being the set of initial active workers from `Generalized AsyncSGD`, there are defined by the recursion:

$$\mathcal{I}_1 = \{(i,0) | i \in \mathcal{S}_0, i \neq J_0\} \tag{D.15}$$

$$\mathcal{I}_{k+1} = \begin{cases} \mathcal{I}_k & \text{if } I_k = k, \\ \mathcal{I}_k \setminus (J_k, I_k) \cup (K_k, k) & \text{otherwise.} \end{cases} \tag{D.16}$$

As $\left\|\mu_k - w_k\right\|$ represents the norm of gradients, it is easier to introduce the sets $\{\mathcal{G}_k\}_{k>0}$, as the set of gradients *scaled* with their respective weight $\frac{-1}{np_i}$ for each client i, that correspond to the indexes in $\{\mathcal{I}_k\}_{k>0}$:

$$\mathcal{G}_k = \{-\frac{1}{np_i}\tilde{g}_i(w_j) | (i,j) \in \mathcal{I}_k\}. \tag{D.17}$$

In the following lines, we will show that the sets $\{\mathcal{G}_k\}_{k>0}$ (and as a consequence the sets $\{\mathcal{I}_k\}_{k>0}$) have a constant cardinal: the number of running tasks in `Generalized AsyncSGD` is fixed during the whole optimization process, and only depends on the initialization.

Remark 54. *The number of running tasks is constant, but the number of active nodes is not! If there is a very slow client i, the number of active clients can be reduced to* 1*: all tasks are currently processed in the queue of client i.*

Lemma 55. *For the sequence $(w_k)_{k\geq 0}$ of updates produced by `Generalized AsyncSGD` and for the sequence of virtual updates $(\mu_k)_{k\geq 0}$ defined in* (D.1)*, it holds that*

$$\mu_1 - w_1 = -\eta \sum\nolimits_{i\in\mathcal{S}_0} \mathbb{1}\{i \neq J_0\} \frac{1}{np_i}\tilde{g}_i(w_0), \tag{D.18}$$

$$\mu_{k+1} - w_{k+1} = -\eta \sum\nolimits_{i\in\mathcal{S}_0} \mathbb{1}\{i \neq J_0\} \frac{1}{np_i}\tilde{g}_i(w_0) + \eta \sum\nolimits_{r=1}^{k} (\frac{1}{np_{J_r}}\tilde{g}_{J_r}(w_{I_r}) - \frac{1}{np_{K_r}}\tilde{g}_{K_r}(w_r)), \quad k \geq 1. \tag{D.19}$$

Proof. The proof follows from the definition of recurrence (D.1). Indeed, first we can note that

$$\mu_1 - w_1 = (w_0 - \eta \sum\nolimits_{i\in\mathcal{S}_0} \frac{1}{np_i}\tilde{g}_i(w_0)) - (w_0 - \eta \frac{1}{np_{J_0}}\tilde{g}_{J_0}(w_{I_0})) \tag{D.20}$$

$$= (w_0 - \eta \sum\nolimits_{i\in\mathcal{S}_0} \frac{1}{np_i}\tilde{g}_i(w_0)) - (w_0 - \eta \frac{1}{np_{J_0}}\tilde{g}_{J_0}(w_0)) \tag{D.21}$$

$$= -\eta \sum\nolimits_{i\in\mathcal{S}_0} \mathbb{1}\{i \neq J_0\} \frac{1}{np_i}\tilde{g}_i(w_0). \tag{D.22}$$

Now for a general iteration number k we have:

$$\mu_{k+1} - w_{k+1} = (\mu_k - \eta \frac{1}{np_{K_k}} \tilde{g}_{K_k}(w_k)) - (w_k - \eta \frac{1}{np_{J_k}} \tilde{g}_{J_k}(w_{I_k})) \tag{D.23}$$

$$= (\mu_k - w_k) + \eta(\frac{1}{np_{J_k}} \tilde{g}_{J_k}(w_{I_k}) - \frac{1}{np_{K_k}} \tilde{g}_{K_k}(w_k)) \tag{D.24}$$

$$= (\mu_1 - w_1) + \sum_{r=1}^{k} \eta(\frac{1}{np_{J_r}} \tilde{g}_{J_r}(w_{I_r}) - \frac{1}{np_{K_r}} \tilde{g}_{K_r}(w_r)) \tag{D.25}$$

$$= -\eta \sum_{i \in \mathcal{S}_0} \mathbb{1}\{i \neq J_0\} \frac{1}{np_i} \tilde{g}_i(w_0) + \eta \sum_{r=1}^{k} (\frac{1}{np_{J_r}} \tilde{g}_{J_r}(w_{I_r}) - \frac{1}{np_{K_r}} \tilde{g}_{K_r}(w_r)). \tag{D.26}$$

□

Lemma 56. *The sets $\{\mathcal{G}_k\}_{k\geq 0}$ have constant cardinal and compile all the gradients in computation at step $k > 0$:*

$$\begin{cases} (i) & |\mathcal{G}_k| = |\mathcal{G}_1| = |\mathcal{S}_0| - 1, \\ (ii) & \mu_k - w_k = \eta \sum_{g \in \mathcal{G}_k} g. \end{cases} \tag{D.27}$$

Proof. **Step (i)**: We are going to proof the result by induction. Assume $|\mathcal{G}_k| = |\mathcal{S}_0| - 1$ for some k. If $I_k = k$ we can immediately conclude that $|\mathcal{G}_{k+1}| = |\mathcal{G}_k|$. Otherwise, $I_k < k$, hence there exists some $i \in [n]$ such that $-\frac{1}{np_i} \tilde{g}_i(w_{I_k}) \in \mathcal{G}_k$. In particular, client J_k is the client that finishes computation at step k, thus $-\frac{1}{np_{J_k}} \tilde{g}_{J_k}(w_{I_k}) \in \mathcal{G}_k$. As a consequence, $|\mathcal{G}_k \setminus \{-\frac{1}{np_{J_k}} \tilde{g}_{J_k}(w_{I_k})\}| = |\mathcal{S}_0| - 2$. Furthermore, by definition, all gradients in $\mathcal{G}_k$ are taken on models older than k. And by taking $I_k < k$, we obtain $-\frac{1}{np_{K_k}} \tilde{g}_{K_k}(w_k) \notin \mathcal{S}_k \setminus \{-\frac{1}{np_{J_k}} \tilde{g}_{J_k}(w_{I_k})\}$. It concludes $|\mathcal{G}_{k+1}| = |\mathcal{G}_k|$.

Step (ii): We also prove it by induction. It is valid for $k = 1$. Now assume $\mu_k - w_k = \sum_{g \in \mathcal{G}_k} g$, for some $k > 1$.

$$\mu_{k+1} - w_{k+1} = (\mu_k - w_k) + \frac{1}{np_{J_k}} \tilde{g}_{J_k}(w_{I_k}) - \frac{1}{np_{K_k}} \tilde{g}_{K_k}(w_k) \tag{D.28}$$

$$= (\mu_k - w_k) + (\mathbb{1}_{\{0\}}(I_k) \frac{1}{np_{J_k}} \tilde{g}_{J_k}(w_0) + \cdots + \mathbb{1}_{\{k\}}(I_k) \frac{1}{np_{J_k}} \tilde{g}_{J_k}(w_k)) - \frac{1}{np_{K_k}} \tilde{g}_{K_k}(w_k) \tag{D.29}$$

$$= (\mu_k - w_k) + \mathbb{1}_{\{0\}}(I_k) \frac{1}{np_{J_k}} \tilde{g}_{J_k}(w_0) + \cdots + \mathbb{1}_{\{k-1\}}(I_k) \frac{1}{np_{J_k}} \tilde{g}_{J_k}(w_{k-1}) \tag{D.30}$$

$$+ (\mathbb{1}_{\{k\}}(I_k) \frac{1}{np_{J_k}} \tilde{g}_{J_k}(w_k) - \frac{1}{np_{K_k}} \tilde{g}_{K_k}(w_k)). \tag{D.31}$$

