SAPIENZA
UNIVERSITÀ DI ROMA

Sapienza University of Rome

Department of Computer, Control and Management Engineering
PhD in Automatic Control, Bioengineering and Operations Research

Doctoral thesis in Operations Research

Controlled mini-batch algorithms for large-scale optimization in Deep Learning

Supervisor
Prof. Laura Palagi

Candidate
Corrado Coppola
1698064

Academic Year 2023-2024 (XXXVII cycle)

Controlled mini-batch algorithms for large-scale optimization in Deep Learning
Ph.D. thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX.

Author's email: corrado.coppola@uniroma1.it

Abstract

The volume of data processed by machine learning models, particularly deep neural networks (DNNs), is rapidly increasing alongside the dimensions of these models. In 2023, Internet users generated an average of approximately 2.5 quintillion bytes (2.5×10^{18}) of data daily[1]. The well-known large language model, ChatGPT, has been trained on a dataset containing hundreds of billions of web pages[2].

As tasks become more complex and training datasets grow larger, the number of trainable parameters in DNNs has been increasing exponentially. State-of-the-art DNNs can have billions, or even trillions, of trainable parameters, thanks to innovations like self-attention layers [102], diffusion models [46], and the use of over-parameterization to achieve perfect interpolation [38]. Encompassing such tasks as regression and image classification, this thesis is focused on the training of deep neural networks in a supervised context, where the training problem is formulated as the unconstrained minimization of a smooth and possibly non convex objective function with respect to the network weights. Hence, the dimension of the target problem depends both on the dimension of the network and on the amount of the input data available for the training, which makes it computationally impossible to use traditional full-batch optimization methods such as Gradient Descent, L-BFGS, and other Newton's method modifications. This lead to the widespread adoption of *mini-batch* methods, which update the network weights using, at each iteration, only a small subset of the available samples to compute an estimate of the gradient of the objective function. Despite their increased computational efficiency with respect to full-batch algorithms, mini-batch methods require strong assumptions (e.g, convexity, strong convexity, gradient-related search directions) to prove the convergence to a stationary point. Furthermore, adaptive mini-batch methods, which adjust the step size based on the gradient estimate, can suffer from instability in the final phases of training.

Seeking for a balance between theoretical guarantees of convergence and stability and the steadily increasing efficiency requirements to minimize the training time, as well as the carbon footprint of the training process itself [166], in this thesis we propose an approach based on enhancing Incremental Gradient (IG) and Random Reshuffling (RR) algorithms through the use of sufficient decrease conditions and derivative-free extrapolation linesearch procedures. More in detail, we consider an efficient version of the Controlled Mini-batch Algorithm (CMA), firstly proposed in Seccia's PhD Thesis and available in [186]. CMA is a framework encompassing both IG and RR with a sufficient decrease condition on the objective function and the possibility to execute a line-search to ensure the convergence even when no assumptions are made on the search direction or the gradient. We further present the *light* version of CMA, CMA Light, which achieves the same convergence results in the IG framework but much better efficiency performance on state-of-the-art regression and image classification tasks using an estimate of the objective function to check the sufficient decrease. With the further variant Block-Decomposed CMA Light (BD-CMAL), we also investigate the possibility of coupling our mini-batch method with a block decomposition framework, trying to expand further what proposed in [164]. Finally, with Fast-

[1] https://explodingtopics.com/blog/big-data-stats

[2] https://openai.com/index/chatgpt/

Controlled Mini-batch Algorithm (F-CMA), we extend the convergence theory of CMA Light to the case, when the samples are reshuffled at every iteration and we develop a line-search procedure that uses an approximation of the objective function. We test F-CMA on networks with up to 120 millions of trainable parameters against the most commonly used state-of-the-art optimizers for deep learning, and we show a significant advantage both in terms of stability and in terms of generalization capability of the trained models.

Keywords: *Deep Learning, Large-scale Optimization, Smooth Optimization.*

Acknowledgements

The development of this thesis, as well as all the contributions of my three-years doctoral research, has been made possible by the continuous and invaluable support of my PhD Supervisor Prof. Laura Palagi, who encouraged me both on the professional and personal side through any hardships, encountering my needs for a solid guidance, orienting my activities toward the most proficuous directions for me and helping clarifying my interests and talents. Besides Her, I want to thank the professional figures that enhanced and facilitated my research activity. In particular, Prof. Giampaolo Liuzzi and Prof. Irene Amerini, whom I had the pleasure and the honor to have as co-authors of different papers, and Prof. Coralia Cartis, who hosted me as a visiting researcher at the University of Oxford and assisted me in getting familiar with the complexity analysis, which resulted in a strong contribution to this thesis. I am also grateful to my co-authors and colleagues Dr. Marco Boresta, Lorenzo Papa and Dr. Ruggiero Seccia and to all the people that contributed to a creative and friendly work environment, in a particular way to my MS thesis co-advisor Dr. Marta Monaci and to Christian Piermarini. Finally, I want to remark that the content of this thesis has been significantly enriched and enhanced by the research I carried out in tutoring two brilliant MS thesis students, Lorenzo Ciarpaglini and Ilaria Ciocci, to whom I extend my admiration and gratitude.

Contents

List of Figures

List of Tables

Nomenclature

General notations

$\|\cdot\|$	ℓ_2-norm (or Euclidean norm)
$\mathbf{1}_N$	Unitary vector of N elements
$\mathcal{L}(x, y, \varphi(x; w))$	General notation for the loss function
$\mathcal{T}$	Training set
$\varphi(x; w)$	General notation for the output of a deep network
$\widetilde{w}_i^k$	Vector of trainable parameters at step i of epoch k
d	Number of trainable parameters
d^k	Search direction at step or epoch k
$f(w)$	Objective function
L	Lipschitz constant of the $\nabla f(w)$
L_f	Lipschitz constant of the $f(w)$
n	Number of input features
P	Number of samples in the data set
w	Vector of trainable parameters
w^k	Vector of trainable parameters at the beginning of epoch k
x	Vector of the P input samples
y	Vector of the P input labels

Acronyms

CEL Cross-Entropy Loss

CMA Controlled Mini-batch Algorithm

CNN Convolutional Neural Network

DL Deep Learning

EDFL Extrapolation Derivatife-Free Line-search

FNN Feed-forward Neural Network

LS Line-Search

ML Machine Learning

MSE Mean squared error

SVM Support Vector Machine

VC Vapnik-Chervonenkis

Chapter 1

Introduction

1.1 Research motivation

Big Data and *Deep Learning* represent significant challenges in the modern era of artificial intelligence. The demand for AI-based systems is steadily growing across an extensive range of real-world applications, from image and video recognition to recommendation engines, user profiling, predictive analytics, and the recently introduced large language models. Each of these applications relies on vast datasets and increasingly complex, resource-intensive models to deliver accurate and responsive outputs. Efficiency is essential in modern AI-based systems [193], impacting both economic returns and environmental sustainability [166]. This need for efficiency spans both across the training phase, where models learn tasks, and the inference phase, where models are deployed (for instance on the Web) to handle large-scale data and to process external users' queries. The primary driver of efficiency is the computational cost, typically measured in GPU-hours[1], which directly affects both the operational expenses (e.g., energy, cooling systems) and the carbon footprint associated with model training and deployment. These computational costs are not just financial. They also contribute significantly to environmental costs, such as energy consumption and greenhouse gas emissions. Moreover, they also influence market-related aspects, including the time needed to train a model and the latency experienced by users during inference. As explained in detail in the recent survey [193], there are different and often complementary approaches to increase the efficiency of AI-based system. Some of them are still an open research field for the Computer Science community, such as the regulation of the numerical precision of the system (weights quantization) [124, 178] or the parallelization of the data sampling process [2, 123, 162]. Some others are strictly related to the architectural choices in the model engineering and have been successfully proposed and adopted by the Deep Learning research community since 2010s, like convolutional neural networks [11], self-attention mechanisms [204] or diffusion models [46].

However, when the efficiency is particularly critical in the training phase, that is in supervised machine learning, where models are trained on vast datasets of input-output pairs, the optimization strategy becomes the game changer [163]. In supervised learning, training an industrial-scale

[1] Graphics Processing Units, hardware devices built to optimize the efficiency of parallelized mathematical operations

model is a massive challenge and is out of the possibilities of most research institutions, but at the same time, as we will see, it "simply" requires solving a non-convex unconstrained minimization problem with smooth objective function. Using fast, stable and reliable optimization algorithms to solve the mathematical problem behind the training of a supervised machine is the key element of *optimization-driven efficiency*.

The aim of this thesis is to propose the Controlled Mini-batch Algorithm (CMA) framework, a novel approach to the problem of maximizing the efficiency of training algorithms for deep learning models. In our opinion, the development of stable, efficient and reliable optimization algorithms with strong mathematical properties is an important efficiency driver in deep learning. For instance, training large models such as GPT-3 [171] or PaLM [41] can cost millions of US dollars. The expense arises from the number of trainable parameters, i.e., the number of variables of our target problem (e.g., GPT-3 has 175 billions of parameters), which results in extensive computational power required for training, often involving GPUs and TPUs[2] for weeks or even months. The computational overhead of the training process can be influenced also by the inherent sequential nature of some models, like the diffusion models [46], that are not easily parallelizable. Cloud services charge AI-system developers based both on compute hours and data storage, making the cost prohibitive for many research institutions or smaller companies. These costs could easily exceed $ 5 million for very large models, depending on the infrastructure used[3]. OpenAI's GPT-3 reportedly cost about $ 12 million to train from scratch[4], while the recently released GPT-4 cost $ 64 million [5].

The most commonly used optimizers, the first-order mini-batch algorithms, are based on the well-known gradient descent method, which updates the variables of the objective function moving alongside the anti-gradient with a properly chosen scalar step-size. As we will see in detail in the following chapters, mini-batch methods uses only a small subset (a.k.a. mini-batch) of the samples available in the training dataset to compute an estimation of the anti-gradient and to perform an update step. Hence, they are much more efficient than traditional first-order methods, which rely on the exact gradient to update the variables. However, mini-batch methods also have weaker convergence properties, often require strong assumptions on the problem and are generally less studied from a theoretical perspective. Furthermore, all the mini-batch methods that have gained increasing popularity since the end of the 1990s [16, 194], come with efficiency limitations.

On the one hand, the non-adaptive methods (like the classical SGD [176]) do not require a critical amount of data storage and can be easily parallelized as each update only depends on the gradient with respect to the samples in the current batch. This simplicity also facilitates parallelization since the step size, a scalar value, can remain fixed or follow a predefined schedule, independent of previous updates. However, tuning the initial step size is crucial yet computationally intensive, as finding an optimal setting involves an exhaustive pre-training phase. This pre-training phase,

[2] Tensor Processing Units, hardware devices built to optimize operations involving tensor products

[3] https://www.cudocompute.com/blog/what-is-the-cost-of-training-large-language-models

[4] https://deeperinsights.com/ai-blog/the-costs-and-complexities-of-training-large-language-models

[5] https://medium.com/@rohanbalkondekar/the-full-training-run-of-gpt-5-has-gone-live-cb06a750a35c

known as hyper-parameters setting [12], involves procedures such as K-Fold cross validation and grid search, which demand substantial GPU resources, making it a significant driver of the overall computational cost.
On the other hand, adaptive methods (such as Adagrad [58] and Adam [103]) have been progressively adopted by deep learning practitioners since the 2010s, as they update the step-size dynamically during the training, using stored information from past iterates. This approach significantly reduces the need for an expensive pre-training phase and the tuning of the initial step-size is not as crucial as for non-adaptive methods. However, the improved efficiency in terms of GPU-hours is counterbalanced by the increased memory storage needed to keep the information (e.g., the mini-batch gradients) from past iterates. While being a merely technical issue, this has an impact on the cost of the training. Furthermore, as we will see in detail in chapter 3, adaptive methods are still poorly understood from a theoretical perspective. Most of the existing adaptive methods that are implemented in state-of-the-art deep learning libraries have never been investigated systematically. And even the few algorithms, like the aforementioned Adam and Adagrad, for which a convergence theory has been developed, rely on strong assumptions (e.g., convex objective function, correct descent search direction), often not satisfied in deep learning applications.
More recently, the idea of using line-search techniques to correct the step-size after or during an epoch of non-adaptive method [80, 133, 139, 205] gained growing interest as a possible solution to have the computational advantages of non-adaptive methods without the need for an expensive pre-training phase. However, to our knowledge, all the existing methods still rely on assumptions that cannot be considered standard in most real-world applications. One of the most frequent assumptions in the non-convex setting, for instance, is that the search direction (i.e., the estimated gradient) is a sufficiently "good" descent direction. While this can be true in certain contexts, modern deep learning models are often trained on upcoming data that can contain corrupted and/or inexact information. In this setting, commonly known as *byzantine* learning [117, 118, 181], model weights can be updated using completely wrong estimated gradients for a certain number of iterations. The CMA framework, introduced in [185, 186] and proposed in this thesis, is a highly reliable and efficient solution to this problem, allowing for a straightforward convergence proof to a stationary point without any assumptions on the search direction. The underlying idea of CMA is to check, after each iteration of non-adaptive method, a certain condition which we name *safeguard rule*. If the safeguard rule is not satisfied, then the step-size is adjusted using an extrapolation derivative-free line-search according to the specific algorithm's mechanics. In the CMA framework, to deal with learning in presence of error (e.g., byzantine learning), we admit the possibility to drive to zero the step-size for a given iteration, if the line-search fails and/or specific conditions on the current search direction are not satisfied.
This thesis is structured to reflect the progressive development of our research on CMA. Step by step, we present our efforts to refine the safeguard rule and line-search procedure mechanics, improving algorithmic efficiency and expanding theoretical results.

1.2 Outline

This thesis is structured as follows.
In chapter 2 we provide the reader with the preliminary concepts to understand the target problem and the existing methods. More in detail, we firstly recall how, according to the Vapnik-Chervonenkis theory, the problem of training a supervised machine consists in solving the unconstrained minimization of the sum of P possibly non-convex smooth functions. This formulation places our target problem in the field of finite-sum smooth unconstrained optimization, bridging the world of machine learning and operations research. Then, we provide a general description of mini-batch methods and we describe the mechanics and the variables update rules of the most commonly used algorithms to train a deep neural network. Finally, we derive a closed expression for our target problem in different scenarios, showing how the objective function strongly depends on the type of neural network we want to train. In particular, we focus on feed-forward neural networks, convolutional neural networks and transformer-based architectures. A significant part of this chapter (sections 2.2 and 2.3) have has been written using the article [44], recently appeared on "TOP - Transactions on Operations Research".
In chapter 3 we review the main existing literature in the field of large-scale optimization for deep learning. We focus on the convergence theory of the existing methods in the non-convex setting. We examine the trade-offs between convergence guarantees, worst-case complexity bounds and assumptions required to prove a given result. In this chapter, we dedicate a section to the analysis of the stochastic methods, which are still the most widely adopted by deep learning practitioners and another one to the review of the existing deterministic convergence results on mini-batch methods. After having reviewed the most widespread traditional mini-batch methods, distinguishing between adaptive and non-adaptive ones, according to the step-size update rule, we recall the most significant existing works that, as this thesis, are based on line-search techniques.
In chapter 4 we present the Controlled Mini-batch Algorithm framework, the main work this doctoral research, which resulted in the article [186], currently at the second step of a peer-reviewing process at "Computational Optimization and Applications". We describe the novel CMA approach, based on the Incremental Gradient (IG) or Random Reshuffling (RR) method, enhanced by using an extrapolation derivative-free line-search procedure to dynamically tune the step-size. Two algorithms are presented, jointly with a fully deterministic convergence theory. We show that these algorithms solve our target problem to stationarity assuming only the coercivity and the Lipschitz-smoothness of the objective function. We extensively test the two algorithms on deep learning tasks, using feed-forward neural networks with up to 1 million trainable parameters and we prove that the CMA approach strongly outperforms simple incremental gradient and the full-batch method L-BFGS. However, we observe that both the algorithms require at least one evaluation of the objective function per iteration, which can result in a extremely heavy computational overload on large-scale problems. In fact, we test the algorithms against the best-performing adaptive optimizers Adam and Adagrad, finding that CMA framework can be outperformed because of its bottleneck - the need for at least one function evaluation per iteration. Eventually, we include in the chapter the worst-case

complexity analysis of a variant of the CMA method, where the step-size is bounded away from zero and a minimal improvement in the value of the objective function is required at each iteration. This analysis is based on the research work carried out at the Mathematical Institute of the University of Oxford under the supervision of Prof. Coralia Cartis.
In chapter 5 we present CMA Light, the evolution of the CMA framework which removes the main bottleneck. The chapter is based on the pre-print [43], which is currently about to be submitted. CMA Light adopts the same approach of CMA but requires the value of the objective function only to perform the line-search procedure, which activates seldomly in most computational applications. However, thanks to an additional hypothesis on the individual coercivity of each term of the objective function, has the same theoretical properties of CMA and converges to a stationary point, if the data batches are sampled according to a deterministic pre-fixed distribution. We conduct massive computational experiments, both on regression tasks (on feed-forward neural networks with up to 7 millions of trainable parameters) and on image classification tasks (on convolutional neural networks with up to 50 millions of trainable variables). We show that CMA Light significantly outperforms all its competitors in almost all the experiments and that it also achieves better and faster generalization performance. Eventually, we present BD-CMAL, a block-decomposed version of CMA Light developed in technical report [42] jointly with the MS student Ilaria Ciocci.
In chapter 6 we present the enhanced version of CMA Light, Fast-CMA, which removes the need for the objective function evaluation even in the line-search and, under the gradient growth condition, can be proved to converge to a stationary point even in case of random reshuffling of the samples. After formalizing and proving the convergence properties of Fast-CMA, we present other computational experiments on image classification tasks, using vision transformers and residual networks with up to 150 millions of trainable parameters. The computational experiments on F-CMA, which required thousands of GPU-hours have been made possible by the strong collaboration with ALCOR Lab. F-CMA strongly outperforms nine state-of-the-art deep learning optimizers in any setting, both in terms of training efficiency and in terms of generalization performance achieved.
Finally, in chapter 7, we discuss the conclusion of this doctoral thesis and we present the early results of a forthcoming work, which is, in our opinion, the natural prosecution of this research. In particular, we discuss the applicability of the CMA-framework in generative tasks (GenAI). We present an heuristic line-search procedure, based on the first-order Armijo conditions [6], which turns F-CMA into a completely objective function-free [73, 74] method. We discuss the early results obtained on training a diffusion model architecture with approximately 170 millions trainable parameters to generate images of handwritten digits.

1.3 Dissemination

Our main contributions to the research in the field of optimization methods for deep learning are in the following articles and pre-prints, that constitute the backbone of this thesis:

- C. Coppola, L. Papa, M. Boresta, I. Amerini, L Palagi, *Tuning parameters of deep neu-*

ral network training algorithms pays off: a computational study, TOP - Transactions on Operations Research. DOI: 10.1007/s11750-024-00683-x

- R. Seccia, C. Coppola, G. Liuzzi, L.Palagi *Convergence of ease-controlled Random Reshuffling gradient Algorithms under Lipschitz smoothness*, submitted at COAP - Computational Optimization and Applications, currently at the second step of peer-reviewing process, arxiv pre-print: arxiv:2212.01848

- C. Coppola, G. Liuzzi, L.Palagi, *CMA Light: a novel Minibatch Algorithm for large-scale non convex finite sum optimization*, arxiv pre-print: arXiv:2307.15775

- C. Coppola, L. Papa, I. Amerini, L. Palagi, *Beyond adaptive gradient: Fast-Controlled Mini-batch Algorithm for large-scale optimization*, submitted at "Expert Systems with Applications", arxiv pre-print: arxiv:2411.15795

- I. Ciocci, C. Coppola, L. Palagi, L. Papa, *Block Layer decomposition applied to a watchdog controlled minibatch algorithm*, Technical Report at DIAG - Sapienza University of Rome - 2024-TR-n02

The research work on the CMA, at different steps of its development, has been exposed in the following venues:

- the 19th EUROPT Workshop on Advances in Continuous Optimization, held in Lisbon (Portugal) in July 2022;

- the Optimization2023 Mathematical Optimization conference held in Aveiro (Portugal) in July 2023;

- the 20th EUROPT Workshop on Advances in Continuous Optimization, held in Budapest (Hungary) in August 2023;

- the "2024 Numerical Analysis Seminaries" for the Trinity Term at the Mathematical Institute of the University of Oxford, in February 2024, Oxford (UK);

- the 25th International Symposium on Mathematical Programming, held in Montreal (Canada) in July 2024;

- the "Operations Research Seminaries" at the FCT-NOVA in Lisbon (Portugal) in October 2024

1.4 Reproducibility

The algorithms presented in this thesis have all been implemented in Python, using the standard machine learning library `Pytorch 2.0`[6].
The implementation details and the hardware used for the different tests is specified, case by case, in the corresponding sections.

[6] https://pytorch.org/

All the computational results that are presented in this thesis are fully reproducible and the source code is available in the following repositories:

- Chapter 4, all sections with CMA, NMCMA, CMA-FD at https://github.com/corradocoppola97/ControlledMinibatchAlgorithm
- Chapter 5, CMA Light, results in section 5.4 at https://github.com/corradocoppola97/CMA_Light
- Chapter 5, BD-CMAL, results in section 5.5 at https://github.com/corradocoppola97/BLD_CMA_L
- Chapter 6, Fast-CMA available at https://github.com/corradocoppola97/Fast_CMA
- Chapter 7, early results on the case-study described in section 7.3 available at https://github.com/corradocoppola97/EAL_Generation

In the repositories web-page further instructions on how to use and/or replicate the code are reported.

Chapter 2

Preliminaries

Training a deep neural network in a supervised setting involves solving the unconstrained minimization problem of a smooth, and often non-convex, objective function. The primary challenge in solving this problem lies in the function's dimensionality, which depends on both the number of network parameters and the size of the training dataset. This makes training deep neural networks a complex task in the field of large-scale optimization. As the demand for more powerful and accurate models grows, the size of these networks is increasing exponentially. This trend is driven by the need to handle complex tasks such as natural language processing [40], computer vision [207], and large-scale recommendation systems [62]. Additionally, reducing training time is crucial to accelerate research and deployment cycles, keep computational costs manageable, and minimize energy consumption [28, 146, 187]. As a result, optimizing deep neural network training remains a significant and challenging problem in modern machine learning. In this chapter, we introduce the foundational concepts necessary to further present the novel algorithms designed to address this challenge.

The chapter is structured as follows. In Section 2.1, we review the fundamentals of supervised learning theory, focusing specifically on the minimization of expected risk and Vapnik-Chervonenkis theory. We then demonstrate how training a supervised learning model is inherently structured as a finite-sum unconstrained minimization problem. In Section 2.2, we provide a concise overview of the most commonly used optimization methods for deep learning. Since these methods are currently considered state-of-the-art, they will serve as our main benchmark in the subsequent chapters of this thesis. Finally, in Section 2.3, we review the primary types of deep neural networks that we will use in our experimental test-beds in the following chapters. Specifically, we demonstrate how to mathematically derive the output of Feed-forward Neural Networks (FNNs), Convolutional Neural Networks (CNNs), and Transformers and how this influences the mathematical structure of the training problem.

This chapter, in particular sections 2.2, 2.3.1, and 2.3.2, was developed using material from the article by Coppola C., Papa L., Boresta M., Amerini I., and Palagi L. [44], which has been published on TOP - Transactions on Operations Research on Sep 17th, 2024.

2.1 Training a neural network: a mathematical perspective on supervised learning

In order to understand why optimization theory plays a crucial role [163] in supervised machine learning we firstly need to recall how the training of a generic *machine* can be always be expressed in the form of an optimization problem. In supervised learning, regardless the specific tasks discussed further, we have a training set of points and labels $\mathcal{T} = \{(x_i, y_i), i = 1, \dots, P\}$ that are sampled from an unknown probability distribution $\mathcal{P}(x, y)$. The points $x_i \in \Phi \subseteq \mathbb{R}^d$ are vectors representing any input information to the machine, such as physical dimensions, pixels of an image or sequences of numbers representing words and/or any other kind of data. The dimension d will be further considered as the number of *input features*. The labels $y_i \in \Psi \subseteq \mathbb{R}^m$, also named *output* or *ground truth*. Ψ can be a discrete set in $\{0, 1\}^m$ in case of a classification task, or a continuous set in $\mathbb{R}^m$ in case of a regression task. The goal of a learning machine is to select a function $\varphi(\cdot; w) : \Phi \to \Psi$, parametrized in $w \in \mathbb{R}^n$ and belonging to a class of functions $\mathcal{F}$, that, for a given input x, returns an accurate prediction $\hat{y} = \varphi(x; w)$ of the corresponding output y. The criterion that driving the choice of this function is a measure of dissimilarity between the prediction returned by $\varphi(\cdot; w)$ and the ground truth y. This measure of dissimilarity $\mathcal{L}(x, y, \varphi(x; w))$ is usually named *loss function*. In all the cases that will be further analyzed, the loss function is a Lipschitz-continuous and differentiable measure of dissimilarity: the mean squared error (MSE) or the cross entropy loss (CEL).
In the statistical learning theory, formalized in the past 30 years in [30, 45, 201–203], the problem of training a machine using a given loss function is expressed in the form of the unconstrained minimization of the *expected risk*, which is the expected value of the loss function in the form:

$$R(w) = \mathbb{E}\left[\mathcal{L}(x, y, \varphi(x; w))\right] = \int \mathcal{L}(x, y, \varphi(x; w)) d\mathcal{P}(x, y) \tag{2.1}$$

However, the problem:

$$\min_{w \in \mathbb{R}^n} R(w) \tag{2.2}$$

is intractable, as it depends both on the unknown data distribution $\mathcal{P}$ and on the inherent *capacity* of the learned function φ to approximate the output, given the input. Conversely, the only available computable measure of dissimilarity is the *empirical risk*:

$$R_{emp}(w) = \frac{1}{P} \sum_{i=1}^{P} \mathcal{L}(x_i, y_i, \varphi(x_i; w)) \tag{2.3}$$

where $\mathcal{L}(x_i, y_i, \varphi(x_i; w))$ is the loss function computed between a single predicted output and its corresponding correct label in the training set $\mathcal{T}$. In the context of deep learning, the empirical risk is commonly referred to as *training loss*.
In 1990s Vladimir Vapnik, Aleksej Chervonenkis, and Corinna Cortes in [45, 201, 203] proved a revolutionary result, on which the current supervised learning theory (thus, including deep learning) is based. They proved that, given a loss function belonging to a class of functions $\mathcal{F}$,

the expected risk is bounded above in probability by the sum of the empirical risk and a non negative term called Vapnik-Chervonenkis (VC) confidence. The result states that:

$$\Pr\left(R(w) \leq R_{emp}(w) + \sqrt{\frac{h\log\left(\frac{2P}{h}+1\right) - \log\left(\frac{\eta}{4}\right)}{P}}\right) = 1-\eta \tag{2.4}$$

where P is the cardinality of the training set, $\eta \in (0,1)$ is a given level of confidence, and h is the VC dimension. The VC dimension measures the capacity of a given class of function of *shattering* input points. More formally, h is the maximum number of input points $(x_1,\dots,x_h)$, whose corresponding outputs can be correctly predicted by the function $\varphi \in \mathcal{F}$ for *any* possible labeling of $(x_1,\dots,x_h)$.
While we refer the reader to [201] for a more detailed analysis of the result stated in (2.4), we underline that there are two main approach to solve (2.2):

- **Minimize the VC dimension**. This is done in Support Vector Machines (SVM), that are not analyzed in this work and for which we refer the reader to [30, 163].
- **Minimize the empirical risk**. This approach is called *empirical risk minimization* (ERM) and is the starting point of deep supervised learning.

From now on, we will focus only on the ERM, when the learning machine is a deep neural network and the problem can be thus be stated as follows:

$$\min_{w\in\mathbb{R}^n} \sum_{i=1}^{P} \mathcal{L}(y_i, \varphi(x_i;w)) \tag{2.5}$$

where:

- $\varphi(\cdot;w)$ is a deep neural network parametrized in w, i.e., w is the vector of the trainable variables;
- (x_i, y_i) are the input-label pairs in the training set $\mathcal{T}$ for $i = 1,\dots,P$;
- L is the loss function to be minimized.

2.1.1 The ERM in deep learning: the main assumptions

As stated, the problem (2.5), even if theoretically tractable, is too general and reflects an amount of practical applications that go beyond deep learning theory. Deep learning is only one of the possible applications of the ERM, alongside with decision trees [34, 66, 132], k-NN [61], boosting methods [67, 69], and Bayesian learning [19, 156].
Deep neural networks (DNNs) are composed of interconnected nodes, or neurons, arranged in a series of layers: an input layer, several hidden layers, and an output layer. Each neuron processes input data and passes it through an activation function to produce an output. The depth of the network, referring to the number of hidden layers, allows the model to learn hierarchical representations of data, which is particularly effective for tasks such as image and speech recognition, natural language processing, and complex pattern recognition. The training of these networks

involves adjusting the weights associated with the connections between neurons to minimize the empirical risk, which is typically done using gradient-based optimization techniques. Hence, the underlying optimization problem in deep learning can vary significantly, depending on the properties of the network itself, i.e., the *architecture*. For this purpose, the architectures discussed in this thesis will be presented in the specific Section ...
However, regardless any architectural difference, there are some general hypothesis we assume on the neural network $\varphi(\cdot; w)$.

Assumption 2.1.1 (Smoothness)**.** *The neural network learning function φ is continuously differentiable and its gradient is Lipschitz-continuous, i.e.,*

$$\|\nabla\varphi(w_1) - \nabla\varphi(w_2)\| \leq L\|w_1 - w_2\|$$

for all $w_1, w_2 \in \mathbb{R}^n$

Assumption 2.1.2 (Coercivity)**.** *The neural network learning function φ is coercive, i.e., all the level sets $LS_\varphi(\bar{w}) = \{w \in \mathbb{R}^n : \varphi(w) \leq \varphi(\bar{w})\}$ are compact for any $\bar{w} \in \mathbb{R}^n$.*

These assumptions, for a proper choice of the loss function $\mathcal{L}$, allow the use of gradient-based optimization to solve the problem (2.5). In particular, we introduce the following assumptions on the loss function.

Assumption 2.1.3 (Smoothness)**.** *The loss function $\mathcal{L}$ is continuously differentiable with respect to the network weights w and its gradient is Lipschitz-continuous, i.e.,*

$$\|\nabla\mathcal{L}(w_1) - \nabla\mathcal{L}(w_2)\| \leq L\|w_1 - w_2\|$$

for all $w_1, w_2 \in \mathbb{R}^n$

Assumption 2.1.4 (Coercivity)**.** *The loss function $\mathcal{L}$ is coercive, i.e., all the level sets $LS_\mathcal{L}(\bar{w}) = \{w \in \mathbb{R}^n : \mathcal{L}(w) \leq \mathcal{L}(\bar{w})\}$ are compact for any $\bar{w} \in \mathbb{R}^n$.*

We remark that, since the loss function $\mathcal{L}$ has a direct dependence on the output of the neural network φ, Assumptions 2.1.3 and 2.1.4 cannot hold if conditions of Assumptions 2.1.1 and 2.1.2 are not satisfied.
Therefore, under the stated assumptions, the problem of training a deep neural network using ERM can be expressed as the unconstrained minimization of the sum of P smooth, coercive, but possibly non-convex functions:

$$\min_{w\in\mathbb{R}^n} \sum_{i=1}^{P} f_i(w) \tag{2.6}$$

Here, to avoid any ambiguity, we define $f_i(w) = \mathcal{L}(y_i, \varphi(x_i; w))$. In other words, the i-th function in the objective represents the value of the loss function between the predicted output for the i-th sample of the training set and its true value y_i.

2.2 Optimization methods for Deep Learning

As mentioned above, under the assumptions we have introduced on the learning machine function (i.e., the neural network output), training a deep neural network is achieved by solving problem (2.6). Assumption 2.1.3 makes suitable the use of gradient-based methods to solve this problem, while Assumption 2.1.4 ensures the existence of at least one optimal solution.
In this section, we firstly explain why traditional gradient-based optimization is not typically used in DL practice. Then, we briefly the most commonly used optimization algorithms for training deep neural networks.
For the detailed analysis of the convergence properties of the state-of-the-art algorithms that are reviewed in this chapter, we refer the reader to Chapter 3.

2.2.1 Full-batch vs mini-batch in Deep Learning: the issue of problem dimension

The problem (2.6) is an unconstrained minimization problem with coercive and smooth objective. Therefore, it is theoretically possible to solve it using any method that exploits the gradient ∇f. The most straightforward approach is the Gradient Descent (GD) method, based on the iteration:

$$w^{k+1} = w^k - \alpha^k \nabla f(w^k)$$

which can convergence to a stationary point for $f(w)$ with a sub-linear convergence rate. While for a comprehensive analysis of the optimization methods for nonlinear smooth optimization we refer the reader to [79], we remark that they are already well-studied in the literature and many of them are proved to achieve global convergence (i.e., regardless the starting point) to a local solution of problem (2.6). A prominent example, is the limited-memory modification of Newton's algorithm, the Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) [126, 160], which is based on the following updating scheme:

$$w^{k+1} = w^k - \alpha^k H^k \nabla f(w^k)$$

where H^k is an approximation of the Hessian matrix of the objective function. L-BFGS is proved to achieve global convergence to local minima of $f(w)$ with a super-linear convergence rate and this property made it attractive also for deep learning [114, 142]. The use of L-BFGS for deep learning, as well as of other methods exploiting the entire gradient of $f(w)$, has been discussed also in [31, 44]. Computational experience shows that, in spite of their strong convergence properties, these methods are not scalable. When the dimension of problem (2.6) grows, computing the entire gradient of $f(w)$ requires a huge amount of time, even on modern hardware. More specifically, the dimension of (2.6) depends on two factor:

Number of parameters n. This number has been exponentially growing in the past years (see Figure 2.1) following both the development of novel attention-based architectures

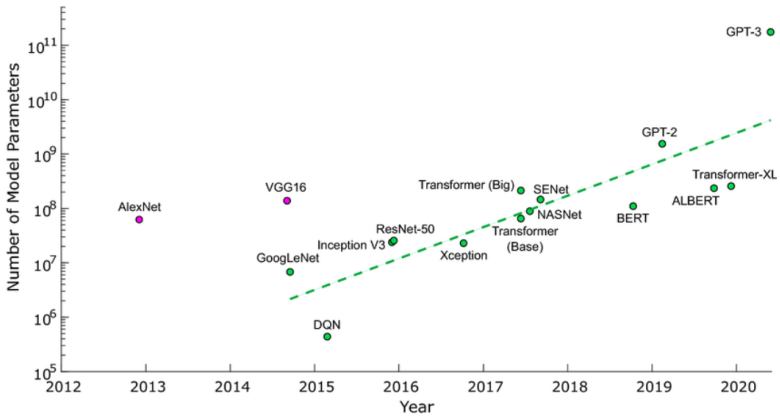


Figure 2.1: Number of trainable parameters of state-of-the-art networks in the past decade. The image has been taken from [13].

[52, 102, 125, 171, 204] (see also Section 2.3.3) and the more frequent adoption of over-parametrization to achieve perfect interpolation [5, 29, 38, 210].

Number of samples P. In the era of *Big Data*, the amount of data available for training deep learning models is continually increasing [175, 199]. This growth in data volume leads to a proportional increase in the computational burden associated with evaluating the gradient.

As a result, the traditional full-batch gradient descent becomes impractical for large-scale problems due to the enormous computational resources and time required. Notice that computing $\nabla f(w)$ implies computing the partial derivatives:

$$\frac{\partial f_i}{\partial w_j}, i = 1, \dots, P \quad j = 1, \dots, n$$

where both n and P can be in the order of billions, i.e., $nP \sim 10^{18}$.
Therefore, approximating the gradient by using only a small subset of the data at each iteration is generally acknowledged as a better approach for the training of deep neural networks. This approach is based on optimization algorithms, named *mini-batch methods*, that strike a balance between computational feasibility and the accuracy of the gradient approximation, enabling the handling of increasingly large datasets in deep learning applications.
A general mini-batch method is based on the following updating scheme:

$$w^{k+1} = w^k - \alpha^k d^k$$

where α^k is the step-size at iteration k and d^k is an estimate of the gradient $\nabla f(w^k)$ obtained using only a subset (i.e., a batch) of samples in the training set.

2.2.2 SGD and its variants

Stochastic Gradient Descent (SGD) is the first proper mini-batch method. It was proposed in 1951, in [176], and is universally acknowledged as the basic algorithm to perform the mini-

mization of the objective function using a search direction d^k which is random estimate of its gradient. More formally, in SGD the search direction d^k is the gradient of the objective function computed using only one or few samples in the training set. This results in the following expression

$$d^k = \frac{1}{|\mathcal{B}_k|} \sum_{i \in \mathcal{B}_k} \nabla f_i(w^k) \text{ for some } \mathcal{B}_k \subset \{1, \dots, P\} \tag{2.7}$$

The updating rule is given by

$$w^{k+1} = w^k - \alpha^k d^k$$

This rule can be modified to improve the stability of the algorithm by adding a momentum term (which depends on the hyper-parameter β), which determines the contribution of the past search directions to the current update. This variant is called SGD with momentum and the basic iteration is:

$$w^{k+1} = w^k - \alpha^k d^k + \beta(w^k - w^{k-1}) \tag{2.8}$$

Another commonly used SGD modification involves the use of the Nesterov acceleration step jointly with the momentum term [197]. The acceleration step is an extrapolation procedure along the difference between the two past iterations to enhance the convergence speed of the algorithm. The update rule becomes:

$$w^{k+1} = z^k - \alpha_k \frac{1}{|\mathcal{B}_k|} \sum_{i \in \mathcal{B}_k} \nabla f_i(z^k) \tag{2.9}$$

where $z^k = w^k + \beta(w^k - w^{k-1})$.
There are also other SGD variants, still based on the estimate (2.7), but they differ for how the samples are selected.

Random Reshuffling (RR). The entire dataset is shuffled randomly at the beginning of each epoch. The algorithm then processes the samples in this new random order.

Incremental Gradient (IG). The samples are processed in a fixed, cyclic order without reshuffling. This means that the algorithm goes through the dataset in the same order repeatedly.

Incremental Aggregated Gradient (IAG). It maintains an aggregated estimate of the gradient that is updated incrementally. Instead of using only the current sample to compute the gradient, IAG updates a stored average of the gradients from all previous iterations.

The convergence theory of SGD and its variants has been deeply studied in the literature for the past decades and it is still an open field. The main existing results, crucial to the aim of this thesis, will be discussed in detail in chapter 3 .

2.2.3 Adagrad

Adagrad [58] is the first mini-batch method that, as most of the following ones, makes use of adaptive learning rates to discriminate more informative and rare features. The general update rule of $w^k \in \mathbf{R}^h$ involves complex matrix operations, for which we need to introduce some other notation. At iteration k we introduce the cumulative vector

$$G_k = \sum_{t=0}^{k-1} (d_t \odot d_t)$$

where $d_t = \sum_{i \in \mathcal{B}_t} \nabla f_i(w^t)$ as in (2.7).
Given $\varepsilon > 0$ and the identity matrix I, we define the following matrix:

$$H_k(\varepsilon) = [I\varepsilon + \text{diag}(G_k)]^{\frac{1}{2}}$$

where $\text{diag}(v)$, with $v \in \mathbf{R}^h$, denotes the diagonal $h \times h$ matrix with elements v_i on the diagonal. Thus, the updating rule resulting after the minimization of a specific proximal function (see [58]) is the following:

$$w^{k+1} = w^k - \eta H_k(\varepsilon)^{-1} d_k \tag{2.10}$$

There is a vast literature on the convergence results concerning Adagrad, which will be discussed in the follwing chapter. However, Adagrad requires in any cases strong assumptions on the search direction and/or on the objective function (e.g., strong convexity) to prove the convergence.

2.2.4 Adam

Adam [104], after Adagrad, is one of the first SGD extensions, where the gradient estimate is enhanced with the use of an exponential moving average according to two coefficients: $\beta_i \in (0, 1)$ with $i = 1, 2$. The index $i \in \{1, 2\}$ is referred to as the moment of the stochastic gradient, i.e., the first moment (expected value) and the second moment (non-centred variance). Being d^k the same mini-batch approximation used in (2.7), we define the following first and second-moment estimators at iterate k:

$$\mathbb{E}\left[\nabla f(w^k)\right] \sim m^k = (1 - \beta_1) \sum_{i=1}^{k} \beta_1^{k-i} d_i \tag{2.11}$$

$$\mathbb{E}^2\left[\nabla f(w^k)\right] \sim v^k = (1 - \beta_2) \sum_{i=1}^{k} \beta_2^{k-i} (d_i \odot d_i) \tag{2.12}$$

where $\odot$ is the Hadamard component-wise product among vectors.
Given the following matrix:

$$\widetilde{V}_k(\varepsilon) = \frac{1}{1 - \beta_2^k} [I\varepsilon + \text{diag}(v_k)]^{\frac{1}{2}}$$

where $\text{diag}(v)$ denotes the diagonal $h \times h$ matrix with elements v_i on the diagonal, and $\epsilon > 0$,

the updating rule is the following:

$$w^{k+1} = w^k - \eta_k \widetilde{V}_k(\varepsilon)^{-1} \frac{1}{1-\beta_1^k} m_k \tag{2.13}$$

where m_k is given in (2.11).
Despite being much less studied than SGD, Adam can be proved to converge under certain hypotheses, that will be discussed in detail in the literature review.