By induction, we have:

$$\mu_{k+1} - w_{k+1} = \sum_{g \in \mathcal{G}_k} g + \mathbb{1}_{\{0\}}(I_k) \frac{1}{np_{J_k}} \tilde{g}_{J_k}(w_0) + \cdots + \mathbb{1}_{\{k-1\}}(I_k) \frac{1}{np_{J_k}} \tilde{g}_{J_k}(w_{k-1}) \tag{D.32}$$

$$+ (\mathbb{1}_{\{k\}}(I_k) \frac{1}{np_{J_k}} \tilde{g}_{J_k}(w_k) - \frac{1}{np_{K_k}} \tilde{g}_{K_k}(w_k)). \tag{D.33}$$

If $I_k = k$ we have $J_k = K_k$: some client contributes instantaneously. This results in:

$$\mu_{k+1} - w_{k+1} = \sum_{g\in\mathcal{G}_k} g + \underbrace{\mathbb{1}_{\{0\}}(I_k)\frac{1}{np_{J_k}}\tilde{g}_{J_k}(w_0) + \cdots + \mathbb{1}_{\{k-1\}}(I_k)\frac{1}{np_{J_k}}\tilde{g}_{J_k}(w_{k-1})}_{=0} \tag{D.34}$$

$$+ \underbrace{(\mathbb{1}_{\{k\}}(I_k)\tilde{g}_{J_k}(w_k) - \frac{1}{np_{K_k}}\tilde{g}_{K_k}(w_k))}_{=0} \tag{D.35}$$

$$= \sum_{g\in\mathcal{G}_{k+1}} g. \tag{D.36}$$

Same idea as in Step (i) allows us to conclude $\mu_{k+1} - w_{k+1} = \sum_{g\in\mathcal{G}_{k+1}} g$ when $I_k < k$. □

Lemma 57. *For the sequence $(w_k)_{k\geq 0}$ of updates produced by* `Generalized AsyncSGD` *and for the sequence of virtual updates $(\mu_k)_{k\geq 0}$ defined in* (D.1)*, it holds that*

$$\frac{1}{T+1}\sum_{k=0}^{T}\mathbb{E}[\|\mu_k - w_k\|^2] \leq 2\eta^2 \mathrm{C}\sum_{i=1}^{n}\frac{\mathrm{m}_{i,0}^T}{(n^2p_i^2)(T+1)}(2G^2+\sigma^2) \tag{D.37}$$

$$+ 4\eta^2\mathrm{C}\frac{\mathbb{E}[\|\nabla f(w_0)\|^2]\,\mathrm{m}_0^T}{(n^2p_i^2)(T+1)} \tag{D.38}$$

$$+ 4\eta^2\mathrm{C}\sum_{k=1}^{T}\frac{\mathrm{m}_k^T}{(n^2p_i^2)(T+1)}\mathbb{E}[\|\nabla f(w_k)\|^2] \tag{D.39}$$

$$+ 2\eta^2\mathrm{C}\sum_{i=1}^{n}\frac{\sum_{k=1}^{K}\mathrm{m}_{i,k}^T}{(n^2p_i^2)(T+1)}(2G^2+\sigma^2). \tag{D.40}$$

Proof. Using the statement of Lemma 56, we bound the expected value of the correction term as follows:

$$\mathbb{E}[\|\mu_k - w_k\|^2] = \eta^2\mathbb{E}[\|\sum_{g\in\mathcal{G}_k} g\|^2] \leq \eta^2\mathbb{E}[|\mathcal{G}_k|\sum_{g\in\mathcal{G}_k}\|g\|^2] \leq \eta^2\mathbb{E}[\mathrm{C}\sum_{g\in\mathcal{G}_k}\|g\|^2]. \tag{D.41}$$

As the cardinal of sets $|\mathcal{G}_k| := \mathrm{C}$ is constant among iterations, and in particular it is independent from g's, we can simplify the form above.

We also introduce the set U_k:

$$U_k = \{i \in \{1,\ldots,n\} | X_{i,k} > 0\}. \tag{D.42}$$

Hence, from (D.41) and **A**16, we get

$$\mathbb{E}[\|\mu_k - w_k\|^2] \leq \eta^2\mathrm{C}\mathbb{E}[\sum_{(i,j)\in\mathcal{I}_k\cup\{U_k\times\{0\}\}}\frac{1}{n^2p_i^2}2\|\nabla f_i(w_j)\|^2 + 2\sigma^2] \tag{D.43}$$

$$\leq \eta^2\mathrm{C}\mathbb{E}[\sum_{i=1}^{n}\frac{1}{n^2p_i^2}\underbrace{\mathbb{1}_{U_k\cap\mathcal{S}_0}(i)(4G^2+4\|\nabla f(w_0)\|^2+2\sigma^2)}_{\text{gradients on initial model}} \tag{D.44}$$

$$+ \sum_{j=1}^{k}\mathbb{1}_{\mathcal{I}_k}\big((i,j)\big)(4G^2+4\|\nabla f(w_j)\|^2+2\sigma^2)] \tag{D.45}$$

Now we average over T iterations:

$$\frac{1}{T+1}\sum_{k=0}^{T}\mathbb{E}[\|\mu_k - w_k\|^2] \leq 2\eta^2 \mathrm{C}\sum_{i=1}^{n}\Big(\sum_{k=0}^{T}\frac{\mathbb{P}(i\in U_k\cap\mathcal{S}_0)}{T+1}\Big)\Big(\frac{2G^2+\sigma^2}{n^2p_i^2}\Big) \tag{D.46}$$

$$+ 4\eta^2\mathrm{C}\,\mathbb{E}[\|\nabla f(w_0)\|^2]\sum_{i=1}^{n}\frac{1}{n^2p_i^2}\Big(\sum_{k=0}^{T}\frac{\mathbb{P}(i\in U_k\cap\mathcal{S}_0)}{T+1}\Big) \tag{D.47}$$

$$+ \frac{4}{T+1}\eta^2\mathrm{C}\sum_{i=1}^{n}\mathbb{E}\Big[\sum_{k=1}^{T}\frac{1}{n^2p_i^2}\sum_{j=1}^{k}\Big(\mathbb{1}_{\mathcal{I}_k}\big((i,j)\big)\|\nabla f(w_j)\|^2\Big)\Big] \tag{D.48}$$

$$+ \frac{4G^2+2\sigma^2}{T+1}\eta^2\mathrm{C}\sum_{i=1}^{n}\mathbb{E}\Big[\sum_{k=1}^{T}\frac{1}{n^2p_i^2}\sum_{j=1}^{k}\mathbb{1}_{\mathcal{I}_k}((i,j))\Big]. \tag{D.49}$$

We rearrange the last 2 terms:

$$\frac{1}{T+1}\sum_{k=0}^{T}\mathbb{E}[\|\mu_k - w_k\|^2] \leq 2\eta^2 \mathrm{C}\sum_{i=1}^{n}\frac{\sum_{k=1}^{T}\mathbb{P}(i\in U_k\cap\mathcal{S}_0)}{T+1}\Big(\frac{2G^2+\sigma^2}{n^2p_i^2}\Big) \tag{D.50}$$

$$+ 4\eta^2\mathrm{C}\,\mathbb{E}[\|\nabla f(w_0)\|^2]\sum_{i=1}^{n}\frac{1}{n^2p_i^2}\frac{\sum_{k=1}^{T}\mathbb{P}(i\in u_k\cap\mathcal{S}_0)}{T+1} \tag{D.51}$$

$$+ 4\eta^2\mathrm{C}\sum_{k=1}^{T}\mathbb{E}\Big[\frac{\sum_{i=1}^{n}\frac{1}{n^2p_i^2}\sum_{r=k}^{T}\mathbb{1}_{\mathcal{I}_r}((i,k))}{T+1}\|\nabla f(w_k)\|^2\Big] \tag{D.52}$$

$$+ 2\eta^2\mathrm{C}\sum_{k=1}^{T}\mathbb{E}\Big[\frac{\sum_{i=1}^{n}\frac{1}{n^2p_i^2}\sum_{r=k}^{T}\mathbb{1}_{\mathcal{I}_r}((i,k))}{T+1}(2G^2+\sigma^2)\Big]. \tag{D.53}$$