2.2.5 Adamax

Adamax [104] performs mainly the same operations described in Adam, but it does not make use of the parameter ϵ, and the algorithm exploits a modification of the infinite norm to average the gradient.
More formally, let

$$u_k = \max_{i=1,\dots,k} \beta_2^{k-i} \|d_i\| \qquad U_k = \mathrm{diag}(u_k)$$

and m_k as in (2.11).
Thus, the updating rule is:

$$w^{k+1} = w^k - \eta_k U_k^{-1} \frac{1}{(1-\beta_1^k)} m_k \tag{2.14}$$

2.2.6 Nadam

Nadam [56], also known as Nesterov-Adam, performs the same updating rule as Adam but employs the Nesterov acceleration step in (2.9). Nadam is expected to be more efficient, but the Nesterov trick involves only the order in which operations are carried out and not the updating formula itself.

2.2.7 RMSProp

Proposed by G. Hinton et al. in the unpublished lecture [89], RMSProp (Root Mean Square Propagation) performs a similar operations as Adagrad, but the update rule is modified to slow down the learning rate decrease.
In particular, following the notation introduced in the last subsections, at iteration k the following matrix is used:

$$V_k(\varepsilon) = [I\varepsilon + \mathrm{diag}(v_k)]^{\frac{1}{2}}$$

where v_k be given by (2.12). The update rule is:

$$w^{k+1} = w^k - \eta V_k(\varepsilon)^{-1} d_k \tag{2.15}$$

where d_k is again from (2.7).

2.2.8 Adadelta

Adadelta [217] can be considered as an extension of Adagrad, which allows for a less rapid decrease in learning rate. Let us consider the same matrix

$$V_k(\varepsilon) = [I\varepsilon + \text{diag}(v_k)]^{\frac{1}{2}}$$

used in RMSProp.
Further, let $\delta_k = w^{k+1} - w^k$, then we define:

$$\widetilde{\delta}_k = \sum_{t=1}^{k} \rho^{k-t}(1-\rho)\,(\delta_t \odot \delta_t) \qquad \widetilde{\Delta}_k(\varepsilon) = \left[I\varepsilon + \text{diag}(\widetilde{\delta}_k)\right]^{\frac{1}{2}}$$

Thus, we can write the Adadelta updating rule as follows:

$$w^{k+1} = w^k - V_k(\varepsilon)^{-1}\widetilde{\Delta}_{k-1}(\varepsilon)g_k \tag{2.16}$$

2.2.9 FTRL

FTRL (Follow The Regularized Leader) [145], is a regularized version of SGD, which uses the L1 norm to perform the update of the variables. Given, at every iteration k, $d_k = \sum_{t=1}^{k} g_t$, and fixed the quantity σ_k such that $\sum_{t=0}^{k} \sigma_t = \frac{1}{\eta_k}$, the update rule is the following:

$$w^{k+1} = \arg\min_{\omega}\{d_k^T\omega + \sum_{t=1}^{k}\sigma_t||\omega - w^t||^2 + \lambda||\omega||_1\}$$

As proved in [145], the minimization problem in the update rule can be solved in closed form, setting:

$$w^{k+1,i} = \begin{cases} 0 \quad \text{if } |z^{k,i}| \leq \lambda \\ -\eta_k(z^{k,i} - \texttt{sgn}(z^{k,i})\lambda) \text{ otherwise} \end{cases} \tag{2.17}$$

where $z^{k,i} = d_k - \sum_{s=1}^{k}\sigma_s w^s$ and $\texttt{sgn}(\cdot)$ is the Signum function.

2.2.10 Radam

Rectified Adam (RAdam) [127] is a variant of the Adam optimizer that introduces a term to rectify the variance of the adaptive learning rate. This addition aims to address the issue of large variance in the early stages of training. RAdam aims incorporates the advantages of Adam and improves it by reducing the variance of the adaptive learning rate during the initial stages of training, leading to more stable and reliable convergence.
Let d^k be the same estimation stated in (2.7), we define the first and second-moment estimators at iterate k as in Adam in (2.11) and (2.12).
RAdam introduces a rectification term, r_k, dependent on the variance of the moving averages. The rectified term is computed as follows:

$$r_k = \begin{cases} \sqrt{\frac{(k-4)}{(2-\beta_2)(1-\beta_2^{k-1})}} \left(2 - \frac{k}{k+1}\right) \text{ if } k > 4 \\ \frac{2}{1-\beta_2} - 1 \text{ if } k = 0, \ldots, 4 \end{cases}$$

where t denotes the time step.
The rectified first and second-moment estimators are then given by:

$$\widehat{m}^k = \frac{m^k}{1-\beta_1^k} \qquad \widehat{v}^k = \frac{v^k}{1-\beta_2^k}$$

where m^k and v^k are defined as in (2.11) and (2.12).
Given the following matrix:

$$\widetilde{V}_k(\varepsilon) = \left(\frac{1}{1-\beta_2^k}\left[I\varepsilon + \text{diag}(\widehat{v}^k)\right]\right)^{\frac{1}{2}}$$

where $\text{diag}(v)$ denotes the diagonal $h \times h$ matrix with elements v_i on the diagonal, and $\epsilon > 0$, the updating rule is:

$$w^{k+1} = w^k - \eta_k r_k \widetilde{V}_k(\varepsilon)^{-1} \widehat{m}^k \tag{2.18}$$

2.3 Deep neural networks: tasks and architectures

In this section, we discuss in detail the tasks and the neural networks used to present the results included in this thesis. We first describe the tasks of the learning machine, i.e., the deep neural network, and then analyze the structure of the different networks.
Training a deep neural network involves solving the problem (2.5). Therefore, the problem depends on the functions φ and $\mathcal{L}$. The output function of the deep neural network, φ, is determined by the organization of the network layers and their connections. This overall structure is generally referred to as the *architecture*. The architecture defines the output function φ, which we discuss separately for Feedforward Neural Networks (FNNs) in 2.3.1, Convolutional Neural Networks (CNNs) in 2.3.2, and Transformer models in 2.3.3.
Conversely, the loss function $\mathcal{L}$ depends on the metric used to measure the dissimilarity between the ground truth and the predicted output. This function must satisfy Assumptions 2.1.3 and 2.1.4. In this thesis, we use two different loss functions: the mean squared error (MSE) and the cross entropy loss (CEL).
The MSE is used when the task involves predicting a numerical univariate or multivariate output. The MSE for a DNN is defined as:

$$\mathcal{L}_{MSE} = \frac{1}{P}\sum_{i=1}^{P} \|\varphi(x_i; w) - y_i\|^2 \tag{2.19}$$

The CEL is used in classification tasks, where we need to predict the class a given input belongs

to, meaning the output is a boolean vector $y_i \in \{0,1\}^N$ The CEL for a DNN is defined as:

$$\mathcal{L}_{CEL} = -\frac{1}{P}\sum_{i=1}^{P}\sum_{j=1}^{N}[y_i]_j \log([\varphi(x_i;w)]_j) \tag{2.20}$$

where we use the notation $[\cdot]_j$ to indicate the j-th vector of the vector in squared brackets. Both in (2.19) and (2.20), the term $\rho\|w\|^2$, being $\rho > 0$, can be added to ensure that Assumption 2.1.4 holds. This technique is known in DL as *explicit regularization*, or simply L-2 regularization.

2.3.1 Feedforward Neural Networks (FNN)

A Feedforward Neural Network (FNN) consists of several processing units, named *neurons* arranged in layers connected in a feed-forward way with weights. The architecture of a FNN can be represented as an acyclic oriented network, where the computational flow is from the input to the output, as schematically reported in Figure 2.2 A connection edge between two neurons i and j in two adjacent layers ℓ and $\ell+1$ is weighted by a scalar w_{ij}^{ℓ}. Each neuron i in each layer ℓ is characterized by a nonlinear activation function σ_i^{ℓ}. For the sake of simplicity we suppose that the activation function σ^{ℓ} is the same for all the neurons in a layer ℓ.
In order to analytically express the output function $\varphi(x;w)$ of a FNN, we use the following notation. We denote by L the number of layers, while with N^{ℓ} the number of neurons in layer $\ell = 0,\dots,L$. We consider the layer $\ell = 0$ as the input layer. Furthermore, we name d the dimension of the input and m the dimension of the output, meaning that $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}^m$ for each sample $(x_i, y_i) \in \mathcal{T}$. The vector w of the trainable parameters of a FNN, i.e., the vector of the variables of problem (2.6) can be seen as:

$$w = \left[\left[w_{ij}^{l}\right] \text{ for } i,j = 1,\dots,N^{\ell}\right]_{\ell=0,\dots,L}$$

where, to avoid an abuse of notation, we omit the biases of each neuron, which can be represented using a fake unitary input [163].
Now, naming $z^{\ell} \in \mathbb{R}^{N^{\ell}}$, the output of the layer ℓ and being $W^{\ell+1}$ the $N^{\ell+1} \times N^{\ell}$ matrix of the weights of layer $\ell+1$, we can write the output of the subsequent layer as:

$$N_{\ell+1} = \sigma^{\ell+1}\left(W^{\ell+1}z^{\ell}\right) \tag{2.21}$$

Now, naming $z^{\ell} \in \mathbb{R}^{N^{\ell}}$, the output of the layer ℓ and being $W^{\ell+1}$ the $N^{\ell+1} \times N^{\ell}$ matrix of the weights of layer $\ell+1$, we can write the output of the subsequent layer as:

$$N_{\ell+1} = \sigma^{\ell+1}\left(W^{\ell+1}z^{\ell}\right) \tag{2.22}$$

Eventually, we conclude that, for a FNN, the output function can be seen in a nested form as follows:

$$\varphi(x;w) = W^{L}\sigma^{L-1}(W^{L-1}\sigma^{L-2}(\dots\sigma^{1}(W^{1}x)\dots)) \tag{2.23}$$

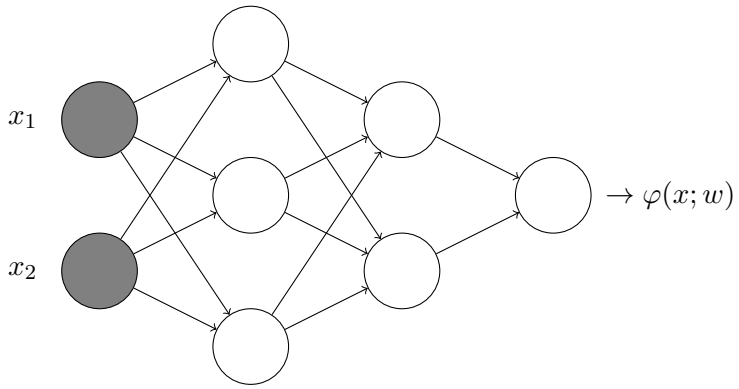


Figure 2.2: Example of a feedforward neural network with $L = 2$ hidden layers, an output of dimension $n = 2$, and a univariate output.

FNNs represent the simplest form of neural networks and are universal function approximators [91], which means they can approximate any continuous function given sufficient neurons and layers. This makes them extremely versatile for various tasks, including both regression and classification. However, they suffer from the *curse of dimensionality*. As the number of input features (dimensionality) increases, the volume of the input space grows exponentially, making the data sparse. This sparsity makes it challenging for FNNs to generalize well from the training data because the network would require an exponentially large amount of data to learn the underlying patterns effectively. In the computational practice, this makes it impossible to use FNNs when the dimension of the input is more than few hundreds or thousands.

2.3.2 Convolutional Neural Networks (CNN)

To address the limitations of FNNs regarding the curse of dimensionality, Convolutional Neural Networks (CNNs) [11] introduce a novel approach to manage high-dimensional input data. CNNs are specifically designed to handle grid-like topology in data, such as images, by exploiting spatial hierarchies through *convolutional layers*. Convolutional layers operate by applying a series of numerical filters (or kernels) across the input data, performing a local operation that captures spatial relationships and patterns. This local connectivity means that each filter processes only a small portion of the input at a time, drastically reducing the number of parameters compared to fully connected layers in FNNs. This reduction in parameters helps mitigate the curse of dimensionality, as the model requires less data to learn effectively. While CNNs are used in a wide range of applications, including natural language processing and time-series analysis, their most prominent success stories are in the domain of image-related tasks. While for a more comprehensive analysis of CNNs we refer the reader to [10, 11], we describe here how to obtain a closed expression for the output $\varphi(x; w)$ of a CNN when the input is a 3-axis tensor x of dimensions $C \times H \times W$ (channels, height, width). Notice that, if the input is a colorful red-green-blue (RGB) image, the number of channels is $C = 3$, and $H \times W$ represents the number of pixels.

The overall structure of a CNN is schematically depicted in Figure 2.3. In the figure, a colorful

input x, is sent into a series of convolutional layers. Then, the output of the last convolutional layer is flattened and stacked in a single-axis vector. Finally, this vector becomes the input of a series of fully connected layers. To write in formulas the relation between the input 3-axis tensor and the output vector, we firstly need to understand how does a convolutional layer work.

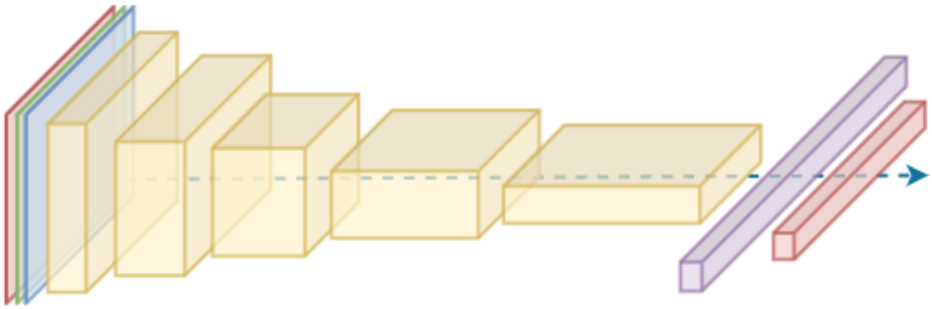


Figure 2.3: An example of CNN with an RBG image as input, five convolutional layers, and two fully connected layers. Figure taken from [44].

Each convolutional layer performs up to four subsequent operations: a standard 2D-convolution, followed by an activation function, then a pooling operation, and finally the batch normalization. The input $X^{\ell-1}$ to a layer ℓ is a tensor of dimension $[d^{\ell-1}, m^{\ell-1}, m^{\ell-1}]$ where $d^{\ell-1}$ are the channels and m^ℓ is the height/width of each channel $c = 1, \dots, d^{\ell-1}$, and the output X^ℓ is a tensor of dimension $[d^\ell, m^\ell, m^\ell]$. The input X^0 at layer $\ell = 0$ of the layer is the colourful sample image represented with m^0 pixels and $d^0 = 3$ colour channels (red, green, blue). We denote by X_c^ℓ the matrix $m^\ell \times m^\ell$ for each channel $c = 1, \dots, d^\ell$ and by $x_c = X_c^0 \in \mathbb{R}^{m \times m}$ for $c = 1, \dots, d^0$. A layer ℓ applies a discrete convolution on the input $X^{\ell-1}$. This operation consists in applying filters (also called kernels) $w_c^k \in \mathbb{R}^{n \times n}$ for $k = 1, \dots, d^\ell$ to the c-th input channel $X_c^{\ell-1}$ with $c = 1, \dots, d^{\ell-1}$.

The convolution operation depends on the integer stride $s \geq 1$, representing the amount by which the filter w_c^k shifts around the input $X_c^{\ell-1}$. The stride is commonly set to $s = 1$. The dimension of the filter n, the number of channels d^ℓ, and the stride $s \geq 1$, for each layer ℓ, are network hyperparameters. Let us denote as $\otimes_s$ the discrete convolution operation with stride $s \in \mathbb{N}$. The component-wise expression of the convolutional operation between the filter $w_c^k \in \mathbb{R}^{n \times n}$ and the input feature $X_c^{\ell-1}$ is

$$\left[w_c^k \otimes_s X_c\right]_{ij} = \sum_{a=1}^{n} \sum_{b=1}^{n} [w_c^k]_{a,b} [X_c]_{i+sa, j+sb} \quad i, j = 0, \dots, m-n \tag{2.24}$$

where, for the sake of simplicity, we avoid the use of the superscript ℓ. As reported in [10] Chapter 9 (equation (9.4) with $s = 1$), the k-th convoluted output is the matrix $F_k^\ell \in \mathbb{R}^{\frac{(m^{\ell-1}-n+1)}{s} \times \frac{(m^{\ell-1}-n+1)}{s}}$ defined as the sum over the channels, namely

$$F_k^\ell = \sum_{c=1}^{d^{\ell-1}} w_c^k \otimes_s X_c^{\ell-1} \qquad k = 1, \dots, d^\ell \tag{2.25}$$

The convoluted output F^ℓ of layer ℓ is thus:

$$F^\ell = [F_k^\ell]_{k=1,\dots,d^\ell} = (F^1, \dots, F^{d^\ell}) \in \mathbb{R}^{d^\ell \times \frac{(m^{\ell-1}-n+1)}{s} \times \frac{(m^{\ell-1}-n+1)}{s}} \tag{2.26}$$

The parameters of the convolutional layer ℓ are denoted as

$$W_{Conv}^\ell = \left[w_c^k\right]_{c=1,\dots,d^{\ell-1}}^{k=1,\dots,d^\ell} \in \mathbb{R}^{n \times n \times d^\ell \times d^{\ell-1}} \tag{2.27}$$

The nonlinear activation function σ applied to the F^ℓ output ℓ can be, for instance, the ReLU $\sigma(z) = \max\{0, z\}$ or the SiLU $\sigma(z) = \frac{z}{1+e^{-z}}$ [147, 172].
The output can be submitted to the pooling operation aimed at further reducing the image dimensionality. This practice is often referred to as *downsampling*. The pooling operation `Pool` involves sliding a two-dimensional non-overlapping $p \times p$ matrix, where p is the pool size, over each convoluted output F_k^ℓ and contracting the features lying within the $p \times p$ region covered by the filter by using the `max` or the `mean` operation. More formally, let us introduce the set $\mathcal{P}_{(i,j)}$ of positions i.e., for each (i, j) the rows and columns of the matrix F_k^ℓ, that are located in the $p \times p$ region; then `Pool`: $F_k^\ell \to G_k^\ell$ where the k-th pooled output is

$$G_k^\ell \in \mathbb{R}^{\frac{(m^{\ell-1}-n+1)}{sp} \times \frac{(m^{\ell-1}-n+1)}{sp}} \tag{2.28}$$

The output of the Max-Pool or Mean-Pool layer is computed respectively as:

$$(G_k)_{ij}^{Max} = \max_{(r,c)\in\mathcal{P}_{(i,j)}} (F_k)_{r,c} \qquad (G_k)_{ij}^{Mean} = \frac{1}{\left|\mathcal{P}_{(i,j)}\right|} \sum_{(r,c)\in\mathcal{P}_{(i,j)}} (F_k)_{r,c} \tag{2.29}$$

where for the sake of simplicity we removed the superscript ℓ. The set $\mathcal{P}_{(i,j)}$ depends on how drastically we want to reduce the dimensionality. In our experiments, we have set $p = 2$. In this case, the moving region is just a 2×2 matrix, thus we halve the dimension of F. For instance, $\mathcal{P}_{(1,1)} = \{(1, 1), (1, 2), (2, 1), (2, 2)\}$, meaning that G_{11}^k is the maximum/mean value between four different values $\{F_{1,1}^k, F_{1,2}^k, F_{2,1}^k, F_{2,2}^k\}$.
The output of a convolutional layer with downsampling, obtained by 2.25 and 2.29, is finally given as:

$$Z^\ell = \left\{\texttt{Pool}\left[\sigma\left(F_k^\ell\right)\right]\right\}_{k=1,\dots,d_\ell} \tag{2.30}$$

In most applications, Z^ℓ is then normalized to stabilize and speed up the training process. The normalization is performed following the standard batch-normalization procedure described in [92], i.e., subtracting the mean and dividing by the standard deviation.
Finally, naming $Z^L(x; W_{conv})$ the output of the last convolutional layer, depending on the input x and on the convolutional layers parameters $W_{conv} = \left[W_{conv}^\ell\right]_{\ell=1,\dots,L}$, and recalling (2.23) for the output of the LF fully connected layers we can write the closed expression:

$$\varphi(x; w) = W_{fc}^{LF}\sigma(\dots\sigma(W_{fc}^1 Z^L(x; W_{conv}))\dots) \tag{2.31}$$

where $w = \left(W^1_{conv}, \ldots, W^L_{conv}, W^1_{fc}, \ldots, W^{LF}_{fc}\right)$ is the vector of the trainable parameters of the CNN, i.e., the vector of variables of problem (2.5).

2.3.3 Self-attention mechanism: the Transformer architecture

Transformers, originally designed in [204] for natural language processing (NLP) tasks, utilize a mechanism called *self-attention*, which allows them to weigh the importance of different input elements, regardless of their position. This enables Transformers to capture global context and long-range dependencies in input data more effectively than CNNs. Thanks to this ability, transformers can better understand complex patterns and relationships in data, making them suitable for a wider range of tasks.

The output function ϕ for a transformer varies significantly depending on the architectural solution adopted and it is not straightforward to write a unique closed expression. For a detailed analysis of the most commonly used transformers, we refer the reader to the surveys [85, 125]. Additionally, we will provide specific citations in further chapters when discussing state-of-the-art architectures. Here, we introduce the overall structure of a transformer layer, which is schematically reported in Figure 2.4.

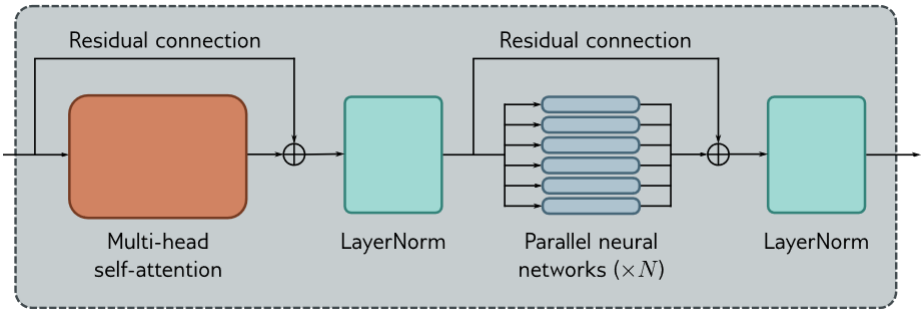


Figure 2.4: Schematic representation of a transformer layer. Picture taken from [169].

Differently from what we have seen for CNNs and FNNs, the input of a transformer layer is mathematically a sequence of N vectors $x_i \in \mathbb{R}^D$ with $i = 1, \ldots, N$. Notice that each x_i can in general be a multi-axis tensor but for the sake of simplicity we show how to derive the output function in the case $x_i \in \mathbb{R}^D$. In the computational application that we will present in this thesis, when transformer layers are used, x_i is a tensor representing a *patch* of an RGB image, i.e., a small, contiguous block of pixels taken from an image. A more detailed analysis can be found in Chapter 12 of [169]. A typical transformer layer, as in figure 2.4, is composed of several key components: Multi-Head Self-Attention (`MhSa`), Layer Normalization (LayerNorm), and Multi-Layer Perceptrons (MLPs) arranged in parallel, followed by another LayerNorm. Each of these components plays a crucial role in the functionality of the layer.

To understand how an input sample $x = [x_1, x_2, \ldots, x_N] \in \mathbb{R}^{D \times N}$ is transformed, we firstly describe the *self-attention* mechanism. Self-attention (`Sa`) is a mechanism that enables a model to weigh the importance of different elements of the input sequence relative to each other. `Sa` aims to capture dependencies between elements, regardless of their position (e.g., different objects in a picture). The output of the self-attention mechanism is, as the input, a matrix

$\texttt{Sa}(x) \in \mathbb{R}^{D\times N}$. To obtain this matrix, the mechanism firstly computes three new representations of the input elements as follows:

- Queries $Q = \beta_q \mathbf{1}_N^T + \Omega_q x$;
- Keys $K = \beta_k \mathbf{1}_N^T + \Omega_k x$;
- Values $V = \beta_v \mathbf{1}_N^T + \Omega_v x$.

Notice that these representations have the same dimension of the input $D \times N$ and depend on the network trainable parameters $\Omega_q, \Omega_k, \Omega_v \in \mathbb{R}^{D\times D}$ and $\beta_q, \beta_k, \beta_v \in \mathbb{R}^{D\times 1}$. The terms "queries," "keys," and "values" are derived from information retrieval and database systems, where they describe how information is accessed and matched. Once Q, K, V are computed, the queries and the keys are used to compute the scaled dot-product attention to get the matrix of attention scores A. These scores indicate how much focus should be given to each element in the input sequence, as they measure the similarity between two different representations of the same input. The matrix A is obtained applying the $\texttt{Softmax}$ operator to the scaled dot-product between the queries and keys as follows:

$$A = \texttt{Softmax}\left(\frac{K^T Q}{\sqrt{D}}\right) \in \mathbb{R}^{N\times N} \tag{2.32}$$

More in detail, this means that the single element of the attention score matrix can be written as:

$$A_{ij} = \frac{e^{k_i^T q_j}}{\sum_{r=1}^{N} e^{k_r^T q_j}} \tag{2.33}$$

where k_i is the i-th column of the keys matrix K and q_j is the j-th column of the queries matrix Q. The self-attention mechanism is schematically reported in Figure 2.5.

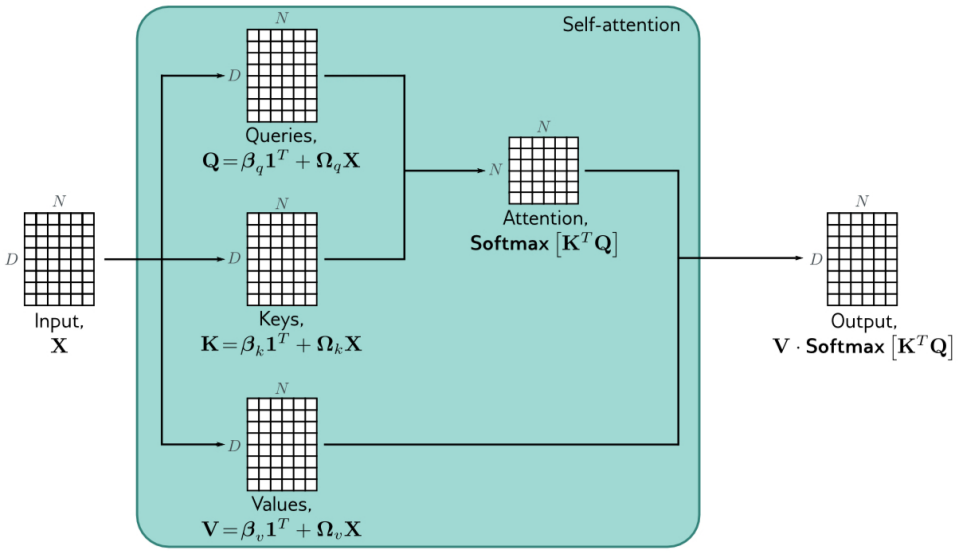


Figure 2.5: Schematic representation of how the output of a self-attention mechanism is obtained from an input X. Picture taken from [169].

The multi-head self-attention only aggregates the outputs from H different self-attention mech-

anisms as in Figure 2.5, using an aggregator function denoted as $\mathcal{A}(\cdot)$, in the form:

$$\texttt{MhSa}(x) = \mathcal{A}\left(\texttt{Sa}_1(x), \texttt{Sa}_2(x), \ldots, \texttt{Sa}_H(x)\right) \tag{2.34}$$

where $\texttt{Sa}_h(x)$ is the output of the h-th self-attention process and $\mathbb{A}(\cdot)$ is usually the arithmetical average.

Following the representation of Figure 2.4, the output of the multi-head self-attention is then normalized subtracting the mean and dividing by the standard deviation, as seen for CNNs. The normalized $\texttt{MhSa}(x)$, along with a residual connection where x is added back, is then fed into N parallel FNNs. More formally let the normalized output of the multi-head self-attention be:

$$Z = x + \texttt{MhSa}(x) = [z_1, z_2, \ldots, z_N] \in \mathbb{R}^{D \times N}$$

To obtain the final output of the transformer layer, each element z_i is fed into a different FNN. We indicate with $\phi(z_i; w_i)$ the output of the i-th FNN, where the function ϕ is the same as in (2.23) and w_i is vector of the trainable parameters of the i-th FNN. Eventually, naming $\hat{\phi}(z_i; w_i)$ the normalized output of the i-th FNN, we can write the output of the transformer layer as:

$$\texttt{TransformerLayer}(x) = [\hat{\phi}(z_1; w_1), \ldots, \hat{\phi}(z_N; w_N)] \in \mathbb{R}^{D \times N} \tag{2.35}$$

We remark that a single transformer layer has a significantly larger amount of trainable parameters than any other previously seen layer. This number is proportional to the number of parameters of N FNNs and of H self-attention mechanisms, while each single self-attention mechanism has a number of trainable parameters proportional to D^2. When working on images D represents the pixels and is usually in the order of thousands, meaning that a single self-attention mechanism can have millions of parameters. Despite the overall dimension of a transformer-based network varies across different architectural solutions, it can be easily deduced from these simple considerations how the number of trainable parameters can grow up to order of billions (or even trillions in the field of natural language processing) when a large number of transformer layers are stacked in sequence. Hence, due the number of variables of problem (2.6), the transformer architecture poses a serious challenge to any optimization method and has become a driver pushing forward the boundaries of the large-scale optimization theory.

Chapter 3

Optimization methods for deep learning: an overview of the convergence theory

The ambitious goal of this thesis is to propose novel algorithms with a fully developed convergence theory for optimizing large-scale neural models. Achieving this objective requires a deep understanding of the existing literature, including what has been done in the past and the current state-of-the-art. In this chapter, we provide the reader with the essential theoretical references to grasp the forthcoming content.

While the previous chapter reviewed the fundamentals of machine learning, outlined state-of-the-art algorithms, and introduced the neural networks that form the experimental testbed, as well as formulated our target problem as an unconstrained minimization problem, this chapter investigates the various existing theoretical approaches. More in detail, we examine the trade-offs between convergence guarantees, the assumptions required for proving convergence, and computational efficiency for the algorithms we mentioned in the previous chapter. This chapter is dedicated to the literature review of the convergence theory of optimization algorithms for deep learning, in a highly non-convex framework.

Seeking for a gentle introduction to the novel algorithmic approach that is proposed in this thesis (starting from chapter 4), we structured this chapter as follows. In section 3.1 we briefly review the main theoretical aspects of stochastic methods, which are still the most widespread in deep learning applications. Then, in section 3.2, we dive deeper into the theory of deterministic algorithms. While these are not the most commonly used methods in deep learning, we still need to recall their convergence theory, as we will adopt a fully deterministic approach in our proofs, even when elements of randomness will be introduced. Finally, we dedicate section 3.3 to the review of the main existing works that are based on line-search techniques and, as our framework, does not fall in any traditional classification.

3.1 The convergence of stochastic methods

In the context of deep learning, stochastic optimization methods aim to solve the large scale minimization problem formalized in (2.6) exploiting first-order information. As all mini-batch methods, which have been discussed in section 4.3, instead of using the full dataset to compute the gradient, they rely on a subset of data (a mini-batch) to estimate the gradient, which makes them much more efficient in terms of computational time. We consider as *stochastic* any mini-batch method that uses an estimate of the gradient based on random elements. In particular, stochastic methods rely on randomly selected mini-batches and can be categorized in various ways [26, 95, 111, 128]. However, for the purpose of this thesis, a useful distinction can be made between adaptive and non-adaptive methods, according to the step-size updating rule. Non-adaptive methods maintain a constant learning rate or use pre-defined decreasing schedules that do not depend on the observed mini-batch gradients during training. This approaches is simple, computationally efficient, and universally acknowledged as the foundation for understanding more advanced methods. In contrast, adaptive methods, such as most of algorithms discussed in the previous chapter in section 4.3, adjust the step-size dynamically based on the magnitude of the past gradients. These methods often lead to faster convergence in practice because they handle noisy and sparse gradients more effectively.
Studying the convergence of stochastic methods presents significant challenges. Unlike deterministic methods where the gradient estimate is available in a closed formula, in stochastic methods this estimate depends on the sampling distribution over the dataset. The introduction of randomness into the optimization process would require advanced stochastic tools to study its behavior rigorously, which is beyond the scope of this thesis. Nonetheless, stochastic methods remain the most relevant, efficient, and well-established competitors to the novel approach proposed in this thesis. In the following subsections, we will review the history and key theoretical foundations of the most widely used stochastic mini-batch methods. We will begin with the earliest algorithm, stochastic gradient descent, move on to random reshuffling, and conclude with adaptive methods.

3.1.1 Stochastic gradient descent (SGD)

The Stochastic Gradient Descent (SGD) method, initially developed in 1951 by Robbins and Monro [176], is one of the most fundamental optimization techniques in machine learning. As the reader is already familiar with the basic SGD iteration (recalled in section 4.3), we will not delve into the mechanics of the algorithm here. For the sake of completeness, we only report the algorithmic scheme of SGD in Algorithm 3.1.

Stochastic notation and basic assumptions. In SGD, the variables are updated at each step based on the current gradient estimate, which is derived from a randomly sampled mini-batch. Since the mini-batch is chosen randomly (step 4 of Algorithm 3.1), there is no closed expression for the update direction as a function of the gradients ∇f_i. This is why the direction d^k is treated as a random variable, often referred to as a gradient estimator. A key challenge here is that this random variable must be somehow related to the true gradient and appropriately

Algorithm 3.1 `Stochastic Gradient Descent (SGD)`

Input: $w^0 \in \mathbb{R}^n$, α^0
Set $k = 0$
while (stopping criterion not satisfied) **do**
  Randomly select a sample $\mathcal{B}_k \subset \{1, \dots, P\}$
  $\tilde{d}^k = -\sum_{i \in \mathcal{B}_k} \nabla f_i(w^k)$
  Set $w^{k+1} = w^k + \alpha^k d^k$
  Update step-size α^k
  Set $k = k + 1$
end while
Output w^k

bounded to prove the convergence to a stationary point. Hence, beyond the already stated assumptions 2.1.1, 2.1.2, 2.1.3, 2.1.4, the convergence of SGD requires additional hypotheses on this gradient estimate. These hypotheses can vary significantly across the literature but, adapting the survey by Bottou et al. [26] to our notation, the most relevant ones can be formalized as follows:

Assumption 3.1.1 (Standard assumptions on d^k)**.** *The gradient estimator is a stochastic variable satisfying the following properties:*

i) d^k *is an unbiased estimator of the anti-gradient, i.e.:*

$$\mathbb{E}[d^k] = -\nabla f(w^k) \ \forall k = 0, 1, \dots \tag{3.1}$$

ii) d^k *is bounded in the Euclidean norm, i.e., there exists a positive constant* $A > 0$ *such that:*

$$\|\mathbb{E}[d^k]\| \leq A \|\nabla f(w^k)\| \ \forall k = 0, 1, \dots \tag{3.2}$$

iii) d^k *is expected to be a sufficiently good descent direction for the objective function, i.e., there exists a positive constant* $B > 0$ *such that:*

$$\nabla f(w^k)^T \mathbb{E}[d^k] \leq -B \|\nabla f(w^k)\|^2 \ \forall k = 0, 1, \dots \tag{3.3}$$

iv) d^k *has a bounded variance, i.e., there exists two positive constants* $M_1, M_2 > 0$ *such that:*

$$\mathbb{V}[d^k] \leq M_1 + M_2 \|\nabla f(w^k)\|^2 \ \forall k = 0, 1, \dots \tag{3.4}$$

If assumption 3.1.1 holds, then, thanks to the well-known descent lemma [14], it is possible to prove (see section 4.2 of [26]) that, for all steps $k = 0, 1, \dots$:

$$\mathbb{E}[f(w^{k+1})] - f(w^k) \leq \alpha^k \nabla f(w^k)^T \mathbb{E}[d^k] + \frac{1}{2} L (\alpha^k)^2 \mathbb{E}[\|d^k\|^2] \tag{3.5}$$

This result is the starting point for the convergence analysis of SGD in different settings, and is generally used to determine the minimal per-step expected improvement in the value of the

objective function. However, the assumptions 3.1.1 are not trivial and not always easily to attain in deep learning. In particular, the (3.3) means that, in the expected value, we are following a sufficiently good direction, i.e., a descent direction. However, the computational practice in deep learning shows that 3.3 may not hold under certain conditions. One of the most recent examples of this case is the *byzantine* federated learning framework [47, 97], a situation often observed in real-world applications when the upcoming information from the data source can be corrupted and, hence, for some epochs, d^k is a wrong estimator for $\nabla f(w)$. A robust approach to address these situations is being developed but still under the bounded variance hypothesis [98, 116, 117] or the convexity of the objective function [118].

Convergence of SGD with diminishing step-size. The theoretical result that marked a milestone in studying the convergence of SGD is the proof provided 24 years ago by Bertsekas and Tsitsiklis in [16]. They prove the global convergence of SGD under the assumptions 3.1.1 and under the condition that the step-size α^k is chosen according to a decreasing rule that satisfies:

$$\lim_{k\to\infty} \alpha^k = 0 \qquad \sum_{k=0}^{\infty} \alpha^k = \infty \tag{3.6}$$

The convergence proof relies on (3.5) and is reported in Proposition 3 of [16]. More recently Bottou, Curtis and Nocedal applied this result and formalized, in Theorem 4.10 of [26], the minimal expected per-step improvement under the assumptions 3.1.1 as follows:

$$\mathbb{E}[f(w^{k+1})] - \mathbb{E}[f(w^k)] \leq -\frac{1}{2} B\alpha^k \mathbb{E}[\|\nabla f(w^k)\|^2] + \frac{1}{2}(\alpha^k)^2 L M_2 \tag{3.7}$$

where L is the Lipschitz constant of ∇f, following the notation introduced in the previous chapter.
Since the step-size goes to zero as in (3.6), the minimal expected per-step improvement will go to zero too and, in this case, it is not possible to derive a universal worst-case complexity bound to find an ε-stationary point. We remark this consideration because this will be the case in most of the algorithms presented in this thesis as well. However, for the sake of completeness, we briefly recall the alternative approach.

Convergence of SGD with fixed step-size. The case of a fixed step-size, while lacking the theoretical guarantees of global convergence associated with diminishing step-size, is still widely used in practice. The behavior of SGD under a constant step-size $\alpha^k = \alpha > 0$ has been extensively studied, and although it does not necessarily guarantee convergence to a stationary point, it can still provide meaningful results for non-convex problems with a properly chosen α. One of the first theoretical result on SGD in a non convex setting with a properly chosen step-size bounded away from zero dates back to 2013. Gadhimi et al. proved, in Theorem 2.1 of [70] and in the following corollary, that, if $\sigma > 0$ exists such that:

$$\mathbb{E}[\|d^k - \nabla f(w^k)\|^k] \leq \sigma^2 \tag{3.8}$$

then SGD finds an ε-stationary point with a $\mathcal{O}(\varepsilon^{-4})$ complexity. However, finding the minimal expected improvement and the complexity bound relies on an additional hypothesis (step-size $\alpha < \frac{2}{L}$), which is impossible to verify without the knowledge of the Lipschitz constant. Furthermore, condition (3.8) is even stronger than (3.4).
More recently, Bottou proved in Proposition 2.12 of [26], that, under the assumptions 3.1.1, if $\alpha^k = \alpha \in (0, \frac{B}{L(M_2+A^2)})$, it holds the following bound:

$$\mathbb{E}\left[\frac{1}{k}\sum_{j=0}^{k}\|\nabla f(w^j)\|^2\right] \leq \frac{\alpha L M_1}{B} + \frac{2(f(w^0) - f^{low})}{kB\alpha} \tag{3.9}$$

where f^{low} is a global lower bound of the objective function. This result implies, in practice, that SGD can converge to an ϵ-stationary point within $\mathcal{O}(\varepsilon^{-2})$ steps. However, the step-size must be again chosen knowing the Lipschitz constant and the constants A, B, M_2. Furthermore, Khaled et al. in [101], claiming that (3.2) is too restrictive, provided a trivial case where (3.2) does not hold.
In [101], Khaled et al. prove the same convergence result with the same fixed step-size than in [26], but with a relaxed hypothesis on the gradient estimator, the expected smoothness of the search direction:

$$\mathbb{E}[\|d^k\|^2] \leq A(f(w^k) - f^{low}) + B\|\nabla f(w^k)\|^2 + C \tag{3.10}$$

with A, B, C positive constants. However, the chose of the step-size still depends on the Lipschitz constant (lemma 2 of section 5.1 in [101]).

Practical approach in deep learning. Despite the remarkable results achieved in the convergence theory of SGD during the past 25 years, the use of SGD in deep learning is still highly affected by the difficult choice of the step-size. All the convergence results discussed here rely on the knowledge of constants that are, in deep learning, unknown and hardly to estimate [44]. In particular, when using logarithmic losses like CEL (2.20), using fixed step-sizes that are inversely proportional to the Lipschitz constant would be highly ineffective and lead to slow convergence [8, 206]. The most commonly used approach is to use a pre-set decreasing schedule for the step-size α^k. The decreasing rule and its parameters often depend by hyper-parameters that are tuned adopting a trial-and-error approach, but they mostly satisfy the condition in (3.6). However, there have also been attempts to develop more elaborated techniques to decrease the step-size, trying to make SGD closer to adaptive methods [94].

3.1.2 Random Reshuffling (RR)

The convergence of SGD with random reshuffling, named for the sake of simplicity Random Reshuffling (RR), is generally harder to prove and less studied than the standard case of simple SGD (sampling with replacement). The core reason for this difficulty lies in the bias introduced by the dependencies between the samples in random reshuffling. RR, which we report in Algorithm 3.2 in the case of mini-batches containing only one sample, involves shuffling the entire dataset at the beginning of each epoch and then iterating over the data without replacement.

This introduces a fundamental dependency between the gradient estimates since the data points are correlated within each epoch. The gradient estimate d^k at any step depends not just on the current sample but also, implicitly, on the samples that have already been used in the current epoch and that cannot be picked up again.