We can simplify the bounds with the following identity:

$$\sum_{r=k}^{T}\mathbb{1}_{\mathcal{I}_r}((i,k)) = \mathbb{1}_{\{i\}}(K_{k+1})\sum_{r=k}^{T}\mathbb{1}_{(\sum_{l=k}^{r}\mathbb{1}_{J_l=i})<X_{i,k}} = \mathrm{M}_{i,k}^{T}, \text{for } k>0. \tag{D.54}$$

And with a slight abuse of notation, we take: $\mathrm{M}_{i,0}^{T} = \sum_{k=1}^{T}\mathbb{1}_{U_k\cap\mathcal{S}_0}(i)$.

Combining the above bounds, we obtain

$$\frac{1}{T+1}\sum_{k=0}^{T}\mathbb{E}[\|\mu_k - w_k\|^2] \leq 2\eta^2\mathrm{C}\sum_{i=1}^{n}\frac{\mathbb{E}[\mathrm{M}_{i,0}^{T}]}{T+1}\Big(\frac{2G^2+\sigma^2}{n^2p_i^2}\Big) \tag{D.55}$$

$$+ 4\eta^2\mathrm{C}\,\mathbb{E}[\|\nabla f(w_0)\|^2]\sum_{i=1}^{n}\frac{1}{n^2p_i^2}\frac{\mathbb{E}[\mathrm{M}_{i,0}^{T}]}{T+1} \tag{D.56}$$

$$+ 4\eta^2\mathrm{C}\sum_{k=1}^{T}\frac{\sum_{i=1}^{n}\frac{1}{n^2p_i^2}\mathbb{E}[\mathrm{M}_{i,k}^{T}]}{T+1}\mathbb{E}[\|\nabla f(w_k)\|^2] \tag{D.57}$$

$$+ 2\eta^2\mathrm{C}\sum_{i=1}^{n}\frac{\sum_{k=1}^{T}\mathbb{E}[\mathrm{M}_{i,k}^{T}]}{T+1}\Big(\frac{2G^2+\sigma^2}{n^2p_i^2}\Big), \tag{D.58}$$

and the statement follows using the definition m_k^T. □

D.2.1 Proof of Theorem 14

We apply the bound Lemma 53 and use Lemma 57 to control the correction term $\frac{1}{T+1}\sum_{k=0}^{T}\mathbb{E}[\|\mu_k - w_k\|^2]$. Hence, we get

$$\frac{1}{4(T+1)}\sum_{k=0}^{T}\mathbb{E}[\|\nabla f(w_k)\|^2] \leq \frac{\mathbb{E}[f(\mu_0)-f(\mu_{T+1})]}{\eta(T+1)} + \frac{L^2}{2(T+1)}\sum_{k=0}^{T}\mathbb{E}[\|\mu_k - w_k\|^2] + \eta L\sum_{i}^{n}\frac{2G^2+\sigma^2}{n^2 p_i} \tag{D.59}$$

$$\leq \frac{\mathbb{E}[f(\mu_0)-f(\mu_{T+1})]}{\eta(T+1)} + \eta L\sum_{i}^{n}\frac{2G^2+\sigma^2}{n^2 p_i} \tag{D.60}$$

$$+ L^2\eta^2\mathrm{C}\left(\sum_{i=1}^{n}\frac{\sum_{k=0}^{T}\mathrm{m}_{i,k}^{T}}{n^2 p_i^2(T+1)}(2G^2+\sigma^2) + \frac{2\mathbb{E}[\|\nabla f(w_0)\|^2]\,\mathrm{m}_0^T}{T+1}\right) \tag{D.61}$$

$$+ \frac{L^2\eta^2\mathrm{C}}{T+1}\sum_{k=1}^{T}2\,\mathrm{m}_k^T\,\mathbb{E}[\|\nabla f(w_k)\|^2]. \tag{D.62}$$

Hence we have:

$$\frac{1}{T+1}\sum_{k=0}^{T}(1/4 - 2\,\mathrm{m}_k^T L^2\eta^2\mathrm{C})\mathbb{E}[\|\nabla f(w_k)\|^2] \leq \frac{\mathbb{E}[f(\mu_0)-f(\mu_{T+1})]}{\eta(T+1)} + \eta L\sum_{i}^{n}\frac{2G^2+\sigma^2}{n^2 p_i} \tag{D.63}$$

$$+ L^2\eta^2\mathrm{C}\sum_{i=1}^{n}\frac{\sum_{k=0}^{T}\mathrm{m}_{i,k}^{T}}{n^2 p_i^2(T+1)}(2G^2+\sigma^2). \tag{D.64}$$

Now we impose the step size condition

$$\eta \leq \sqrt{\frac{1}{16L^2\,\mathrm{C}\max_{k\in\{1,\ldots,T\}}\mathrm{m}_k^T}}, \tag{D.65}$$

which enable us to conclude that

$$\frac{1}{8(T+1)}\sum_{k=0}^{T}\mathbb{E}[\|\nabla f(w_k)\|^2] \leq \frac{\mathbb{E}[f(\mu_0)-f(\mu_{T+1})]}{\eta(T+1)} + \eta L\sum_{i}^{n}\frac{2G^2+\sigma^2}{n^2 p_i} \tag{D.66}$$

$$+ L^2\eta^2\mathrm{C}\sum_{i=1}^{n}\frac{\sum_{k=0}^{T}\mathrm{m}_{i,k}^{T}}{n^2 p_i^2(T+1)}(2G^2+\sigma^2), \tag{D.67}$$

and the statement follows.

D.2.2 Influence of the strong growth condition

The assumption **A**16 can be generalized to the *strong growth condition* (S. Vaswani, F. Bach, and Schmidt 2019):

$$\mathbb{E}[\|\widetilde{g}_i(x) - \nabla f_i(x)\|^2] \leq \sigma^2 + \rho^2\|\nabla f_i(x)\|^2.$$

All previous derivations are impacted by a factor ρ^2. But the proofs remain the same. In particular, the quantity Δ_k from Lemma 53 can be bounded as:

$$\Delta_k \leq (-\frac{\eta}{2} + 2\eta^2 L \sum_{i=1}^n \frac{1+\rho^2}{n^2 p_i})\|\nabla f(w_k)\|^2 + \frac{\eta}{2} L^2 \|\mu_k - w_k\|^2 + 2\eta^2 L \sum_{i=1}^n \frac{(1+\rho^2)G^2}{n^2 p_i} \tag{D.68}$$

$$+ \eta^2 L \sigma^2 \sum_{i=1}^n \frac{1}{n^2 p_i}. \tag{D.69}$$

This slightly change the condition on the step size: $\eta \leq \frac{n^2}{8L\sum_{i=1}^n \frac{1+\rho^2}{p_i}}$. The rest of the proof is similarly impacted. We now impose the additional step size condition:

$$\eta \leq \sqrt{\frac{1}{(1+\rho^2)16L^2 \mathrm{C} \max_{k\in\{1,\dots,T\}} \mathrm{m}_k^T}}, \tag{D.70}$$

which enable us to conclude that

$$\frac{1}{8(T+1)} \sum_{k=0}^T \mathbb{E}[\|\nabla f(w_k)\|^2] \leq \frac{\mathbb{E}[f(\mu_0) - f(\mu_{T+1})]}{\eta(T+1)} + \eta L \sum_i^n \frac{2(1+\rho^2)G^2 + \sigma^2}{n^2 p_i} \tag{D.71}$$

$$+ L^2 \eta^2 \mathrm{C} \sum_{i=1}^n \frac{\sum_{k=0}^T \mathrm{m}_{i,k}^T}{n^2 p_i^2 (T+1)} (2(1+\rho^2)G^2 + \sigma^2). \tag{D.72}$$

D.3 Proofs of Section 7.5

D.3.1 Proof of Proposition 15

Given the assumptions, it is straightforward to check that the dynamics are those of the Markov process with the given generator.

Then the result follows from classical results in queuing theory: the Markov process corresponds to a Jackson quasi-reversible network with an explicit stationary distribution, see for instance Theorem 1.12 in Serfozo 1999.

Before continuing, we need a fundamental property of closed Jackson network which is the arrival Theorem also called MUSTA in the literature:

Theorem 58 (Arrival Theorem, Prop 4.35 in Serfozo 1999)**.** *Suppose the system is stationary. Upon arrival to a given node (i.e., just before waiting or being served in the queue), a task sees the network according to the distribution π_{C-1}, i.e.,*

$$\mathbb{P}(X_{\tau_{i,1}^-} = x) = \pi_{C-1}(x).$$

For the link with Palm probabilities, we also refer to Example 3.3.4. in Baccelli and Bremaud 2002.

Now, define S_i the first sojourn time on node i, i.e.,

$$S_i = \inf\{t \geq \tau_{i,1} | D_i(t) = X_i(\tau_{i,1}^-) + 1\} \tag{D.73}$$

Recall that, $\mathbb{E}^C$ corresponds to the stationary average for a system with C tasks (in particular the process X_t follows π_C for all times $t \geq 0$). We denote in turn by $\mathbb{E}^{C-1}$ the Palm probability associated to the event of an arrival at a given node (say i) and corresponding informally

to conditioning to $\tau_{i,1} = 0$. Using the Arrival Theorem, the distribution at time 0 at node i corresponds in this case to π_{C-1}, the dynamics under the Palm probabilities being unchanged (seeBaccelli and Bremaud 2002).

D.3.2 Proof of Proposition 16

Assuming the system is stationary and using the definition of $\mathrm{m}_{i,k}^T$ we have that for any k,

$$\mathrm{m}_{i,k}^T = \mathrm{m}_i^T = \mathbb{E}^{\mathrm{C}}\Big[\Big(\sum\nolimits_n \mathbb{1}(\tau_{i,1} \le T_n \le S_i)\Big) \wedge T\Big]. \tag{D.74}$$

Using the monotone convergence Theorem,

$$\lim_{T\to\infty} \mathrm{m}_{i,k}^T = m_i = \mathbb{E}^{\mathrm{C}}\Big[\sum\nolimits_n \mathbb{1}(\tau_{i,1} \le T_n \le S_i)\Big]. \tag{D.75}$$

We then use the arrival Theorem (58) for closed Jackson network and using the Palm probability, we can write that

$$\mathbb{E}^{\mathrm{C}}[\sum\nolimits_n \mathbb{1}(\tau_{i,1} \le T_n \le S_i)] = \mathbb{E}^{\mathrm{C}-1}[\sum\nolimits_n \mathbb{1}(0 \le T_n \le S_i)].$$

Then by using the stochastic intensity formula (see Definition 1.8.10 and Example 1.8.3. in Baccelli and Bremaud 2002):

$$\mathbb{E}^{\mathrm{C}-1}[\sum\nolimits_n \mathbb{1}(0 \le T_n \le S_i)] = \mathbb{E}^{\mathrm{C}-1}\Big[\int_0^{S_i} \sum\nolimits_{j=1}^n \mu_j \mathbb{1}(X_j(s) > 0)\, ds\Big]. \tag{D.76}$$

D.3.3 Computation of the constant Γ

$$\Gamma(c) = \frac{\mathbb{E}[X\mathbb{1}_{X+Y\le c}]}{\mathbb{E}[\mathbb{1}_{X+Y\le c}]}, \tag{D.77}$$

with X an Exp(1) and Y an Erlang(F,1) independent from each other. By integrating by parts:

$$\Gamma = \frac{\int_0^\infty \int_0^{c-y} xe^{-x} dx \mathbb{1}(y \le c) dP_Y(y)}{\mathbb{P}(X+Y \le c)}, \tag{D.78}$$

$$= \frac{\int_0^\infty (-(c-y)e^{-c+y} + \int_x 1_{x \le c-y} \mathbb{1}(y \le c) dP_X(x) dP_Y(y)}{\mathbb{P}(X+Y \le c)}, \tag{D.79}$$

$$= \frac{\int_0^\infty -(c-y)e^{-c+y} \mathbb{1}(y \le c) \frac{y^{F-1}}{(F-1)!} e^{-y} dy}{\mathbb{P}(X+Y \le c)} + 1, \tag{D.80}$$

$$= \frac{e^{-c}(-c^{F+1}/F! + Fc^{F+1}/(F+1)!)}{\mathbb{P}(X+Y \le c)} + 1, \tag{D.81}$$

$$= \frac{-e^{-c}c^{F+1}/(F+1)! + 1 - \sum_{k=1}^{F} e^{-c}c^k/(k)!}{1 - \sum_{k=1}^{F} e^{-c}c^k/(k)!}, \tag{D.82}$$

$$= \frac{\mathbb{P}(\sum_{i=1}^{F+2} E_i \le c)}{\mathbb{P}(\sum_{i=1}^{F+1} E_i \le c)}, \tag{D.83}$$

$$\tag{D.84}$$

D.3.4 Proof of Proposition 18

First note that:

$$\mathrm{m}_i(\iota) = \mathbb{E}^{\mathrm{C}-1}\Big[\int_0^{S_i} \sum\nolimits_{j=1}^{n} \mu_j \mathbb{1}(X_j^\iota(s) > 0)\, ds \Big] \le \lambda \mathbb{E}^{\mathrm{C}-1}[S_i].$$

Then it follows from the FIFO representation that

$$\mathbb{E}^{\mathrm{C}-1}(S_i) = \frac{1}{\mu_i}(\mathbb{E}^{\mathrm{C}-1}[X_i^\iota]) + 1)$$

which implies the claim.

D.4 Upper bounds simulations

D.4.1 Illustration of $G(\mathbf{p}, \eta)$ before optimization

We decide to simulate $n = 100$ nodes with $\mathrm{C} = 10$ initial tasks. Nodes can only be fast (sampled with $p_i = p$) or slow (sampled with $p_i = \frac{2}{n} - p$), and are evenly distributed. We estimate the values of $\mathrm{m}_{i,k}^T$ through Monte-Carlo and compute the upper bounds given in Theorem 14. All others constants are kept unitary to ease the computation. In Figure D.1, we plot the value of the previously mentioned upper-bound with respect to the step size η, for several sampling probabilities of fast node p. When the step size considered is small, all sampling strategies are equivalent. Whereas for large value of η, sampling around the uniform one is a good strategy. Large value of p, close to the limit $\frac{2}{n}$, hinders the bound because it increases the delays for slow nodes by sampling fast nodes quite often. In Figure 7.2 and Figure 7.3, we define grids of 50 values of p around the uniform one, and for each p we compute the exact optimal step size by solving the cube roots. The optimal values of the sampling p on the grid, and of the optimal step size are further used to compute the optimal bound and compare it against the one obtained with uniform sampling.