Algorithm 3.2 `Random Reshuffling (RR)`

Input: w^0, α^0
Set $k = 0$
while (stopping criterion not satisfied) **do**
  Let $I_k = \{h_1, \dots, h_P\}$ be a random permutation of $\{1, \dots, P\}$
  Set $\widetilde{w}^k_{h_0} = w^k$
  for $i = 1, ..., P$ **do**
    $\tilde{d}^k_{h_i} = -\nabla f_{h_i}(\widetilde{w}^k_{h_{i-1}})$
    $\widetilde{w}^k_{h_i} = \widetilde{w}^k_{h_{i-1}} + \alpha^k \tilde{d}^k_{h_i}$
  end for
  Set $\widetilde{w}^k = \widetilde{w}^k_P$
  Update α^k
  Set $k = k + 1$
end while
Output: w^k

However, in the computational practice, the random reshuffling of the data samples is proved to achieve faster convergence and improved stability during training [23, 173, 196], with only few exceptions on highly specific tasks [49].

To the best of our knowledge, the specific literature on RR is much more limited if compared to SGD. RR is often (but from a mathematical perspective erroneously) considered nothing but a variant of SGD and is embedded in the SGD algorithm in state-of-the-art deep learning libraries. However, considering the non convex setting, which is of our interest for deep learning applications, there have been two remarkable theoretical results in the past years.

In 2020, Nguyen et al. [159] developed a unified and organized convergence theory of RR under different hypotheses. While the main results rely on convexity or strong convexity, they also provide an absolutely novel convergence proof of RR in the non convex settings, under the assumptions 3.1.1 and with a fixed step-size inversely proportional to the Lipschitz constant of the gradient. In theorem 6 of [159], the authors determine the minimal expected per-step improvement and compute a convergence rate that depends on the Lipschitz constant of the gradient, on the constant A from (3.2) and on the number of samples P. Mischenko et al. [148] rewrite this bound as a function of the expected number of steps needed to attain an ε-stationary point and find a worst-case bound of $\mathcal{O}\left(LP(\varepsilon^{-2} + A\varepsilon^{-3})\right)$ steps to achieve ε-stationarity. While being worse than the bound proved by [26] for SGD from (3.9), Nguyen et al. prove that this bound can be significantly improved in certain cases. More in detail, in theorem 3 of [159], the authors prove that, if (3.8) holds, then RR finds an ε-stationary point within a maximum expected number of $\mathcal{O}\left(L\sigma\sqrt{P}\varepsilon^{-\frac{3}{2}}\right)$ steps, which is significantly better than the SGD bound in [26] at least by a factor $\varepsilon^{-\frac{1}{3}}$. Notice that the same improvement holds even if we consider the results of [70], where the hypothesis (3.8) was initially formulated. This result is relevant to us because it somehow explains why in many deep learning applications the use of random reshuffling leads to a remarkable improvement in terms of efficiency.

The other relevant result for RR is the extension of the results of [159] provided by Mischenko

et al. in [148]. The authors enlighten the assumption (3.4) and prove, in theorem 4 of [148], that if there exists positive constants C_1, C_2 such that

$$\frac{1}{P}\sum_{i=1}^{P}\|\nabla f_i(w) - \nabla f(w)\|^2 \leq 2C_1(f(w) - f^{low}) + C_2^2 \tag{3.11}$$

then, RR converges to an ε-stationary point within $\mathcal{O}\left(L(P\varepsilon^{-2} + \sqrt{P}(\sqrt{C_1} + C_2)\varepsilon^{-3})\right)$ steps. However, to prove this result the step-size must be inversely proportional to $\max\{2LP, (C_1L^2P^2)^{\frac{1}{3}}\}$. To provide a practical example of how such a step-size can lead to extreme slow convergence, let us consider a classification task on the Imagenet dataset [51]. In such a case, we optimize the cross entropy loss (2.20) over approximately $P = 14M$ (14 millions) samples. While it is impossible to determine exactly L for deep neural networks, it can be estimated [18, 112, 167] and it can certain assume extremely high values for logarithmic losses like CEL. However, even if taking the value of $L = 2$, which only holds for the CEL on the Softmax function, the step-size would be proportional to approximately 1.2×10^{-8}, a learning rate that is at least 5 orders magnitudes smaller than the default in state-of-the-art libraries.
Despite the remarkable theoretical advances in the recent years, we believe that the convergence properties of RR are still an open research field and a new theory is still to be developed.

3.1.3 Adaptive methods

The underlying mathematical theory of adaptive methods is much less studied than SGD and RR. Excluding some recent attempts to study adaptive methods within a generalized framework, which will be discussed in section 3.3, to the best of our knowledge, a proper convergence theory in the non convex setting has been developed only on Adam and Adagrad. While for the mechanics of the single algorithms we refer the reader to section 4.3, we remark that proving the convergence of adaptive methods is much more challenging, compared to the SGD/RR analyses reported in the previous subsection. On the one hand, adaptive methods are inherently stochastic - the gradient estimates, as for SGD/RR, depend on the sampling distribution over the dataset. On the other hand, studying the behavior in the limit of adaptive methods, one cannot direct apply (3.5). In fact, even in the optimistic scenario where all the assumptions 3.1.1 hold, the step-size α^k does not properly exist. The step-size, at each update, is individual for each network parameter and is based on the previously observed mini-batch gradients.
Concerning Adagrad [58], the first remarkable result in the non-convex setting, dates back to 2019, eight years after the algorithm was proposed. Li et al. proved, in theorem 1 of [121], that, under the assumptions 3.1.1, $\lim_{k\to\infty}\inf\mathbb{E}[\|\nabla f(w^k)\|^2] = 0$ with probability 1. Moreover, in 2020, Xie et al. proved, in theorem 3 of [209], a $\mathcal{O}(\log(\varepsilon^{-1}))$ under the Poljak-Lojasiewicz condition. While their result has a limited application on those particular ML frameworks where the Poljak-Lojasiewicz condition holds [87, 215], it is remarkable that the proof holds for *any* initial step-size. In 2021, Kairouz et al. [96] managed to prove a $\mathcal{O}((\varepsilon P)^{-1})$ convergence rate of Adagrad in the specific case of ERM with differential privacy. A similar result can be found in [130], but assuming a specific form of quasi convexity of the objective function,

named γ-quasar convexity (assumption 1 in [130]). The most generalized result for Adagrad in the non-convex space, with a potential extension to Adadelta [217], is the proof provided by Ward et al. in [208]. The authors, adopting a fully deterministic setting, prove, in theorem 2.2 of [208], the convergence of Adagrad to an ε-stationary point within $\mathcal{O}(\varepsilon^{-2}(L\Delta + \frac{\Delta}{\alpha})^2)$ steps, being $\Delta = f(w^0) - f^{low}$. Differently from what we have seen for RR [148] and SGD [101], Ward et al. do not require neither (3.2), nor its deterministic version.
Being Adam [104] more recent than Adadelta, its convergence theory is much less developed. convergence proofs have been provided under different hypotheses, most of all involving convexity or only achieving local convergence. In 2019 Bock et al. [21] proved the local convergence[1] of Adam to a stationary point in a non-convex setting. More recently, in 2022, Zhang et al. [218] proved that, for certain hyper-parameters setting, Adam can converge to a neighborhood of a stationary point. The authors prove that, if (3.4) holds and β_1, β_2 are such that $\beta_1 < \sqrt{\beta_2} < 1$, then Adam converges to a point that is within an $\mathcal{O}(\sqrt{M_1})$ distance from a stationary point, where M_1 is the same constant from (3.4). Eventually, in 2024, Li et. al [119] proved that Adam converges to an ε-stationary point within $\mathcal{O}(\varepsilon^{-4})$ steps under the assumptions 3.1.1, local smoothness of the objective function and bounded norm of Hessian matrix $\nabla^2 f(w)$.

3.2 Deterministic methods

Recalling the definition of stochastic method provided at the beginning of the previous section, we consider as *deterministic* any mini-batch method that relies on the following estimate of the gradient:

$$d^k = \nabla f(w^k) + e^k \tag{3.12}$$

where $\nabla f(w^k)$ is the true gradient in w^k and e^k is an error that can be written with a *closed deterministic expression*. The convergence of deterministic methods has been studied for decades by the Operations Research community and relies on many consolidated results. However, deterministic mini-batch methods, i.e., algorithms that use approximation techniques to estimate first-order information, gained popularity only with the rising of machine learning between the end of 1990s and the first half of 2000s. In deep learning, deterministic methods are currently much less used than stochastic but their underlying theory remains a milestone for anyone following a deterministic approach to prove the convergence of mini-batch methods, as [208] and us.

Mini-batch gradient method with error. The convergence theory of any mini-batch deterministic method relies on two revolutionary results formalized at the end of 1990s. Considering a generalized gradient descent step with error in the form:

$$w^{k+1} = w^k - \alpha^k(d^k + e^k) \tag{3.13}$$

[1] i.e., under the condition that the starting point finds itself within a certain distance from a stationary point

Solodov [194] and Bertsekas and Tsitsiklis [16] proved the convergence of the method in two different settings, under proper assumptions on the behavior of the error sequence $\{e^k\}$. Considering their historical importance, we report the two results in the following propositions, adapting the original notation to the present thesis.

Proposition 3.2.1 (Theorem 2.1 of [194] - Bounded away from zero step-size (1998))**.** *Let $f(w)$ be L-smooth and coercive. Let w^k be updated according to the rule* (3.13) *and the following conditions hold:*

i) d^k is the exact gradient of $f(w)$ for any k, i.e., $\nabla f(w^k) = d^k$;

ii) the error sequence $\{e^k\}$ is such that, for any k, $\|e^k\| \leq \alpha^k B$ for a constant $B > 0$;

iii) the step-size α^k is such that $\lim_{k\to\infty} \alpha^k = \bar{\alpha} \in (\theta, \frac{2}{L-\theta})$, for a $\theta \in (0, \frac{1}{L}]$.

Then, if the sequence $\{f(w^k)\}$ is limited, any accumulation point $\bar{w}$ of the sequence $\{w^k\}$ is such that $\|\nabla f(\bar{w})\| \leq C\bar{\alpha}$ for a positive constant C.

Proposition 3.2.2 (Proposition 1 of [16] - Decreasing step-size (2000))**.** *Let $f(w)$ be L-smooth and coercive. Let w^k be updated according to the rule* (3.13) *and the following conditions hold:*

i) d^k is a descent direction for any k such that $-\nabla f(w^k)^T d^k \geq c_1 \|\nabla f(w^k)\|^2$ for a constant $c_1 > 0$;

ii) d^k is bounded by the true gradient, i.e., there exists a constant $c_2 > 0$ such that, for any k, $\|d^k\| \leq c_2(1 + \|\nabla f(w^k)\|)$;

iii) the error sequence $\{e^k\}$ is such that, for any k, $\|e^k\| \leq \alpha^k(p + q\|\nabla f(w^k)\|)$ for positive constants p, q;

iv) the step-size α^k converges to zero following any pre-determined schedule that satisfies (3.6)*.*

Then, $\lim_{k\to\infty} \|\nabla f(w^k)\| = 0$, and any accumulation point of $\{w^k\}$ is a stationary point for $f(w)$.

Both the results served as foundational theoretical results for the two methods we are going to review here. However, as [185], we remark that the trade-off between the assumptions required to prove the convergence and the convergence guarantees is already evident. The assumptions required in case of a step-size bounded away from zero are mild and they hold in most deep learning applications. They can also serve as a starting point to establish a minimal per-step improvement under proper conditions on the search direction and/or on the step-size. However, the convergence result is still much weaker than the obtained in [16], when the search direction is ensured to be a "good enough" descent direction (assumption i) of proposition 3.2.2). Notice also that the aforementioned assumption is the deterministic equivalent of (3.3). As remarked there, this may not hold in deep learning applications.

Incremental Gradient (IG). The first and most basic application of the results in [16, 194] is the deterministic counterpart of SGD/RR, the Incremental Gradient (IG) method. The method, reported in Algorithm 3.3, can be seen a particular case of RR, where the random permutation of the samples (step 4 of Algorithm 3.2) is always the same, i.e., $I_k = \{1, 2, \dots, P\}$ for any epoch k.

Algorithm 3.3 `Incremental Gradient (IG)`

Input: w^0, α^0
Set $k = 0$
while (stopping criterion not satisfied) **do**
  Set $\widetilde{w}_0^k = w^k$
  for $i = 1, ..., P$ **do**
    $\tilde{d}_i^k = -\nabla f_i(\widetilde{w}_{i-1}^k)$
    $\widetilde{w}_i^k = \widetilde{w}_{i-1}^k + \alpha^k \tilde{d}_i^k$
  end for
  Set $\widetilde{w}^k = \widetilde{w}_P^k$
  Update α^k
  Set $k = k + 1$
end while
Output: w^k

IG is fully deterministic and any epoch can be seen as an update step in the form of (3.13), where $d^k = \sum_{i=1}^P \nabla f_i(\widetilde{w}_{i-1}^k)$ and $e^k = \sum_{i=1}^P \left(\nabla f_i(w^k) - \nabla f_i(\widetilde{w}_{i-1}^k)\right)$. Solodov proved [194] that, applying proposition 3.2.1 to Algorithm 3.3, if all the mini-batch gradients ∇f_i are bounded above, then IG converges to an $\bar{\alpha}$-stationary point (being $\bar{\alpha}$ as in iii), proposition 3.2.1). Analogously, Bertsekas proved that if the step-size goes to zero according to (3.6) and the mini-batch gradients are such that, for some positive constants a, b, $\|\nabla f_i(w^k)\| \leq a + b\|\nabla f(w^k)\|$, then the deterministic error sequence satisfies assumption iii) of proposition 3.2.2. Hence, in this case, IG converges to a stationary point, as stated in proposition 3.2.2.
Unfortunately, while the convergence theory of IG has been deeply investigated in the case of quadratic or convex objective functions [14, 20, 151, 157, 158], to our knowledge there have not been attempts to obtain a deterministic convergence rate in the non convex setting. This makes the results in [16] amongst the most advanced in the field of convergence theory of deterministic methods. One objective of this thesis is to propose an alternative approach to find a similar convergence results but enlightening the assumptions of proposition 3.2.2.

Variants of IG. In the past years there have been some attempts to use the Bertsekas and Tsitsiklis' result as a starting point to develop novel incremental methods. The most known is certainly the averaged counterpart of IG, the Incremental Aggregated Gradient (IAG) method, reported in 3.4.
Blatt et al. proved in 2007 [20] the convergence of IAG applying the same theoretical framework developed by Bertsekas and Tsitsiklis. In particular, Blatt et al. prove that IAG converges to a stationary point, with a fixed step-size, both in the quadratic case and in the general case under the standard assumption of L-smoothness and the boundedness of the mini-batch gradients (assumption 2 of [20]). IAG has been further studied in the convex and/or quadratic setting and a deterministic convergence rate has been established in the strongly convex [81] and convex [82] case. However, to our knowledge, the only more recent result on IAG in the

Algorithm 3.4 `Incremental Aggregated Gradient (IAG)`

Input: α^0, P starting points $w_1^0, w_2^0, \dots, w_P^0$
Set $k = 0$
Set $d^k = \sum_{i=1}^{P} \nabla f_i(w_i^0)$
while (stopping criterion not satisfied) **do**
    Set $w^{k+1} = w^k - \alpha^k d^k$
    Compute $p^k =$ mod $(k+1, P)$
    Set $d^{k+1} = d^k - \nabla f_{p^k}(w^k) + \nabla f_{p^k}(w^{k+1})$
    Set $k = k + 1$
    Update α^k
end while
Output: w^k

non convex setting is the proximal modification proposed by [200], which is proved to converge to a stationary point for a sufficiently small step-size. However, as seen for SGD, the step-size must be proportional to $\mathcal{O}(\frac{1}{L})$, as explained in lemma 3.5 of [200].
Eventually, a very recent modification of IG, tailored for deep learning applications, is the Block-coordinate Incremental Gradient (BIG) method, developed by Seccia et al. [164]. BIG converges to a stationary point under the same assumptions of [16], but outperforms IG in deep learning applications and supports per-coordinate updates, which can be particulary useful in federated learning with model partitioning [64, 195].

3.3 Line-search based approaches

Recent advancements in mini-batch methods have already employed line-search techniques to enhance convergence rates and stability, particularly in noisy environments and over-parameterized models. From a theoretical perspective, the line-search approach offers the advantage of not requiring gradient information during updates within an epoch, leading to reduced memory usage, similar to standard SGD. At the same time, line-search methods can be studied relying on a well-established and generalized convergence theory, for which we refer the reader to [35]. The approach proposed in this thesis, as we will see in the forthcoming chapters, is based on performing one epoch of mini-batch method, followed by a correction in the search direction using a specific line-search technique. This approach is not completely novel and is beneficial because the line-search corrects the step-size miming the behavior of adaptive methods, but with less computational overhead, allowing us to prove convergence under milder assumptions.
Mahsereci et al. [139] are the first, in 2017, to introduce a line-search technique specifically as an additional tool for deep learning applications. The authors developed a probabilistic line-search method (algorithm 1 of their article) that combines deterministic techniques with Bayesian optimization, dynamically adjusting the step size to improve stability and reduce parameter tuning needs. Their method can be seen as a generalized mini-batch gradient algorithm with a line-search to adjust the step-size instead of tuning it with a computationally expensive grid search. The authors prove a significant advantage of their approach against SGD/RR on different FNNs (see section 2.3.1).
In 2017, a remarkable work by Gulcehre et al. [80] has been published. The authors explore the possibility of using an Armijo-like [6] line-search technique to select step sizes in the stochastic

setting. To this aim, they propose the AdaSecant algorithm, which ensures certain properties of the search direction and improves convergence speed and robustness by adapting to the local landscape of the objective function. AdaSecant outperformed Adam with different hyper-parameters setting on state-of-the-art machine learning tasks for that period.
However, the first milestone work in this field dates back to 2019, when Vaswani et al. [205] proposed to embed the Armijo line-search within the single SGD/RR update step. The proposed method, differently from [80], does not store any past gradient data to perform the Armjio step and, at the same time, achieves convergence rates under different hypotheses on the objective function. In the standard non-convex L-smooth case, SGD with Armijo line-search converges to an ε-stationary point within a maximum expected number of $\mathcal{O}(4L\varepsilon^{-2}(f(w^0)-f(w^*))$ iterations, being w^* a minimizer for problem (2.6). However, as the reader may have already noticed, the result, proved in theorem 3 of [205], relies on the initial assumption that a minimizer w^* exists. More formally, the authors assume that the model being trained is able to perfectly interpolate the input data (assumption in section 2 of the article). Furthermore, the convergence proof holds only if the step-size α^k is such that $\max_{k=0,1,\dots} \alpha^k \leq \mathcal{O}(\frac{3}{2L})$, which is only slightly better than the $\mathcal{O}(L^{-1})$ step-size we have already encountered in the SGD convergence theory.
More recently, in 2021, Loizou et al, [133] proposed to integrate Polyak step-size [168] in the SGD update step to adapt the learning rate based on current function values and gradients. The authors try to imitate the behavior of adaptive algorithm, ensuring, at the same time, efficient convergence in over-parameterized models without needing problem-specific constants or additional computational overhead. While the results on particular types of functions (e.g., convex, strongly-convex, Polyak-Lojasiewicz) marked a significant advancement in the theory, the convergence rate in the standard non-convex L-smooth setting, under assumptions 3.1.1, are not significantly different from [205].

Chapter 4

Controlled Minibatch Algorithm Framework

In this chapter we introduce the Controlled Minibatch Algorithm (CMA) framework. Firstly introduced in the doctoral thesis [185], the CMA framework has been extended and improved in [131], which is currently under a second peer-review stage at "COAP - Computational Optimization and Applications". The submitted article [131] represents the backbone of this chapter, as well as the first formalization of the CMA framework. However, the original content of the article has been enriched with a specific section (the section 4.7), realized between January and April 2024 under the guidance of Prof. Coralia Cartis, within a visiting research program at the Mathematical Institute of the University of Oxford.

As we will see throughout the chapter, the CMA framework encompasses two algorithms based on a mini-batch gradient method enhanced with sufficient decrease conditions and a derivative-free line-search procedure that ensures the global convergence to a stationary point without *any* assumption on the search direction.

The remainder of the chapter is organized as follows. After the introduction in section 4.1, we discuss the main contributions in 4.2. In section 4.3 we present a unified framework and the notation for the Mini-batch Gradient (MG) methods and we prove some general convergence properties for these methods, also reported in [78]. In section 4.4, we define the main assumptions used to prove the convergence results and we formalize the underlying idea of Controlled Mini-batch Method (CMA) framework. In section 4.5 we introduce the two algorithms, a monotone version CMA and a non-monotone version NMCMA, and for each of them provide convergence results under mild assumptions. In section 4.6 we report two class of computational experiments that we have carried out to assess CMA framework performance against both full-batch and adaptive methods. Finally, as mentioned above, in section 4.7 we propose an alternative approach based on a fixed decrease condition that, relying on some additional assumptions, ensures a deterministic bound of $\mathcal{O}(\varepsilon^{-2})$ worst-case complexity in finding an ε-stationary point. This approach is supported by additional computational tests, presented separately from the main results in section 4.6.