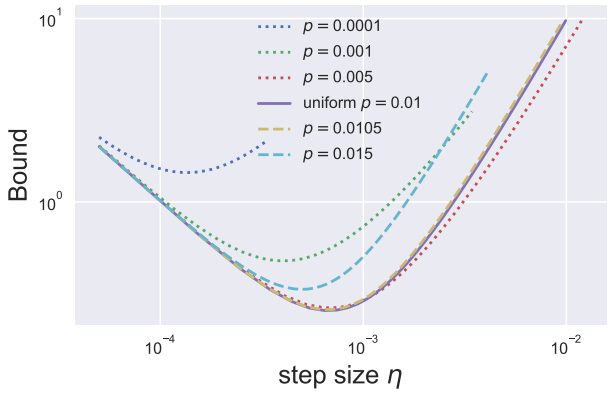


Figure D.1: Variation of the non-convex upper-bound with respect to the step size η, for $K = 10^4$ server steps and different values of sampling p. The maximum step size value is different for each case and equal to $\sqrt{\frac{1}{8L^2 \mathrm{C} max \mathrm{m}_k^T}}$.

D.4.2 Bounds w.r.t physical time

We want here to focus on the relative improvements of the upper bounds when time rather than CS steps is considered as fixed. Indeed, when we determine complexity in terms of number of communications, we don't take into account the time intervals between two successive arrivals at the central server. In particular, the results from Section 7.4 propose to sample more frequently slow nodes. But this results into an increased waiting time between two consecutive server steps. As a consequence, in this section, we choose a fixed unit of time $U = 1000$ and optimize the bounds for $T = \lambda(\mathbf{p}) \cdot U$ server steps, where $\lambda(\mathbf{p})$ is the average network speed.

We choose the sampling probabilities $\mathbf{p}$ and the step size η by solving the constrained optimization problem $\min_{\mathbf{p},\eta} G(\mathbf{p}, \eta)$ as a function of $\eta \leq \eta_{\max}(\mathbf{p})$, where

$$G(\mathbf{p}, \eta) = \frac{A}{\eta(\lambda(\mathbf{p})U + 1)} + \frac{\eta LB}{n} \sum_{i=1}^{n} \frac{1}{np_i} + \frac{\eta^2 L^2 B \mathrm{C}}{n} \sum_{i=1}^{n} \frac{\sum_{k=0}^{\lambda(\mathbf{p})U} \mathrm{m}_{i,k}^T}{np_i^2(\lambda(\mathbf{p})U + 1)}, \tag{D.85}$$

and where $A = \mathbb{E}[f(\mu_0) - f(\mu_{T+1})]$. In Figure D.2 we run the same simulation as in Section 7.4, for a fixed amount of time U. Taking into account a fixed amount of time U instead of CS epochs T also favours our approach. The experimental results suggest to sample less fast nodes. It reduces delays (in number of steps), but increases the average time spent between two consecutive server steps. This trade-off is key for optimizing the bounds. When the concurrency is small (w.r.t. n), uniform sampling appears as the best strategy. However, by taking $p = 8.5 \cdot 10^{-3}$, for full concurrency ($\mathrm{C} = n$), the bound can be reduced by 40%.

D.5 2 clusters under saturation

D.5.1 Example with 2 saturated clusters

In Section 7.5, we propose a study of the delays and queue lengths when the number of task goes to infinity with a rate controlled by some $\iota > 0$. In particular, we introduce the scaled intensities

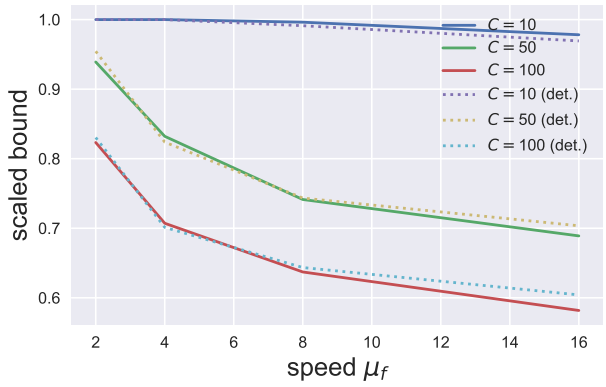


Figure D.2: Relative improvements of the upper bounds as a function of the speed for different concurrency levels.

for slow and fast nodes as:

$$\begin{cases} \gamma_s(\iota) = \frac{\max_{i\in[1,n]}(\theta_i)}{\theta_s} = \frac{\theta_s}{\theta_s} = 1 \\ \gamma_f(\iota) = \frac{\max_{i\in[1,n]}(\theta_i)}{\theta_f} = \frac{\theta_s}{\theta_f} = 1 + \underbrace{c_f \cdot \iota^{\alpha-1}}_{\text{deviation from slow speed}} \end{cases} \tag{D.86}$$

Note $c_f > 0$ and $\alpha \leq 1$ are parameters we are free to choose to match the number of tasks in the network. In particular, the total number of tasks also scales as follows: $\beta l^{1-\alpha} = \mathrm{C}+1$. Thanks to Proposition 18, we can bound the number of server steps when a task arrive and quits some node i as:

$$\lim_{\iota\to\infty} \iota^{\alpha-1}\, \mathrm{m}_i(\iota) \leq \lim_{\iota\to\infty} \frac{\lambda}{\mu_i}(\iota^{\alpha-1}\mathbb{E}[X_i^\iota] + 1), \tag{D.87}$$

where $\lambda = n_f\mu_s + (n - n_f)\mu_s$. Hence,

$$\lim_{\iota\to\infty} \iota^{\alpha-1}\, \mathrm{m}_i(\iota) \leq \frac{n_f\mu_s + (n-n_f)\mu_s}{\mu_i}(\frac{1}{c_f}\Gamma(c_f\beta) + 1) \tag{D.88}$$

We will further assume $n_f = \frac{n}{2}$, and $p_i = \frac{1}{n}$. Under this setting, we have $\Gamma(c_F\beta) \simeq 1$ and

$$\frac{\iota^{1-\alpha}}{c_f}\Gamma(c_f\beta) = \frac{1}{\gamma_f(\iota) - 1} \tag{D.89}$$

$$= \frac{1}{\frac{\theta_s}{\theta_f} - 1} \tag{D.90}$$

$$= \frac{1}{\frac{\mu_f p_s}{\mu_s p_f} - 1} \tag{D.91}$$

$$= \frac{1}{\frac{\mu_f}{\mu_s} - 1}. \tag{D.92}$$

For fast nodes, the delay can be simplified as:

$$\lim_{\iota\to\infty} \mathrm{m}_i(\iota) \leq \frac{n}{2}\frac{\mu_s+\mu_f}{\mu_f}\frac{1}{\frac{\mu_f}{\mu_s}-1}. \tag{D.93}$$

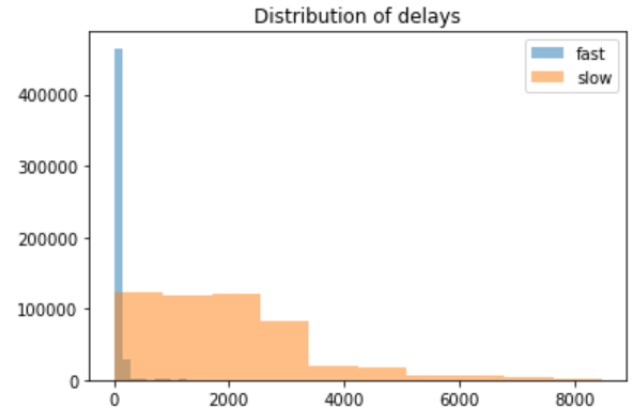
For slow nodes, this simplifies as:

$$\lim_{\iota\to\infty} \mathrm{m}_i(\iota) \leq \frac{n}{2}\frac{\mu_s+\mu_f}{\mu_s}(\frac{2}{n}\mathrm{C}-\frac{1}{\frac{\mu_f}{\mu_s}-1}). \tag{D.94}$$

In the following we consider $n = 10$ clients, split in two clusters of same size: fast nodes with rate $\mu_f = 1.2$, and slow nodes with rates $\mu_s = 1$. We simulate up to $T = 10^6$ server steps, and plot the distribution of the delays (in number of server steps). We saturate the network with $\mathrm{C} = 1000$ tasks. Hence we can estimate the following:

$$\begin{cases} \lim_{\iota\to\infty} \mathrm{m}_i(\iota) \leq \frac{n}{\frac{\mu_f}{\mu_s}-1} \simeq 5n, \quad \forall i \in [1, n_f], \\ \lim_{\iota\to\infty} \mathrm{m}_i(\iota) \leq (\frac{2\mathrm{C}}{n} - \frac{1}{\frac{\mu_f}{\mu_s}-1})n \simeq 195n, \quad \forall i \in [n_f+1, n]. \end{cases} \tag{D.95}$$

All delays bounds estimations have a closed form in the 2-cluster saturated regime: they only depend on the number of tasks in the network C, on the number of nodes n, and on the intensity of nodes μ_f, μ_s.




```
avg delay fast =  59.17222612780303
avg delay slow =  1938.58848847666
avg queue fast 5.926313126313127
avg queue slow 194.07368687368688
```

Figure D.3: Histogram of fast and slow delays (in number of server steps) for a uniform sampling scheme.

Our numerical experiment in Figure 7.5 gives an average delay of $59 \sim 5n$ for fast nodes. The average delay for slow nodes reaches the value $1938 \simeq 195n$. And the queue lengths also correspond to the expected values. It is also important to point out that the average delays are way smaller than the maximum delay experienced in the $K = 10^6$ steps. This further highlights the necessity to switch from analysis that depend on the τ_{max} quantity, to our analysis that only

depends on the expected delays.

D.5.2 Optimal sampling strategy under saturation

In the previous paragraph we kept the sampling probability p_i uniform. The previous computation gives

$$\frac{l^{1-\alpha}}{c_f}\Gamma(c_f\beta) = \frac{1}{\gamma_f^l - 1} \tag{D.96}$$

$$= \frac{1}{\frac{\theta_s}{\theta_f} - 1} \tag{D.97}$$

$$= \frac{1}{\frac{\mu_f p_s}{\mu_s p_f} - 1}. \tag{D.98}$$

Sticking to the previous assumptions, we want to minimize the quantity

$$G(\mathbf{p},\eta) = \frac{A}{\eta(T+1)} + \frac{\eta LB}{n}\Big(\sum_{i=1}^{n_f}\frac{1}{np} + \sum_{i=n_f+1}^{n}\frac{1}{n(\frac{1}{n\_n_f} - p\frac{n_f}{n-n_f})^{-1}}\Big) + \frac{\eta^2L^2BC}{n}\Big(\sum_{i=1}^{n_f}\frac{m_f}{np^2} + \sum_{i=n_f+1}^{n}\frac{m_s}{n(\frac{1}{n\_n_f} - p\frac{n_f}{n-n_f})^{-2}}\Big), \tag{D.99}$$

This is equivalent to minimizing:

$$G(\mathbf{p},\eta) = \frac{A}{\eta(T+1)} + \frac{\eta LB}{n}\Big(\sum_{i=1}^{n_f}\frac{1}{np} + \sum_{i=n_f+1}^{n}\frac{1}{n(\frac{1}{n\_n_f} - p\frac{n_f}{n-n_f})^{-1}}\Big) + \frac{\eta^2L^2BC}{n}\Big(\sum_{i=1}^{n_f}\frac{\frac{n_f\mu_s+(n-n_f)\mu_s}{\mu_f}\frac{1}{\frac{\mu_f p_s}{\mu_s p_f}-1}}{np^2} + \sum_{i=n_f+1}^{n}\frac{\frac{n_f\mu_s+(n-n_f)\mu_s}{\mu_s}(\frac{\mathrm{C}}{n-n_f} - \frac{n_f}{n-n_f}\frac{1}{\frac{\mu_f p_s}{\mu_s p_f}-1})}{n(\frac{1}{n-n_f} - p\frac{n_f}{n-n_f})^{-2}}\Big), \tag{D.100}$$

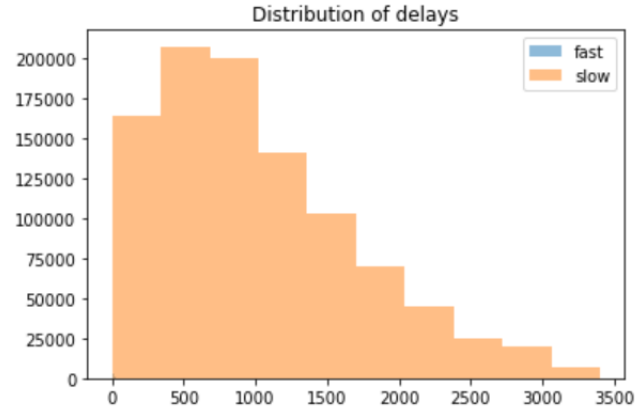
Hence we want to find $(\mathbf{p},\eta)$ that minimize the following:

$$G(\mathbf{p},\eta) = \frac{A}{\eta(T+1)} + \frac{\eta LB}{n}\Big(\sum_{i=1}^{n_f}\frac{1}{np} + \sum_{i=n_f+1}^{n}\frac{1}{n(\frac{1}{n\_n_f} - p\frac{n_f}{n-n_f})^{-1}}\Big) + \frac{\eta^2L^2BC}{n}\Big(\sum_{i=1}^{n_f}\frac{\frac{n_f\mu_s+(n-n_f)\mu_s}{\mu_f}\frac{1}{\frac{\mu_f}{\mu_s}(\frac{1}{p(n-n_f)}-\frac{n_f}{n-n_f})^{-1}}}{np^2} + \sum_{i=n_f+1}^{n}\frac{\frac{n_f\mu_s+(n-n_f)\mu_s}{\mu_s}(\frac{\mathrm{C}}{n-n_f} - \frac{n_f}{n-n_f}\frac{1}{\frac{\mu_f}{\mu_s}(\frac{1}{p(n-n_f)}-\frac{n_f}{n-n_f})^{-1}})}{n(\frac{1}{n-n_f} - p\frac{n_f}{n-n_f})^{-2}}\Big), \tag{D.101}$$

The uniform sampling strategy ($p = \frac{1}{n} = 0.1$) and higher probability values, give larger average delays.But very small sampling probabilities lead to a sharp increase in the delays. It is not easy to find a closed form formula of the optimal probability value $\mathbf{p}$. Our simulations

suggest an optimal value of $p = 7.5 \cdot 10^{-3}$.

In Figure D.4, we have run the same simulations as in Figure 7.5, except that we do not sample nodes uniformly at random. Instead, we sample fast nodes with a probability $p = 7.5 \cdot 10^{-3}$, and slow nodes with a probability $\frac{2}{n} - p$. Our simulations shows the average delay is divided by 10 and 2, for fast and slow nodes respectively.




```
avg delay fast =  5.368705035971223
avg delay slow =  1015.5117977385339
avg queue fast 0.01730941730941731
avg queue slow 199.9826905826906
```

Figure D.4: Histogram of fast and slow delays (in number of server steps) for an optimal sampling strategy.

D.6 3 clusters scaling regime under saturation

We consider three clusters of nodes of size n_f, $n_m - 1 - n_f$, and $n - 1 - n_m$, respectively. Nodes $\forall i \leq n_f$ are considered as fast, whereas nodes $\forall i > n_m$ are considered as slow (and will likely get more saturated).

The medium nodes ($\forall i \in [n_f + 1, n_m]$) have an intermediate computational speed. We assume nodes from the same cluster have the same intensity μ_f, μ_m, μ_s, for fast, medium, and slow nodes respectively. For practical reason we assume now that nodes $\forall i > n_m$ are the slowest ones, and has an intensity $\theta_s > \theta_j, \forall j < n$. This assumption is not restrictive due to the close nature of the network, and allows us to simplify the problem by splitting nodes into clusters.