4.1 Introduction

We consider the finite-sum optimization problem

$$\min_{w\in\mathbb{R}^n} \qquad f(w) = \sum_{p=1}^{P} f_p(w) \tag{4.1}$$

where $f_p : \mathbb{R}^n \to \mathbb{R}$, $p = 1, \dots, P$ are continuously differentiable and possibly non-convex functions, as already stated in (2.6).

As discussed in chapter 2, problem (4.1) plays a key role in many applications and has consequently attracted researchers from many different fields. Some applications can be found in e.g. methods for solving large-scale feasibility problems by minimizing the constraints violation [143, 219] or for tackling soft constraints by a penalization approach, or Stochastic Programming problems, where the elements of the sum represent a set of realizations of a random variable that can take a very large set of potential finite values [22, 27]. Similar problems also arise in Sensor Networks, where inference problems are solved in a distributed way, namely without gathering all the data in a single data center but storing part of the data in different databases [14].

One of the most important applications which rely on the solution of very large finite-sum problems is the training phase of Machine Learning models (and the tools based on that) [27, 188] where (4.1) represents the empirical risk minimization problem, and the goal is to find the model parameters w which minimizes the average loss on the P observations. The computational per-iteration burden needed to solve problem (4.1) depends on the size of the problem itself, namely both on the number of terms P in the summation (which is large e.g. in big data setting) and on the size of the variable vector n (which is large e.g. in deep networks framework). Traditional first or higher-order methods for smooth optimization can be naively applied to solve this problem [15, 161]. However, these methods (a.k.a. *batch methods*) usually use at each iteration all the P samples to update the full vector w and become too computationally expensive when P is very large. Thus, there is an incentive to use less expensive per-iteration methods that exploit the structure of the objective function to reduce the computational burden meanwhile avoiding slowing down the convergence process.

As discussed in section 4.3, widely used approaches for solving problem (4.1) are online or mini-batch gradient methods, which cycle through the component functions using a random or deterministic order and update the iterates using the gradient of a single or a small batch of component functions. The reason for the success of online/mini-batch methods mainly relies on the different trade-offs in terms of per-iteration costs and expected per-iteration improvement in the objective function (see e.g. comments in [27]). An entire pass over all the samples P is called an epoch, and solving problem (4.1) usually requires tens to hundreds of epochs, even if a single epoch can be prohibitive when P is huge (depending on available computational time). These methods typically outperform full batch methods in numerical studies since each inner iteration makes reasonable progress towards the solution providing a good trade-off between per-iteration improvement and cost-per-iteration, possibly enabling early stopping rules which are used to avoid overfitting in machine learning.

Among the motivations for using such mini-batch methods, there is also an implicit randomness

injection to the basic gradient iteration that can lead to better performance both in training and generalization. Indeed, since the search direction may be not a descent direction, these methods possess an inherent non-monotonicity which allows them to explore larger regions, thus possibly escaping regions of attraction of inefficient local minimizers or saddle points. Moreover, it has been observed that the larger the batch size, the higher the chance to enter the basin of attraction of sharp minimizers where the function changes rapidly [100], negatively impacting the ability of the trained model to generalize on new data [90].

The milestone methods in this class of methods are the Stochastic Gradient Descent (SGD), a.k.a. Robbins-Monro algorithm, [177], which selects samples using a random with-replacement rule, and the Incremental Gradient algorithm (IG), which selects samples following a cyclic deterministic order, [14, 75, 137]. Finally, in practical implementations of mini-batch algorithms, a broadly used method is the Random Reshuffling algorithm (RR) which selects samples using a without-replacement sampling strategy, consisting in shuffling the order of samples at each epoch, and processing that permutation by batches. Empirical results show how RR methods often lead to better performance than other mini-batch methods (see e.g. [9, 25, 33, 149, 189]). Intuitively, without-replacement sampling strategies process data points more equally and are often easier and faster to implement, as it requires sequential data access, instead of random data access.

Most of the state-of-the-art methods are variants of these methods [71] that can be categorized mainly in *gradient aggregation* (non-adaptive methods) and *adaptive sampling* (adaptive methods) strategies. As discussed in the previous chapter, non-adaptive methods usually lead to good efficiency performance, but their convergence has been proved only under strong hypotheses, that are not always verified in real-world applications. Adaptive methods, instead, follow approaches that dynamically modify the number of terms f_p used to estimate ∇f in order to improve the estimates. They usually has a more stable behaviour and are more robust to the choice of hyper-parameters, but are less computationally efficient and their convergence theory is still an open research field. Theoretically, they would require the batch size to increase to infinity (see e.g. [48, 68]) or the computation of exact quantities (e.g. the gradient ∇f), or good estimates of them [22, 31].

By computing only approximations of ∇f, mini-batch methods have worse convergence properties with convergence rates usually slower than full batch methods. With the only exception of the fully deterministic methods IG o IAG (i.e., the batches are selected in a pre-fixed order), most of the convergence theory available for such algorithms provides only stochastic results, with convergence proofs or complexity bounds valid only in expected value. Moreover, when it comes to the assumptions needed to prove convergence, mini-batch methods usually require stronger conditions to be satisfied with respect to full batch gradient methods. Indeed, most of the full batch gradient methods only require Lipschitz-smoothness of each of the components f_i that allow proving the Descent Lemma, and in turn convergence for sufficiently small and fixed stepsize is proved. Instead, when analyzing mini-batch methods, in an either probabilistic or deterministic fashion, additional requirements are needed, and the stepsize must be driven to zero (which in turn allows driving the error to zero as well). The convergence analysis and

related assumptions are quite different for random or deterministic sampling.
As discussed in detail in section 3.1, when analyzing the convergence and rate of SGD-like methods, it is required either convexity and/or some kind of growth conditions on the second moments, such as assumptions 3.1.1 or the expected smoothness, as in equation (3.11). Furthermore, the stepsize must be fixed to a value depending on the Lipschitz constant of the objective function. In general, many papers have been devoted to study convergence and/or rate of convergence of modification of the basic SGD possibly weakening the required assumptions and for a more comprehensive analysis we refer the reader to section 3.1. Regarding IG-like methods, as discussed in 3.2, convergence has been proved under assumptions that holds only for a few classes of functions. Proving convergence for the RR algorithm is in general even more difficult because without-replacement sampling introduces correlations and dependencies among the sampled gradients within an epoch that are harder to analyze, as discussed in [83, 189], although recently, in [149] a fairly simpler analysis has been developed that allows having a better insight into the behavior of the RR algorithm. However, most of the convergence results of the RR algorithm and its variants require additional assumptions besides the L-smoothness, such as (strong) convexity [83, 141, 149, 150, 181, 189], some other kind of generalized convexity [1, 120] or growth conditions on the gradients of the single functions [140, 149] (see also 3.1. Therefore, although online methods are broadly applied to solve large finite-sum problems, especially in the machine learning field, convergence analysis for such algorithms in the non-convex setting is still far from satisfactory.
In this paper, we avoid to use such strong assumptions and we consider only L-smoothness and compactness of a level set which can be considered as standard assumptions in unconstrained optimization. Another critical issue in this class of methods is the practical choice of the decreasing rule for the stepsize (also known as learning rate). In practice, a preferred schedule is chosen manually by testing many different schedules in advance and choosing the one leading to the smallest training or generalization error. However, this process can be computationally heavy and can become prohibitively expensive in terms of time and computational resources incurred [208]. Adaptive gradient methods such as AdaGrad [59, 208] and Adam [56, 103] update the stepsize on-the-fly according to the gradients received along the iterations.
Quite recently, following the new emerging approach of line-search enhanced method (see section 3.3), in [140], a new approach has been proposed with new convergence results of a slight modification (the Nastya algorithm) of the basic RR method in the non-convex case has been proved. The authors' approach is closely related to our proposal in the fact that they prove effectiveness of making extra step along the basic RR direction. However, the convergence analysis of [140] is based on stochastic arguments and the convergence results are only in expected value. Furthermore, the convergence theory holds only for a stepsize proportional to L^{-1}, which can result in a very slow convergence for loss functions characterized by high values of L, such as the cross-entropy loss.
It is evident how the trade-off between theoretical properties and cost per iteration plays a crucial role in the definition of optimization methods for large-scale finite-sum problems. On the one hand, establishing convergence of an algorithm with mild assumptions requires a computational

overhead to determine exact quantities or good estimations with low variance. On the other hand, the cost per iteration of the algorithm can be reduced at the extra-cost of requiring more stringent assumptions on the objective function and, thus, reducing the range of theoretical applicability of the algorithm.

4.2 Main contributions

In this chapter, we focus on ease-controlled modifications of Random Reshuffling gradient (RR) methods, including the case of the IG method. We aim to define schemes that converge under mild assumptions, and in particular, without requiring either convexity assumptions (or similar-like) on the functions or growth and/or boundness conditions on the gradients of the single components f_p.
Most of the discussed literature is focused either on the stochastic convergence analysis for mini-batch gradient-like methods, or on the development of enhanced versions of these methods. Aiming at the latter, in this paper we present Controlled Mini-batch Algorithm (CMA) and its Non-Monotone variant (NMCMA), two RR improved versions with a safeguard rule on the sufficient decrease of the objective function. The underlying idea of CMA and NMCMA is to check, after each iteration, whether sufficient decrease condition is satisfied and, if this is not the case, to perform a derivative-free line-search to adjust the current step-size. While for a thorough analysis of the proposed algorithms and their convergence theory we refer to reader to the further sections, the main contributions of this chapter can be summarized as follows:

- we develop two RR modified algorithms jointly with a fully deterministic convergence theory, independent from the sampling distribution of the training points. Hence, our convergence theory is deterministic and does not use any probability concept;
- we prove global convergence (i.e., convergence independently from the starting point) to a stationary point under the assumptions of coercivity and Lipschitz-smoothness of the objective function without assuming any conditions on the search direction;
- we show that the convergence theory holds for any choice of the initial step-size, as there is no dependency of it from the Lipschitz constant: the step-size is automatically adjusted during the optimization process;
- we provide computational experiments on problems from some thousands up to almost 10 millions of variables, proving that CMA and NMCMA are scalable and more efficient than traditional full-batch methods and than an incremental gradient method with no safeguard rules.

In order to assess the performance of the proposed methods, numerical results have been carried out by solving the optimization problem behind the learning phase of Deep Neural Networks. We perform numerical experiments using different Deep Neural Architectures on a benchmark of varying size datasets. We compare our implementation with both a full batch gradient method (i.e. L-BFGS) and a standard implementation of IG/RR online methods, proving that the

computational effort is comparable with the corresponding online methods and that the control on the learning rate may allow a faster decrease in the objective function. The numerical results on standard test problems suggest the effectiveness of CMA-based methods in leading to better performance results than state-of-the-art batch methods (i.e. L-BFGS) and standard mini-batch methods (i.e. IG), by finding better solutions with smaller generalization errors when dealing with large problems. However, although NMCMA slightly outperforms CMA in our computational experiments, from a theoretical perspective it is not possible to claim a superior performance of CMA over NMCMA or vice versa.

4.3 A unified framework for Mini-batch Gradient (MG) methods

Given the problem (4.1), a Mini-batch Gradient (MG) method updates the variables by using a small batch of the samples $I_p \subset \{1, \dots, P\}$, that for the sake of simplicity and without loss of generality will be shortened up as a single term p. Thus, we denote

$$f_p(w) = \sum_{h_i \in I_p} f_{h_i}(w) \quad \text{and} \quad \nabla f_p(w) = \sum_{h_i \in I_p} \nabla f_{h_i}(w)$$

An iteration of the MG method, which we name `Inner_Cycle`, is obtained after a full cycle over P updates. Given a current point w^k and a current step-size ζ^k, this iteration, also called *epoch*, is defined in Algorithm 4.1.

Algorithm 4.1 `Inner_Cycle`

Input: w^k, ζ^k
Set $\widetilde{w}_0^k = w^k$
Let $I^k = \{h_1^k, \dots, h_P^k\}$ be a permutation of $\{1, \dots, P\}$
for $i = 1, ..., P$ **do**
  $\tilde{d}_i^k = -\nabla f_{h_i^k}(\widetilde{w}_{i-1}^k)$
  $\widetilde{w}_i^k = \widetilde{w}_{i-1}^k + \zeta^k \tilde{d}_i^k$
end for
Output $\widetilde{w}^k = \widetilde{w}_P^k$ and $d^k = \sum_{i=1}^{P} \tilde{d}_i^k$

Each inner step $i = 1, ..., P$ of Algorithm 4.1 is thus very cheap, involving only the computation of the gradient $\nabla f_{h_i}(\widetilde{w}_{i-1}^k)$ corresponding to a mini-batch of samples.
The general mini-batch method is reported in Algorithm 4.2.

Algorithm 4.2 Mini-batch gradient method

Given step-size $\zeta^0 \in \mathbb{R}_+$ and initial point $w^0 \in \mathbb{R}^n$
for $k = 0, 1 \dots$ **do**
  Compute $(\widetilde{w}^k, d^k) =$ `Inner_Cycle`(w^k, ζ^k)
  Set $w^{k+1} = \widetilde{w}^k = w^k + \zeta^k d^k$
  Update ζ^k according to a given rule
end for

We first observe that we have, for all $i = 1, \dots, P$,

$$\widetilde{w}_i^k = \widetilde{w}_0^k - \zeta \sum_{j=1}^{i} \nabla f_{h_j}(\widetilde{w}_{j-1}^k). \tag{4.2}$$

In Incremental Gradient (IG) [14, 17] and Random Reshuffling (RR) [83] methods, the sequence $\{w^k\}$ is obtained as $w^k = \widetilde{w}_P^{k-1}$ for $k = 1, 2, \dots$ where $\widetilde{w}_P^k$ is obtained as in Algorithm 4.1 starting with $\widetilde{w}_0^k = w^k$ so that for each $k = 1, 2, \dots$ at the end of each epoch we obtain:

$$\widetilde{w}_P^k = \widetilde{w}_0^k - \zeta \sum_{j=1}^{P} \nabla f_{h_j}(\widetilde{w}_{j-1}^k)$$

In IG or Shuffle-Once, samples are always chosen in the same order (which is fixed in a deterministic or random way respectively) per epoch, e.g. $h_1 = 1, \dots, h_P = P$ for all k, whereas in RR methods $h_1, \dots, h_P$ are sampled without replacement from the set of P terms available.
Convergence properties of these methods are studied with respect to the behaviour of the outer sequence $\{w^k\}$.
In this section, we aim to derive properties of the sequence $\{\tilde{w}_i^k\}$ generated by Algorithm 4.1 when a sequence $\{w^k\} \subseteq \mathbb{R}^n$ and a sequence of non negative scalars ζ^k are given and each element w^k of the sequence represents the starting point $\widetilde{w}_0^k$ of the inner MG iterations.
Depending on the assumptions on the sequences $\{w^k\}$ and $\{\zeta^k\}$ we can prove different properties satisfied by the sequences $\{\widetilde{w}^k\}$, $\{\widetilde{w}_i^k\}$ for $i = 1, \dots, P$, and $\{d^k\}$ generated by the overall scheme.
Let us now make the following assumption on the objective function:

Assumption 4.3.1. *The gradients ∇f_p are Lipschitz continuous for each $p = 1, \dots, P$, namely there exists $L > 0$ such that*

$$\|\nabla f_p(u) - \nabla f_p(v)\| \leq L\|u - v\| \qquad \forall\ u, w \in \mathbb{R}^n$$

Under Assumption 4.3.1, Algorithm 4.1 has the following property.

Lemma 4.3.2. *Let Assumption 4.3.1 hold and let w^k and ζ^k be a given point and stepsize. Let $\widetilde{w}_i^k$ be the points generated by Algorithm 4.1. Then we have, for all $i = 1, \dots, P$,*

$$\|\widetilde{w}_i^k - \widetilde{w}_0^k\| \quad \leq \quad \zeta \left(L \sum_{j=1}^{i} \|\widetilde{w}_{j-1}^k - \widetilde{w}_0^k\| + \sum_{j=1}^{i} \|\nabla f_{h_j}(\widetilde{w}_0^k)\| \right). \tag{4.3}$$

Proof. For the sake of simplicity we consider the permutation $I^k = \{1 \dots P\}$. From Algorithm 4.1, we have that, $\widetilde{w}_1 = \widetilde{w}_0 - \zeta \nabla f_1(\widetilde{w}_0)$. Hence, we obtain

$$\|\widetilde{w}_1 - \widetilde{w}_0\| = \zeta \|\nabla f_1(\widetilde{w}_0)\| \tag{4.4}$$

and (4.3) holds for $i = 1$.

Now, using the Lipschitz assumption we can write for all i

$$\begin{aligned}
\|\widetilde{w}_i - \widetilde{w}_0\| &= \left\| -\zeta \sum_{j=1}^{i} \nabla f_j(\widetilde{w}_{j-1}) \right\| \\
&= \zeta \left\| \sum_{j=1}^{i} (\nabla f_j(\widetilde{w}_{j-1}) - \nabla f_j(\widetilde{w}_0) + \nabla f_j(\widetilde{w}_0)) \right\| \\
&\leq \zeta \sum_{j=1}^{i} \|\nabla f_j(\widetilde{w}_{j-1}) - \nabla f_j(\widetilde{w}_0) + \nabla f_j(\widetilde{w}_0)\| \\
&\leq \zeta \sum_{j=1}^{i} (\|\nabla f_j(\widetilde{w}_{j-1}) - \nabla f_j(\widetilde{w}_0)\| + \|\nabla f_j(\widetilde{w}_0)\|) \\
&\leq \zeta \sum_{j=1}^{i} (L \|\widetilde{w}_{j-1} - \widetilde{w}_0\| + \|\nabla f_j(\widetilde{w}_0)\|)
\end{aligned}$$

which concludes the proof. □

Let us now assume that we have assigned sequences $\{w^k\}$ and $\{\zeta^k\}$. We start by proving that, when both $\{w^k\}$ and $\{\zeta^k\}$ are bounded, then the inner sequences $\{\widetilde{w}_i^k\}$ are bounded too for all $i = 1, \ldots, P$.

Proposition 4.3.3. *Let Assumption 4.3.1 hold and let $\{w^k\}$ and $\{\zeta^k\}$ be bounded sequences of points and positive scalars. Let $\{\widetilde{w}_i^k\}$ for $i = 1, \ldots, P$ be the sequences of points generated by Algorithm 4.1.*
Then, the sequences $\{\widetilde{w}_i^k - w^k\}$, for $i = 1, \ldots, P$, are all bounded.

Proof. The proof is by induction. Using the definition of $\widetilde{w}_i^k$ as in step 5 and step 6 of Algorithm 4.1 we can write for $i = 1$:

$$\|\widetilde{w}_1^k - \widetilde{w}_0^k\| = \|\widetilde{w}_1^k - w^k\| = \zeta^k \|\nabla f_{h_1^k}(w^k)\|$$

which, considering the boundedness of w^k, ζ^k and continuity of $\nabla f_{h_1^k}$, proves that $\{\widetilde{w}_1^k - \widetilde{w}_0^k\}$ is bounded.
Thus, we assume that $\{\widetilde{w}_j^k - w^k\}$ are bounded for all $j = 1, \ldots, i-1$, and prove that $\{\widetilde{w}_i^k - w^k\}$ is bounded as well. In fact, thanks to Lemma 4.3.2 we can write

$$\|\widetilde{w}_i^k - w^k\| \leq \zeta^k \left(L \sum_{j=1}^{i} \|\widetilde{w}_{j-1}^k - w^k\| + \sum_{j=1}^{i} \|\nabla f_{h_j^k}(w^k)\| \right),$$

which, by the induction assumption, boundedness of w^k and ζ^k and the continuity of each ∇f_i, proves boundedness of $\{\widetilde{w}_i^k - w^k\}$ for all i. □

We now prove that when the stepsize ζ^k is driven to zero, the bounded sequences $\{\widetilde{w}_i^k\}$ converge to the same limit point of the outer sequence $\{w^k\}$ and that the sequence of directions $\{d^k\}$ converges to $-\nabla f(\bar{w})$. This result is at the basis of the convergence of IG and RR methods.

Proposition 4.3.4. *Let Assumption 4.3.1 hold. Assume also that $\{w^k\}$ is bounded and that $\lim_{k\to\infty} \zeta^k = 0$. Let $\{\widetilde{w}_i^k\}$ and $\{d^k\}$ be the sequences of points and directions produced by Algorithm 4.1. Then, for any limit point $\bar{w}$ of $\{w^k\}$ a subset of indices K exists such that*

$$\lim_{k\to\infty,k\in K} w^k = \bar{w}, \tag{4.5}$$

$$\lim_{k\to\infty,k\in K} \zeta^k = 0, \tag{4.6}$$

$$\lim_{k\to\infty,k\in K} \widetilde{w}_i^k = \bar{w}, \quad \textit{for all } i = 1,\dots,P \tag{4.7}$$

$$\lim_{k\to\infty,k\in K} d_k = -\nabla f(\bar{w}). \tag{4.8}$$

Proof. Let us consider any limit point $\bar{w}$ of $\{w^k\}$. Then, recalling that $\lim_{k\to\infty} \zeta^k = 0$, we have that a subset of indices $\bar{K}$ exists such that

$$\lim_{k\to\infty,k\in \bar{K}} w^k = \bar{w},$$
$$\lim_{k\to\infty,k\in \bar{K}} \zeta^k = 0.$$

Now, for every iteration index k, let I^k denote a permutation of the set of indices $\{1,\dots,P\}$. Since P is finite, the number of such permutations is finite. Then, we can extract an infinite subset of indices $K \subseteq \bar{K}$ such that

$$\lim_{k\to\infty,k\in K} w^k = \bar{w} \tag{4.9}$$

$$\lim_{k\to\infty,k\in K} \zeta^k = 0 \tag{4.10}$$

$$I_k = \bar{I} = \{\bar{h}_1,\dots,\bar{h}_P\},\ \forall\ k \in K$$

Thus, (4.9) and (4.10) prove, respectively, (4.5) and (4.6). From the definition of Algorithm 4.1, and applying Proposition 4.3.2 we have that for $i = 1,\dots,P$ and $k \in K$, we can write

$$\|\widetilde{w}_i^k - w^k\| \le \zeta^k \left(L \sum_{j=1}^{i} \|\widetilde{w}_{j-1}^k - w^k\| + \sum_{j=1}^{i} \|\nabla f_{\bar{h}_j}(w^k)\| \right). \tag{4.11}$$

Now, to prove (4.7), we proceed by induction on i. Consider first $i = 1$ so that

$$\|\widetilde{w}_1^k - w^k\| \le \zeta^k \|\nabla f_{\bar{h}_1}(w^k)\|$$

Taking the limit for $k \to \infty$, $k \in K$ recalling that $\{w^k\}$ is bounded and continuity of $\nabla f_{\bar{h}_1}$, we obtain

$$\lim_{k\to\infty,k\in K} \|\widetilde{w}_1^k - w^k\| = 0$$

which means, recalling (4.9), that

$$\lim_{k\to\infty,k\in K} \widetilde{w}_1^k = \bar{w}. \tag{4.12}$$

By induction on i, let us assume that, for all $j = 1, \dots, i-1$,

$$\lim_{k\to\infty, k\in K} \|\widetilde{w}_j^k - w^k\| = 0 \tag{4.13}$$

By taking the limit for $k \to \infty$, $k \in K$, in (4.11), and recalling the induction assumption (4.13), boundedness of $\{w^k\}$ and continuity of $\nabla f_{\bar{h}_j}$ for all j, we have that

$$\lim_{k\to\infty, k\in K} \|\widetilde{w}_i^k - w^k\| = 0$$

which means, recalling (4.9), that

$$\lim_{k\to\infty, k\in K} \widetilde{w}_i^k = \bar{w}. \tag{4.14}$$

Then, (4.12) and (4.14) prove (4.7).
In order to prove (4.8), let us recall the definition of d^k, for $k \in K$, i.e.

$$d^k = -\sum_{i=1}^{P} \nabla f_{\bar{h}_i}(\widetilde{w}_{i-1}^k).$$

Then, (4.8) follows by recalling (4.7) and the continuity of $\nabla f_{\bar{h}_i}$, for $i = 1, \dots, P$. □

Remark 4.3.5. *In Algorithm 4.1 the* for cycle *at step 4 can be modified using in place of P a value $P^k \leq P$ for all k and such that $P^k = P$ for infinitely many iterations. Indeed, under this condition we have that Proposition 4.3.4 still holds on the further subsequence $\{k : \ P^k = P\}$.*

Proposition 4.3.4 shows that, given an "external" converging sequence $\{w^k\}$ and driving the stepsize $\zeta^k \to 0$, the "attached" sequences $\widetilde{w}_i^k$ for $k \in K$ remains in a neighborhood of w^k and converge to the same limit point $\bar{w}$. Thus it proves that the direction d^k used for updating the point w^k tends in the limit to be the gradient itself. This implies that the error

$$\left\| \nabla f(w^k) - \sum_{i=1}^{P} \nabla f_{h_i}(\widetilde{w}_{i-1}^k) \right\|$$

is going to zero in the limit.
The error going to zero has been exploited to prove convergence of deterministic IG and RR methods requiring strong assumptions on the objective functions as done in most papers [17, 24, 81, 165, 184]. Indeed, these strong assumptions are not satisfied in many important machine learning models.
In Proposition 4.3.4, these assumptions on the objective function are replaced by a strong assumption on the sequence $\{w^k\}$, which has to remain bounded. In the next sections, we show how to enforce this assumption by embedding Algorithm 4.1 in a more specific framework producing the sequence $\{w^k\}$.
Another crucial aspect in methods using diminishing stepsize rule is how to drive the stepsize $\zeta^k \to 0$. Indeed many heuristic rules have been defined [71] to avoid the stepsize decreasing too fast and thus slowing down convergence. The algorithms that we define in the next sections also allow tackling this aspect in an automatic way by controlling the reduction of the stepsize.

In particular, the price to pay for lightening the assumptions requires a limited additional computational effort by checking periodically a sufficient reduction of the objective function.

4.4 Main assumptions and underlying idea: the CMA framework

Throughout the paper, we require only the following assumptions on the objective function $f(w)$ and its gradient.

Assumption 4.4.1. *The function f is coercive, i.e. all the level sets*

$$\mathcal{L}(w_0) = \{w \in \Re^n : \ f(w) \leq f(w_0)\}$$

are compact for any $w_0 \in \Re^n$.

Assumption 4.4.2. *The gradients ∇f_p are Lipschitz continuous for each $p = 1, \ldots, P$, namely there exists $L > 0$ such that*

$$\|\nabla f_p(u) - \nabla f_p(v)\| \leq L\|u - v\| \qquad \forall \ u, w \in \mathbb{R}^n$$

Assumption 4.4.1 enforces the existence of a global solution. It is trivially satisfied e.g. in training problems for Deep Neural Networks (DNNs) when using a ℓ_2 regularization term for the empirical error. Assumption 4.4.2 is a standard and mild assumption in gradient-based methods.

The idea at the basis of the proposed schemes, firstly proposed in [185], consists in perturbing as less as possible the basic iteration of the RR/IG gradient algorithm to retain their light computational effort and the low memory requirement per iteration. The ingredients used are indeed the basic mini-batch iteration and tools derived from derivative-free methods.

In particular, we define two algorithmic schemes in which the IG/RR iteration is controlled by using a watchdog rule and a derivative-free linesearch that activates only sporadically to guarantee convergence. The two algorithms differ in the watchdog and the linesearch procedures, which are performed using either a monotonic or a non-monotonic rule. They also allow controlling the updating of the learning rate used in the main IG/RR iteration, avoiding the use of pre-set rules that may drive it to zero too fast, thus overcoming another challenging aspect in implementing online methods, which is the updating rule of the stepsize. We prove convergence under the mild assumption of Lipschitz continuity of the gradients of the component functions, without assuming any further hypotheses on the search direction d^k.

In Algorithm 4.3, we report the rough algorithmic scheme of the proposed algorithms. The underlying idea of CMA and NMCMA is to control both the iteration and the stepsize of RR mini-batch gradient iteration. Indeed, a trial point $\widetilde{w}^k$ is obtained by Algorithm 4.1, as for any other mini-batch methods. However, differently from what happens in the standard mini-batch methods (see Algorithm 4.2), the trial point is accepted only when a *watchdog*-flavoured condition [37] is verified and a sufficient decrease of the objective function is attained (represented with the checking sub-routine `SufficientDecrease()`). When this is not satisfied,

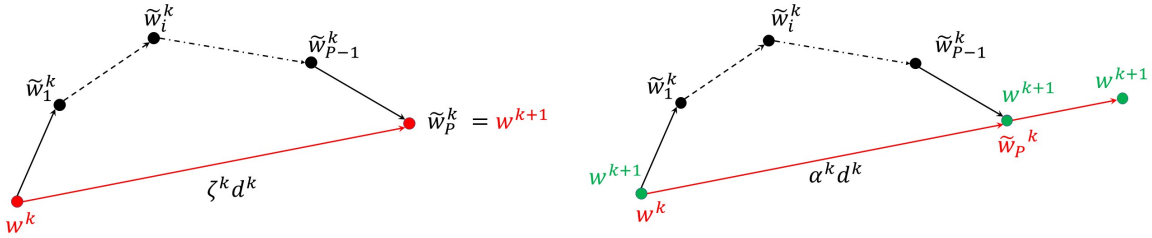
we resort to a linesearch procedure (`DerivativeFreeLineSearch()` in Algorithm 4.3) to enforce convergence. However, since we want to avoid computing the gradient even periodically, we adopt a derivative-free-type linesearch [77, 115, 136], which is used to define the new iteration and to possibly update the stepsize used in the mini-batch cycle. Therefore, the extra cost to pay for lightening the assumptions needed to prove convergence is given by the function evaluation needed at the end of each epoch and possibly in the linesearch. We remark that CMA and NMCMA require different watchdog and linesearch updating rules and are not comparable from a theoretical point of view in the final effort required. However, as it can be seen in Step 9 of Algorithm 4.3, they both produce a point w^{k+1}, which is a modified IG/RR point. In particular, if the watchdog condition is satisfied, we get exactly the IG/RR point, otherwise we either take a longer step along the search direction d^k or restart. In this last case we admit the possibility of wasting one epoch of computational time.

Algorithm 4.3 Sketch of algorithms CMA and NMCMA

1: Let $\zeta^0 > 0$, and $w^0 \in \mathbb{R}^n, k = 0$
2: **for** $k = 0, 1, 2 \dots$ **do**
3: Compute $(\widetilde{w}^k, d^k) =$ `Inner_Cycle`(w^k, ζ^k)
4: **if** *sufficient decrease is attained* **then**
5: Set $\zeta^{k+1} = \zeta^k$ and $\alpha^k = \zeta^k$
6: **else**
7: Compute $\widetilde{\alpha}^k$ by a *linesearch procedure (LS)*
8: Set $\alpha^k = \begin{cases} \widetilde{\alpha}^k & \text{if LS successful} \\ 0 & \text{otherwise} \end{cases}$ $\zeta^{k+1} = \begin{cases} \zeta^k & \text{if LS is successful} \\ \theta\zeta^k \end{cases}$
9: **end if**
10: $w^{k+1} = w^k + \alpha^k d^k$
11: **end for**

4.5 The proposed algorithms: CMA and NMCMA

In this section, we introduce the two algorithms: Controlled Mini-batch Algorithm (CMA) and Non-Monotone Controlled Mini-batch Algorithm (NMCMA). For each of them we prove convergence properties under specific light assumptions. Indeed, the aim of CMA-based algorithms is to define schemes that require a low increased computational cost with respect to traditional RR or IG algorithms, and allow to control convergence without requiring strong assumptions on the objective function. In particular, neither convexity, nor strong growth conditions on the gradients of each f_p and on the full gradient, as in [14, 83, 165] are required, but we use Assumptions 4.4.1 (assumed always satisfied) and 4.4.2 (recalled when needed).
The underlying idea of the Controlled mini-batch Algorithm schemes consists in defining the "external" sequence $\{w^k\}$ by trying to accept as much as possible the points generated by Algorithm 4.1, namely setting $w^{k+1} = \widetilde{w}_P^k$ for *as many iterations k as possible*. The acceptance of the trial point $\widetilde{w}_P^k$ is controlled by means of both a watchdog approach and, in extreme cases, a derivative-free linesearch procedure involving a few evaluations of just the objective function.



(a) MG (b) CMA

Figure 4.1: a) MG gradient outer iteration; b) Controlled mini batch outer iteration: w^{k+1} is selected as on one of the green points.

In particular, the outer iterations w^k are defined as:

$$w^{k+1} = w^k + \alpha^k d^k \tag{4.15}$$

where $d^k = -\sum_{i=1}^{P} \nabla f_{h_i^k}(\widetilde{w}_{i-1}^k)$ is the search direction obtained in the `Inner_Cycle` and the stepsize can take the values

$$\alpha^k = \begin{cases} \zeta^k & \text{the trial point } \widetilde{w}_P^k \text{ is accepted} \\ 0 & \text{restart from the last point } w^k \\ > \zeta^k & \text{take a longer step along } d^k \end{cases}$$

In Figure 4.1, we report a picture of the outer iterations in the MG gradient method (as in Algorithm 4.2) and in the Controlled Mini Batch scheme (Algorithm 4.3). We highlight how the outer iteration in Controlled mini-batch Algorithms allows an extrapolation step along d^k but also the possibility of having a restart, thus discarding completely the iterations done in the last epoch. The computational experience reported in section 4.6 shows that the restart step appears very seldom and does not affect the overall performance. As already mentioned, in [140], a result on the convergence of a modification of the RR method which considers a similar extrapolation step along the direction d^k has been provided with the Nastya algorithm. However, the main difference between Nastya and CMA frameworks relies on the hypothesis needed to prove convergence: we do not need to know to any extent the Lipschitz constant, so that ζ^k is adjusted in an automatic way. Moreover, we also include the possibility of restarting the iteration.

Further, we also have the chance to control the stepsize ζ^k used in the `Inner_Cycle`. Algorithm 4.1 constitutes the inner iterations which defines trial points $\widetilde{w}^k$ and search directions d^k. Eventually, the stepsize ζ^k used in in the `Inner_Cycle` is driven to zero. However, the decreasing rule depends on watchdog rule on the iteration, and it may happen that $\zeta^{k+1} = \zeta^k$. This avoid the inner stepsize to decrease too fast and slowing down the practical convergence.

We call the method "Controlled mini-batch Algorithm" since at each "outer" iteration the watch-

dog approach is leveraged to apply mini-batch iterations as much as possible, while reductions of the objective function over the iterations and of the stepsize ζ^k are checked and, when needed, enforced via a derivative-free linesearch procedure.
In the following, two different versions of the algorithm are introduced that use either a monotone or non-monotone linesearch procedures thus forcing either a monotone or non monotone decreasing sequence $\{f(w^k)\}$. Both the two line-search procedures use as starting stepsize the value of ζ^k and envisage the possibility of extrapolating, namely of taking larger steps along the direction d^k. For the sake of simplicity, we will refer to these two methods as CMA and NMCMA respectively.
The two methods differ not only in the DFLS but also in the acceptance rules of the trial points and in the reduction of the stepsize. From a theoretical point of view we cannot prove the superiority of one w.r.t. the other, while, from a practical perspective, they can explore different regions of the searching space and can achieve different numerical performances.

4.5.1 Monotone Controlled mini-batch Algorithm

In this section, we present the Controlled mini-batch Algorithm (CMA) which is a *monotonically* decreasing method. Following the general description, a trial point $\widetilde{w}^k$ and a direction d^k are obtained by applying Algorithm 4.1 with a constant stepsize ζ^k.
CMA checks if a sufficient reduction in the trial point is satisfied, namely $f(\widetilde{w}^k) - f(w^k) \leq -\gamma\zeta^k$. In this case, the trial point is accepted, and the stepsize ζ^k is left unchanged.
Otherwise, CMA checks whether the direction d^k can be considered as a good estimate of a descent direction or if we are close enough to convergence. In these cases, the inner stepsize ζ^k can be reduced, and the direction can possibly be used within an Extrapolation Derivative-Free Linesearch (EDFL). When used, EDFL computes the outer stepsize α^k used to update the iteration. It checks first for a sufficient monotonic reduction using a derivative-free condition $f(\widetilde{w}^k) - f(w^k) \leq -\gamma\zeta^k\|d^k\|^2$, and when satisfied, it employs an extrapolation procedure for taking longer steps than ζ^k along the direction. The scheme of the Linesearch procedure EDFL is provided in Algorithm 4.4 [136].

Algorithm 4.4 Extrapolation Derivative-Free Linesearch (EDFL)

Input $(w^k, d^k, \zeta^k; \gamma, \delta)$: $w^k \in \mathbb{R}^n, d^k \in \mathbb{R}^n, \zeta^k > 0$, $\gamma \in (0,1), \delta \in (0,1)$
Set $\alpha = \zeta^k$
if $f(w^k + \alpha d^k) > f(w^k) - \gamma\alpha\|d^k\|^2$ **then**
    Set $\alpha^k = 0$ and **return**
end if
while $f(w^k + (\alpha/\delta)d^k) \leq \min\{f(w^k) - \gamma(\alpha/\delta)\|d^k\|^2, f(w^k + \alpha d^k)\}$ **do**
    Set $\alpha = \alpha/\delta$
end while
Set $\alpha^k = \alpha$ and **return**
Output: α^k

The scheme of the CMA is provided in Algorithm 4.5. We note that the stepsize returned by the EDFL is called $\widetilde{\alpha}^k$ and it is used together with ζ^k and d^k to check several different conditions on

the sufficient reduction of the objective function. The final stepsize α^k is set to zero only when all the conditions are not met, and the produced point would be outside the Level set $\mathcal{L}(w^0)$. In this way we try to allow acceptance of the trial point as much as possible, thus perturbing the classical RR or IG iteration as few times as possible. Also, CMA allows a further degree of freedom in choosing w^{k+1} w.r.t. to (4.15) that can actually be set to any point such that $f(w^{k+1}) \leq f(w^k + \alpha^k d^k)$

Algorithm 4.5 Controlled mini-batch Algorithm (CMA)

Set $\zeta^0 > 0, \theta \in (0,1), \tau > 0$, $\gamma \in (0,1), \delta \in (0,1)$
Let $w^0 \in \mathbb{R}^n, k = 0$
for $k = 0, 1, 2 \ldots$ **do**
  Compute $(\widetilde{w}^k, d^k) =$ `Inner_Cycle`(w^k, ζ^k)
  if $f(\widetilde{w}^k) \leq f(w^k) - \gamma\zeta^k$ **then**
    Set $\zeta^{k+1} = \zeta^k$ and $\alpha^k = \zeta^k$
  else
    if $\|d^k\| \leq \tau\zeta^k$ **then**
      Set $\zeta^{k+1} = \theta\zeta^k$ and $\alpha^k = \begin{cases} \zeta^k & \text{if } f(\widetilde{w}^k) \leq f(w^0) \\ 0 & \text{otherwise} \end{cases}$
    else
      $\widetilde{\alpha}^k = EDFL(w^k, d^k, \zeta^k; \gamma, \delta)$
      if $\widetilde{\alpha}^k \|d^k\|^2 \leq \tau\zeta^k$ **then**
        Set $\zeta^{k+1} = \theta\zeta^k$ and $\alpha^k = \begin{cases} \widetilde{\alpha}^k & \text{if } \widetilde{\alpha}^k > 0 \\ \zeta^k & \text{if } \widetilde{\alpha}^k = 0 \text{ and } f(\widetilde{w}^k) \leq f(w^0) \\ 0 & \text{otherwise} \end{cases}$
      else
        Set $\zeta^{k+1} = \zeta^k$, and $\alpha^k = \widetilde{\alpha}^k$
      end if
    end if
  end if
  Set $y^k = w^k + \alpha^k d^k$
  Select w^{k+1} such that $f(w^{k+1}) \leq f(y^k)$
end for

CMA Convergence analysis

We first recall that the Linesearch procedure in Algorithm 4.4 is well defined; namely it terminates in a finite number of iterations.

Lemma 4.5.1. *Algorithm 4.4 is well defined, i.e. it determines in a finite number of steps a scalar α^k such that*

$$f(w^k + \alpha^k d^k) \leq f(w^k) - \gamma\alpha^k \|d^k\|^2. \tag{4.16}$$

Proof. First of all, we prove that Algorithm 4.4 is well-defined. In particular, we show that the while loop cannot infinitely cycle. Indeed, if this was the case, that would imply

$$f\left(w^k + \frac{\zeta^k}{\delta^j} d^k\right) \leq f(w^k) - \gamma\left(\frac{\zeta^k}{\delta^j}\right)\|d^k\|^2 \quad \forall j = 1, 2, \ldots$$

so that for $j \to \infty$ we would have $(\zeta^k/\delta^j) \to \infty$ which contradicts the compactness of $\mathcal{L}(w^0)$ and continuity of f.
In order to prove (4.16), we consider separately the two cases i) $\alpha^k = 0$ and ii) $\alpha^k > 0$. In the first case, (4.16) is trivially satisfied.
In case ii), the condition at step 4 implies

$$f(w^k + \zeta^k d^k) \leq f(w^k) - \gamma \zeta^k \|d^k\|^2.$$

Then, if $\alpha^k = \zeta^k$, the condition is satisfied. Otherwise, if $\alpha^k > \zeta^k$, the stopping condition of the while loop, along with the fact that we already proved that the while loop could not infinitely cycle, is such that we have

$$f(w^k + \alpha^k d^k) \leq f(w^k) - \gamma \alpha^k \|d^k\|^2$$

which again proves (4.16), thus concluding the proof. □

We now prove that the sequence generated by CMA remains in the level set $\mathcal{L}(w^0)$ which is compact by Assumption 4.4.1, thus satisfying the assumption required in Proposition 4.3.4.