This results in the *scaled* intensities $\gamma_f(\iota), \gamma_m(\iota), \gamma_s(\iota)$, where $\gamma_s(\iota) = 1$, $\gamma_m(\iota) = 1 + c_m \iota^{\alpha - 1}$, and $\gamma_f(\iota) = 1 + c_f \iota^{\delta - 1}$; with $\alpha \leq 1$ and $\delta > 1$. The constant task constraint translates into the existence of β such that $\beta \iota^{1-\alpha} = \mathrm{C} + 1$. The particular choice of $\alpha \leq 1$ in Van Kreveld, Dorsman, and Mandjes 2021 allows us to obtain traffic loads of nodes that tend to 1 as $\iota \to \infty$, and we could directly apply Corollary 2 from Van Kreveld, Dorsman, and Mandjes 2021. But this setting is inconsistent with the practical Federated Learning framework we consider in this section: in practice most of fast clients have an empty queue. Hence, we stick to $\delta > 1$, and we can apply the results of Corollary.3, Van Kreveld, Dorsman, and Mandjes 2021. This work gives a precise results on the queue length of saturated nodes, in the limit of high traffic loads. The queue length of the remaining queue are defined by the population size constraint. Note because the

unnormalized queue length of fast nodes converges to a finite-mean random variable, there is no need to scale it.

Proposition 59 (Corollary.3 in Van Kreveld, Dorsman, and Mandjes 2021)**.** *In stationary regime, as* $\iota \to \infty$, $X_i^\iota, \forall i \in [1, n_f]$ *become degenerate with value* 0, *and*

$$c_m \iota^{\alpha-1} X_i^\iota \to_{d.} \mathbb{E}\left[E_i \,|\, \sum_{j=n_f+1}^{n_m} \frac{E_j}{c_m} \leq \beta \right], \forall i \in [n_f+1, n_m], \tag{D.102}$$

with E_i *unit mean exponential distributions.*

As a consequence, using as before dominated convergence we can estimate the following expected value (expected stationary queue lengths of fast, medium, and slow nodes respectively):

$$\begin{cases} \lim_{\iota\to\infty} \mathbb{E}[X_i^\iota] = 0, \quad \forall i \in [1, n_f], \\ \lim_{\iota\to\infty} \iota^{\alpha-1} \mathbb{E}[X_i^\iota] = \frac{1}{c_m}\Gamma(c_m\beta), \quad \forall i \in [n_f+1, n_m], \\ \lim_{\iota\to\infty} \iota^{\alpha-1} \mathbb{E}[X_i^\iota] = \frac{1}{n-n_m}\left(\beta - (n_m - n_f)\frac{1}{c_m}\Gamma(c_m\beta)\right), \quad \forall i \in [n_m+1, n]. \end{cases} \tag{D.103}$$

Hence, we can estimate the number of server steps when a task arrive and quits some node i as :

$$\lim_{\iota\to\infty} \iota^{\alpha-1}\, \mathrm{m}_i(\iota) \leq \lim_{\iota\to\infty} \frac{\lambda}{\mu_i}(\iota^{\alpha-1}\mathbb{E}[X_i^\iota] + 1), \tag{D.104}$$

where $\lambda = n_f \mathbb{P}(X_f > 0)\mu_f + (n_m - n_f)\mu_m + (n - n_m)\mu_s$ (because fast nodes have *almost* empty queue X_f in the considered stationary setting).

We will further assume $n_f = \frac{n}{3}$, $n_m = \frac{2n}{3}$, and $p = \frac{1}{n}$. Under these conditions, we have $\Gamma(c_m\beta) = \Gamma(\mathrm{C}(\frac{\mu_m}{\mu_s} - 1)) \simeq 1$. For fast nodes, the delay can be simplified as:

$$\lim_{\iota\to\infty} \mathrm{m}_i(\iota) \leq \frac{\lambda}{\mu_f}. \tag{D.105}$$

For medium nodes, the delay can be simplified as:

$$\lim_{\iota\to\infty} \mathrm{m}_i(\iota) \leq \frac{\lambda}{\mu_m} \frac{1}{\frac{\mu_m}{\mu_s} - 1}. \tag{D.106}$$

For slow nodes, this simplifies as:

$$\lim_{\iota\to\infty} \mathrm{m}_i(\iota) \leq \frac{\lambda}{\mu_s}\left(\frac{3}{n}\mathrm{C} - \frac{1}{\frac{\mu_m}{\mu_s} - 1}\right). \tag{D.107}$$

In the following we consider $n = 9$ clients, split in three clusters of same size: fast nodes with rate $\mu_f = 10$, medium nodes with rate $\mu_m = 1.2$, and slow nodes with rate $\mu_s = 1$. We simulate up to $T = 10^6$ server steps, and plot the distribution of the delays (in number of server steps).

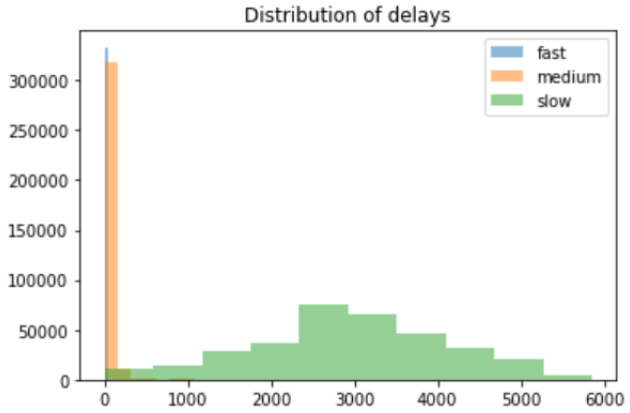
Method	FedBuff	AsyncSGD	`Generalized AsyncSGD`
Accuracy on the CS test set	49.89 ± 0.77	59.09 ± 1.97	66.61 ± 3.26

Table D.1: Performance average (mean ± std) over 10 random seeds for the CIFAR-10 task.

Under this setting, we have $\frac{\iota^{1-\alpha}}{c_m}\Gamma(c_m\beta) = \frac{1}{\frac{\mu_m}{\mu_s}-1} = 5$. Hence we can estimate the following:

$$\begin{cases} \lim_{\iota\to\infty} \mathrm{m}_i(\iota) \leq \frac{n_f\mathbb{P}(X_f>0)\mu_f+(n_m-n_f)\mu_m+(n-n_m)\mu_s}{\mu_f} \simeq 3\mathbb{P}(X_f>0), \quad \forall i \in [1, n_f], \\ \lim_{\iota\to\infty} \mathrm{m}_i(\iota) \leq \frac{n_f\mathbb{P}(X_f>0)\mu_f+(n_m-n_f)\mu_m+(n-n_m)\mu_s}{\mu_m}\frac{1}{\frac{\mu_m}{\mu_s}-1}, \quad \forall i \in [n_f+1, n_m], \\ \lim_{\iota\to\infty} \mathrm{m}_i(\iota) \leq \frac{n_f\mathbb{P}(X_f>0)\mu_f+(n_m-n_f)\mu_m+(n-n_m)\mu_s}{\mu_s}(\frac{3\mathrm{C}}{n}-\frac{1}{\frac{\mu_m}{\mu_s}-1}), \quad \forall i \in [n_m+1, n]. \end{cases} \tag{D.108}$$

The simulation gives $\lambda \simeq 9$, and we recover the theoretical delays: the average delay for fast




```
avg delay fast =  1.3453161915498364
avg delay medium =  55.02906515784922
avg delay slow =  2935.6114257955987
avg queue fast 0.26063426063426065
avg queue medium 6.107726107726108
avg queue slow 326.96497296497296
```

Figure D.5: We assign C = 1000 tasks to a network of $n = 9$ nodes split in 3 clusters.

nodes is close to 1, the average delay for medium node is $55 \simeq 5\frac{\lambda}{\mu_m}$, and the average delay for slow nodes is about $2935 \simeq 325\frac{\lambda}{\mu_s}$.

D.7 Deep learning experiments details

D.7.1 Simulation

We based our simulations mainly on the code developed by J. Nguyen et al. 2022: we assume a server and n clients, each of which initially has a unique split of the training dataset. To adequately capture the time spent on the server side for computations and orchestration of

centralized learning, two quantities are implemented: the server waiting time (the time the server waits between two consecutive calls) and the server interaction time (the time the server takes to send and receive the required data). In all experiments, they are set to 4 and 3, respectively. When a client i receives a new task, we take a new sample from an exponential distribution (with mean $\frac{1}{\mu_i}$, where μ_i is the rate of node i), and stack the gradient computation on top of the client queue.

D.7.2 Implementation

In Section 7.6 we have simulated experiments and run the code for the concurrent approaches AsyncSGD and FedBuff. We also propose an implementation of FedAvg and FAVANO (see Chapter 6). FedAvg is a standard synchronous method. At the beginning of each round, the central node s selects clients uniformly at random and broadcast its current model. Each of these clients take the central server value and then performs exactly K local steps, and then sends the resulting model progress back to the server. The server then computes the average of the s received models and updates its model. In this synchronous structure, the server must wait in each round for the slowest client to complete its update.

AsyncSGD is an asynchronous method that initially randomly selects C clients. Then, a server step is done when a new task is completed and sent back to the server. The server uniformly selects a new client and send a new task. While AsyncSGD was tested on a simple task in Koloskova, Sebastian U Stich, and Jaggi 2022, we have developed a deep learning version of the algorithm (see supplemental material) based on a list of dictionaries (to simulate a network of waiting queues).

For each global step, in FedBuff, the runtime is the sum of the server interaction time and the time spent feeding the buffer of size Z. The waiting time for feeding the buffer depends on the respective local runtimes of the slow and fast clients, as well as on the ratio between slow and fast clients: in the code, we reset a counter at the beginning of each global step and read the runtime when the Z^{th} local update arrives. In AsyncSGD and Generalized AsyncSGD, the runtime is defined by the closed Jackson network properties.

Compression and Federated Learning: an approach to frugal machine learning

Abstract

"Intelligent" devices and tools are gradually becoming the standard, as the implementation of algorithms based on artificial neural networks is experiencing widespread development. Neural networks consist of non-linear machine learning models that manipulate high-dimensional objects and obtain state-of-the-art performances in various areas, such as image recognition, speech recognition, natural language processing, and recommendation systems.

However, training a neural network on a device with lower computing capacity can be challenging, as it can imply cutting back on memory, computing time or power. A natural approach to simplify this training is to use *quantized neural networks*, whose parameters and operations use efficient low-bit primitives. However, optimizing a function over a discrete set in high dimension is complex, and can still be prohibitively expensive in terms of computational power. For this reason, many modern applications use a network of devices to store individual data and share the computational load. A new approach, *federated learning*, considers a distributed environment: Data is stored on devices and a centralized server orchestrates the training process across multiple devices.

In this thesis, we investigate different aspects of (stochastic) optimization with the goal of reducing energy costs for potentially very heterogeneous devices. The first two contributions of this work are dedicated to the case of quantized neural networks. Our first idea is based on an annealing strategy: we formulate the discrete optimization problem as a constrained optimization problem (where the size of the constraint is reduced over iterations). We then focus on a heuristic for training binary deep neural networks. In this particular framework, the parameters of the neural networks can only have two values. The rest of the thesis is about efficient federated learning. Following our contributions developed for training quantized neural network, we integrate them into a federated environment. Then, we propose a novel unbiased compression technique that can be used in any gradient based distributed optimization framework. Our final contributions address the particular case of asynchronous federated learning, where devices have different computational speeds and/or access to bandwidth. We first propose a contribution that reweights the contributions of distributed devices. Then, in our final work, through a detailed queuing dynamics analysis, we propose a significant improvement to the complexity bounds provided in the literature on asynchronous federated learning.

In summary, this thesis presents novel contributions to the field of quantized neural networks and federated learning by addressing critical challenges and providing innovative solutions for efficient and sustainable learning in a distributed and heterogeneous environment. Although the potential benefits are promising, especially in terms of energy savings, caution is needed as a rebound effect could occur.

Keywords: federated learning, quantized neural networks

Laboratoire LISITE
ISEP – Sorbonne Université – – – –

Compression et apprentissage Fédéré : une approche pour l'apprentissage machine frugal

Résumé

Les appareils et outils "intelligents" deviennent progressivement la norme, la mise en œuvre d'algorithmes basés sur des réseaux neuronaux artificiels se développant largement. Les réseaux neuronaux sont des modèles non linéaires d'apprentissage automatique avec de nombreux paramètres qui manipulent des objets de haute dimension et obtiennent des performances de pointe dans divers domaines, tels que la reconnaissance d'images, la reconnaissance vocale, le traitement du langage naturel et les systèmes de recommandation.

Toutefois, l'entraînement d'un réseau neuronal sur un appareil à faible capacité de calcul est difficile en raison de problèmes de mémoire, de temps de calcul ou d'alimentation. Une approche naturelle pour simplifier cet entraînement consiste à utiliser des *réseaux neuronaux quantifiés*, dont les paramètres et les opérations utilisent des primitives efficaces à faible bit. Cependant, l'optimisation d'une fonction sur un ensemble discret en haute dimension est complexe et peut encore s'avérer prohibitive en termes de puissance de calcul. C'est pourquoi de nombreuses applications modernes utilisent un réseau d'appareils pour stocker des données individuelles et partager la charge de calcul. Une nouvelle approche a été proposée, l'*apprentissage fédéré*, qui prend en compte un environnement distribué : les données sont stockées sur des appareils différents et un serveur central orchestre le processus d'apprentissage sur les divers appareils.

Dans cette thèse, nous étudions différents aspects de l'optimisation (stochastique) dans le but de réduire les coûts énergétiques pour des appareils potentiellement très hétérogènes. Les deux premières contributions de ce travail sont consacrées au cas des réseaux neuronaux quantifiés. Notre première idée est basée sur une stratégie de recuit : nous formulons le problème d'optimisation discret comme un problème d'optimisation sous contraintes (où la taille de la contrainte est réduite au fil des itérations). Nous nous sommes ensuite concentrés sur une heuristique pour la formation de réseaux neuronaux profonds binaires. Dans ce cadre particulier, les paramètres des réseaux neuronaux ne peuvent avoir que deux valeurs. Le reste de la thèse s'est concentré sur l'apprentissage fédéré efficace. Suite à nos contributions développées pour l'apprentissage de réseaux neuronaux quantifiés, nous les avons intégrées dans un environnement fédéré. Ensuite, nous avons proposé une nouvelle technique de compression sans biais qui peut être utilisée dans n'importe quel cadre d'optimisation distribuée basé sur le gradient. Nos dernières contributions abordent le cas particulier de l'apprentissage fédéré asynchrone, où les appareils ont des vitesses de calcul et/ou un accès à la bande passante différents. Nous avons d'abord proposé une contribution qui repondère les contributions des dispositifs distribués. Dans notre travail final, à travers une analyse détaillée de la dynamique des files d'attente, nous proposons une amélioration significative des bornes de complexité fournies dans la littérature sur l'apprentissage fédéré asynchrone.

En résumé, cette thèse présente de nouvelles contributions au domaine des réseaux neuronaux quantifiés et de l'apprentissage fédéré en abordant des défis critiques et en fournissant des solutions innovantes pour un apprentissage efficace et durable dans un environnement distribué et hétérogène. Bien que les avantages potentiels soient prometteurs, notamment en termes d'économies d'énergie, il convient d'être prudent car un effet rebond pourrait se produire.

Mots clés : apprentissage fédéré, réseaux de neurones quantifiés