Lemma 4.5.2. *Let $\{w^k\}$ be the sequence of points produced by algorithm CMA. Then, $\{w^k\} \subseteq \mathcal{L}(w^0)$.*

Proof. By definition of algorithm CMA and recalling Lemma 4.5.1, for every iteration k, we have that $f(w^{k+1}) \leq f(y^k)$ (step 20). Furthermore, exactly one of the following cases happens

i) step 7 is executed, i.e. $f(y^k) \leq f(w^k) - \gamma \zeta^k$;

ii) step 16 is executed, i.e. $f(y^k) \leq f(w^k) - \gamma \alpha^k \|d^k\|^2$;

iii) step 10 is executed, then, we have that

- $f(y^k) \leq f(w^0)$ or
- $f(y^k) = f(w^k)$;

iv) step 14 is executed, then, we have that

- $f(y^k) = f(w^k)$ or
- $f(y^k) = f(\widetilde{w}^k) \leq f(w^0)$ or
- $f(y^k) = f(w^k + \alpha^k d^k) \leq f(w^k + \zeta^k d^k) = f(\widetilde{w}^k) \leq f(w^k) - \gamma \zeta^k \|d^k\|^2$.

Then, for every k, we have in particular that either $f(w^{k+1}) \leq f(w^0)$ or $f(w^{k+1}) \leq f(w^k)$. Now, let us split the iteration sequence into two subsets, namely K and $\bar{K}$, such that

$$K = \{k \in \mathbb{N} : k \geq 0, f(w^k) \leq f(w^0)\},$$
$$\bar{K} = \{k \in \mathbb{N} : k > 0, f(w^k) \leq f(w^{k-1})\}.$$

Note that, in particular, $0 \in K$, i.e. $K \neq \emptyset$. It obviously results that

$$\{w^k\}_K \subseteq \mathcal{L}(w^0). \tag{4.17}$$

On the other hand, for every $k \in \bar{K}$, let m_k be the biggest index such that $m_k < k$ and $m_k \in K$. Since $K \neq \emptyset$, m_k is always well-defined. Then, by definition of K, $\bar{K}$ and m_k, we can write

$$f(w^k) \leq f(w^{k-1}) \leq \cdots \leq f(w^{m_k}) \leq f(w^0)$$

Hence, when $k \in \bar{K}$ we still have that $f(w^k) \leq f(w^0)$, so that

$$\{w^k\}_{\bar{K}} \subseteq \mathcal{L}(w^0). \tag{4.18}$$

Then, the proof is concluded recalling (4.17) and (4.18).

□

Now we show that the inner stepsize ζ^k goes to zero as $k \to \infty$.

Proposition 4.5.3. *Let $\{\zeta^k\}$ be the sequence of steps produced by Algorithm CMA, then*

$$\lim_{k\to\infty} \zeta^k = 0.$$

Proof. In every iteration, either $\zeta^{k+1} = \zeta^k$ or $\zeta^{k+1} = \theta\zeta^k < \zeta^k$. Therefore, the sequence $\{\zeta^k\}$ is monotonically non-increasing. Hence, it results

$$\lim_{k\to\infty} \zeta^k = \bar{\zeta} \geq 0.$$

Let us suppose, by contradiction, that $\bar{\zeta} > 0$. If this were the case, there should be an iteration index $\bar{k} \geq 0$ such that, for all $k \geq \bar{k}$, $\zeta^{k+1} = \zeta^k = \bar{\zeta}$. Namely, for all iterations $k \geq \bar{k}$, step 7 or 16 are always executed. Then, for all $k \geq \bar{k}$ we have $f(w^{k+1}) \leq f(w^k)$.
Let us now denote by

$$\begin{aligned} K' &= \{k :\ k \geq \bar{k},\ \text{and step 7 is executed}\} \\ K'' &= \{k :\ k \geq \bar{k},\ \text{and step 16 is executed}\} \end{aligned}$$

Let us first suppose that K' is infinite. Then, for $k \in K'$ and $k \geq \bar{k}$, we would have that infinitely many times

$$\gamma\bar{\zeta} \leq f(w^k) - f(w^{k+1}).$$

Then, taking the limit for $k \to \infty$ and $k \in K'$, we get a contradiction with the compactness of the level set and the continuity of the function f.

On the other hand, let us suppose that K'' is infinite. Then, thanks to Lemma 4.5.1, we would have that, for $k \in K''$,

$$f(w^{k+1}) \leq f(w^k + \alpha^k d^k) \leq f(w^k) - \gamma\alpha^k \|d^k\|^2.$$

By taking the limit in the above relation and recalling the compactness of the level set $\mathcal{L}(w^0)$ and continuity of f, we have that $f(w^k) \to \bar{f} \in \mathbb{R}, k \to \infty$ and we would obtain

$$\lim_{k\to\infty} \alpha^k \|d^k\|^2 = 0.$$

But then, for $k \in K''$ and sufficiently large, it would happen that

$$\alpha^k \|d^k\|^2 \leq \tau\bar{\zeta}$$

which means that Algorithm CMA would execute step 10 setting $\zeta^{k+1} = \theta\bar{\zeta}$ thus decreasing ζ^{k+1} below $\bar{\zeta}$. This contradicts our initial assumption and concludes the proof. □

We are now ready to provide the main convergence result for CMA.

Proposition 4.5.4. *Let Assumption 4.4.2 hold and let $\{w^k\}$ be the sequence of points produced by Algorithm CMA. Then, $\{w^k\}$ admits limit points and (at least) one of them is stationary.*

Proof. By the compactness of $\mathcal{L}(w^0)$ and considering that $w^k \in \mathcal{L}(w^0)$ for all k, we know that $\{w^k\}$ admits limit points.
Now, let us introduce the following set of iteration indices

$$K = \{k : \ \zeta^{k+1} = \theta\zeta^k\}$$

and note that, since $\lim_{k\to\infty} \zeta^k = 0$, K must be infinite. Let $\bar{w}$ be any limit point of the subsequence $\{w^k\}_K$, then, by Proposition 4.3.4, a subset $\bar{K} \subseteq K$ exists such that

$$\lim_{k\to\infty, k\in\bar{K}} w^k = \bar{w}, \tag{4.19a}$$

$$\lim_{k\to\infty, k\in\bar{K}} d^k = -\nabla f(\bar{w}), \tag{4.19b}$$

$$\lim_{k\to\infty, k\in\bar{K}} \zeta^k = 0 \tag{4.19c}$$

Then, let us now split the set $\bar{K}$ into two further subsets, namely,

$$\begin{aligned} K_1 &= \{k \in \bar{K} : \ \|d^k\| \leq \tau\zeta^k\}, \\ K_2 &= \{k \in \bar{K} \setminus K_1 : \widetilde{\alpha}^k \|d^k\|^2 \leq \tau\zeta^k\}. \end{aligned}$$

Note that, K_1 and K_2 cannot be both finite.
First, let us suppose that K_1 is infinite. Then, the definition of K_1, (4.19b) and (4.19c) imply that

$$\lim_{k\to\infty, k\in K_1} \|d^k\| = \|\nabla f(\bar{w})\| = 0$$

thus concluding the proof in this case.

Now, let us suppose that K_2 is infinite. In this case, we proceed by contradiction and assume

that $\bar{w}$ is not stationary, i.e. $\|\nabla f(\bar{w})\| > 0$. By the definition of K_2 and (4.19c), we obtain

$$\lim_{k\to\infty, k\in K_2} \widetilde{\alpha}^k \|d^k\|^2 = 0,$$

which, recalling (4.19b) and the fact that $\|\nabla f(\bar{w})\| > 0$, yields

$$\lim_{k\to\infty, k\in K_2} \widetilde{\alpha}^k = 0. \tag{4.20}$$

Now, let us further partition K_2 into two subsets, namely

i) $\bar{K}_2 = \{k \in K_2 : \ \widetilde{\alpha}^k = 0\}$

ii) $\widehat{K}_2 = \{k \in K_2 : \ \widetilde{\alpha}^k > 0\}$.

Let us first suppose that $\bar{K}_2$ is infinite. This means that, by the definition of algorithm EDFL,

$$\frac{f(w^k) - f(w^k + \zeta^k d^k)}{\zeta^k} < \gamma \|d^k\|^2.$$

Then, by the Mean-Value Theorem, a number $\hat{\alpha}^k \in (0, \zeta^k)$ exists such that

$$-\nabla f(w^k + \hat{\alpha}^k d^k)^T d^k < \gamma \|d^k\|^2.$$

Taking the limit in the above relation, considering (4.19a), (4.19b) and (4.19c), we obtain

$$\|\nabla f(\bar{w})\|^2 \leq \gamma \|\nabla f(\bar{w})\|^2$$

which, recalling that $\gamma \in (0, 1)$, gives

$$\|\nabla f(\bar{w})\| = 0,$$

which contradicts $\|\nabla f(\bar{w})\| > 0$. Now, let us suppose that $\widehat{K}_2$ is infinite. This means that

$$\frac{f(w^k) - f(w^k + (\widetilde{\alpha}^k/\delta) d^k)}{\widetilde{\alpha}^k/\delta} < \gamma \|d^k\|^2.$$

Then, reasoning as in the previous case, and considering (4.20), we again can conclude that $\|\nabla f(\bar{w})\| = 0$, which is again a contradiction and concludes the proof. □

4.5.2 Nonmonotone Controlled mini-batch Algorithm

In this section, we propose a nonmonotone version NMCMA of the Controlled mini-batch Algorithm.

This algorithm is motivated by relaxing the acceptance criteria so to accept even more frequently the trial point $\widetilde{w}^k$ thus possibly decreasing the need for function evaluations and increasing the computational efficiency.

In particular, we introduce a nonmonotone acceptance criterion both in step 6 of CMA and in steps 4 and 7 of Algorithm 4.4. Following classical approaches in nonmonotone scheme [76, 78],

in these conditions we substitute the value $f(w^k)$ with a reference value R^k which does not exceed the value of the objective function in the last M iterations, being M the nonmonotone memory.
Following [78], we introduce the Nonmonotone Derivative Free Linesearch procedure NMEDFL in Algorithm 4.6. We note that the use of the reference value R^k is not the only difference w.r.t. the monotone version. Indeed, steps 3 and 6 require a parabolic reduction condition on the stepsize α instead of a linear one.

Algorithm 4.6 Nonmonotone Extrapolation DF Linesearch (NMEDFL)

1: Input $(R^k, w^k, d^k, \zeta^k; \gamma, \delta)$: $w^k \in \mathbb{R}^n, d^k \in \mathbb{R}^n, R^k \in \mathbb{R}, \zeta^k > 0$; $\gamma \in (0,1), \delta \in (0,1)$
2: Set $\alpha = \zeta^k$
3: **if** $f(w^k + \alpha d^k) > R^k - \gamma(\alpha)^2 \|d^k\|^2$ **then**
4: Set $\alpha^k = 0$ and **return**
5: **end if**
6: **while** $f(w^k + (\alpha/\delta)d^k) \leq \min\{R^k - \gamma(\alpha/\delta)^2\|d^k\|^2, f(w^k + \alpha d^k)\}$ **do**
7: Set $\alpha = \alpha/\delta$
8: **end while**
9: Set $\alpha^k = \alpha$ and **return**
10: Output: α^k

The scheme of the overall nonmonotone algorithm NMCMA is provided in Algorithm 4.7. We note that step 6 of NMCMA relaxes the monotonic requirement of step 6 of CMA but strengthens the sufficient reduction condition, being $\max\{\zeta^k, \zeta^k\|d^k\|\} \geq \zeta^k$. Furthermore, we have no more the additional freedom of selecting w^{k+1} such that $f(w^{k+1}) \leq f(w^k + \alpha^k d^k)$. This is due to the need of controlling the distance $\|w^{k+1} - w^k\|$ among iterates. Thus, it is not possible to state that NMCMA is computationally less expensive than CMA .

NMCMA Convergence analysis

We first recall that the Linesearch procedure in Algorithm 4.7 is well defined, namely it terminates in a finite number of iterations.

Lemma 4.5.5. *Algorithm 4.7 is well defined, i.e. it determines in a finite number of steps a scalar α^k satisfying*

$$f(w^k + \alpha^k d^k) \leq R^k - \gamma(\alpha^k)^2\|d^k\|^2, \tag{4.21}$$

where R^k is a reference value such that

$$f(w^k) \leq R^k \leq \max_{0 \leq j \leq \min\{k,M\}} \{f(w^{k-j})\}.$$

Proof. Algorithm NMEDFL cannot cycle indefinitely within the while cycle because that would imply

$$f\left(w^k + \frac{\zeta^k}{\delta^j} d^k\right) \leq R^k - \gamma\left(\frac{\zeta^k}{\delta^j}\right)^2 \|d^k\|^2 \quad \forall j = 1, 2, \ldots$$

Algorithm 4.7 Nonmonotone Controlled mini-batch Algorithm (NMCMA)

Set $\zeta^0 > 0, \theta \in (0,1), \tau > 0, M \in \mathbb{N}$, $\gamma \in (0,1), \delta \in (0,1)$
Let $w^0 \in \mathbb{R}^n, k = 0$
for $k = 0, 1, 2 \dots$ **do**
  Set $R^k = \max_{0 \le j \le \min\{k,M\}} \{f(w^{k-j})\}$.
  Compute $(\widetilde{w}^k, d^k) =$ `Inner_Cycle`(w^k, ζ^k)
  if $f(\widetilde{w}^k) \le R^k - \gamma \max\{\zeta^k, \zeta^k \|d^k\|\}$ **then**
    $\alpha^k = \zeta^k,\ \zeta^{k+1} = \zeta^k$
  else
    if $\|d^k\| \le \tau \zeta^k$ **then**
      $\zeta^{k+1} = \theta \zeta^k$, $\alpha^k = 0$
    else
      $\alpha^k = NMEDFL(R^k, d^k, w^k, \zeta^k; \gamma, \delta)$
      if $(\alpha^k)^2 \|d^k\|^2 \le \tau \zeta^k$ **then**
        Set $\zeta^{k+1} = \theta \zeta^k$
      else
        Set $\zeta^{k+1} = \zeta^k$
      end if
    end if
  end if
  $w^{k+1} = w^k + \alpha^k d^k$
end for

so that for $j \to \infty$ we would have $(\zeta^k/\delta^j) \to \infty$ which contradicts the compactness of $\mathcal{L}_0$ and continuity of f.
In order to prove (4.21), we consider separately the two cases i) $\alpha^k = 0$ and ii) $\alpha^k > 0$. In the first case, (4.21) is trivially satisfied given the definition of R^k.
In case ii), since the if condition at step 3 of NMEDFL is not satisfied, we have that

$$f(w^k + \zeta^k d^k) \le R^k - \gamma (\zeta^k)^2 \|d^k\|^2.$$

Then, if the final α^k is equal to ζ^k, the condition is satisfied. Otherwise, if $\alpha^k > \zeta^k$, the conditions of the while loop are such that we have

$$f(w^k + \alpha^k d^k) \le R^k - \gamma (\alpha^k)^2 \|d^k\|^2$$

thus the proof is completed. □

We now recall an important lemma (Lemma 1 in [78]) that we will need further.

Lemma 4.5.6 (Lemma 1 from [78])**.** *Assume that a constant $L' > 0$ exists such that*

$$|f(u) - f(v)| \le L' \|u - v\|.$$

Let $\{w^k\}$ be a sequence such that

$$f(w^{k+1}) \le R^k - \sigma(\|w^{k+1} - w^k\|) \tag{4.22}$$

where $\sigma : \Re^+ \to \Re^+$ *is a forcing function and* R^k *is a reference value that satisfies*

$$f(w^k) \leq R^k \leq \max_{0 \leq j \leq \min\{k,M\}} \{f(w^{k-j})\}, \tag{4.23}$$

for a given integer $M \geq 0$*. Suppose that* f *is bounded below, and that it is Lipschitz continuous on* $\mathcal{L}^0$*, that is, there exists a constant* L *such that*

$$|f(x) - f(y)| \leq L\|x - y\|, \quad \textit{for all, } x, y \in \mathcal{L}^0. \tag{4.24}$$

Furthermore, for each $k \geq 0$*, let* $\ell(k)$ *be an integer such that* $k - \min\{k, M\} \leq \ell(k) \leq k$ *and that*

$$f(w^{\ell(k)}) = \max_{0 \leq j \leq \min\{k,M\}} \{f(w^{k-j})\}.$$

Then, we have:

i) $w^k \in \mathcal{L}^0$ *for all* k*;*

ii) $f(w^{k+1}) \leq f(w^{\ell(k)}) - \sigma(\|w^{k+1} - w^k\|)$*;*

iii) $\lim_{k\to\infty} \|w^k - w^{\widehat{\ell}(k)}\| = 0$*, where* $\widehat{\ell}(k) = \ell(k + M + 1)$*;*

iv) the sequence $\{f(w^k)\}$ *is convergent;*

v) $\lim_{k\to\infty} \|w^{k+1} - w^k\| = 0$*.*

Proof. For each $k \geq 0$, let $\ell(k)$ be an integer such that $k - \min(k, M) \leq \ell(k) \leq k$ and that

$$f(w^{\ell(k)}) = \max_{0 \leq j \leq \min(k,M)} [f(w^{k-j})].$$

Then, using (4.22) we get

$$f(w^{k+1}) \leq f(w^{\ell(k)}) - \sigma\left(\left\|w^{k+1} - w^k\right\|\right). \tag{4.25}$$

which proves assertion ii). As a first step we show that the sequence $\{f(w^{\ell(k)})\}$ is non increasing. Indeed, consider the index $k + 1 - \min(k + 1, M) \leq \ell(k + 1) \leq k + 1$ and using $\min(k + 1, M) \leq \min(k, M) + 1$, we get

$$\begin{aligned} &f(w^{\ell(k+1)}) = \max_{0 \leq j \leq \min(k+1,M)} [f(w^{k+1-j})] \leq \max_{0 \leq j \leq \min(k,M)+1} [f(w^{k+1-j})] = \\ &= \max\left\{f(w^{\ell(k)}), f(w^{k+1})\right\} = f(w^{\ell(k)}) \end{aligned}$$

where the last equality follows from (4.25).
As $\{f(w^{\ell(k)})\}$ is non increasing, and $w^{\ell(0)} = w^0$, we get also that $f(w^k) \leq f(w^0)$, so that $w^k \in \mathcal{L}^0$ for all k which proves assertion i). Further the non increasing $\{f(w^{\ell(k)})\}$ sequence admits a limit for $k \to \infty$.
Now we observe that if iii) holds then we easily get iv) and v). Indeed by the Lipschitz assumption

written on points w^k and $w^{\ell(k)}$ and the fact that $\{f(w^{\ell(k)})\}$ sequence admits a limit we get

$$\lim_{k\to\infty} f(w^k) = \lim_{k\to\infty} f(w^{\widehat{\ell}(k)}) = \lim_{k\to\infty} f(w^{\ell(k+M+1)}) = \lim_{k\to\infty} f(w^{\ell(k)})$$

which proves iv). Then by assumptions (4.22) and (4.23) we get also v).
Thus, it remains to prove iii). We can write

$$w^{\widehat{\ell}(k)} = \left(w^{\widehat{\ell}(k)} - w^{\widehat{\ell}(k)-1}\right) + \left(w^{\widehat{\ell}(k)-1} - w^{\widehat{\ell}(k)-2}\right) + \cdots + \left(w^{k+1} - w^k\right) + w^k$$

so that

$$\lim_{k\to\infty} \|w^{\widehat{\ell}(k)} - w^k\| = \lim_{k\to\infty} \left\| \sum_{k=1}^{\widehat{\ell}(k)-k} \left(w^{\widehat{\ell}(k)-j+1} - w^{\widehat{\ell}(k)-j}\right) \right\| \tag{4.26}$$

where $\widehat{\ell}(k) - k \leq M + 1$. We prove by induction on $1 \leq j \leq M + 1$ that

$$\lim_{k\to\infty} \left\| w^{\ell(k)-j+1} - w^{\ell(k)-j} \right\| = 0. \tag{4.27}$$

Let consider the ii) written for $k = \ell(k) - j - 1$ we have

$$f(w^{\ell(k)-j+1}) \leq f(w^{\ell(\ell(k)-j)}) - \sigma\left(\left\| w^{\ell(k)-j+1} - w^{\ell(k)-j} \right\|\right) \tag{4.28}$$

Let $j = 1$, taking the limit, using $f(w^{\ell(k)}) - f(w^{\ell(\ell(k)-1)}) \to 0$ and the definition of forcing function, we get

$$\lim_{k\to\infty} \left\| w^{\ell(k)} - w^{\ell(k)-1} \right\| = 0$$

which is iii) with $j = 1$. Using the Lipschitz continuity, we also get for $j = 1$ that

$$\lim_{k\to\infty} f(w^{\ell(k)-j}) = \lim_{k\to\infty} f(w^{\ell(k)}) \tag{4.29}$$

Now we assume that (4.27) (4.29) hold for j and we prove that they are valid for $j + 1$.
By (4.28) with $j + 1$, using (4.29) and the definition of forcing function we get

$$\lim_{k\to\infty} \left\| w^{\ell(k)-j} - w^{\ell(k)-j-1} \right\| = 0$$

Now using the Lipschitz continuity and (4.29) we get that (4.29) holds for $j + 1$ too. This terminates the induction.
Now from (4.26) we get

$$\lim_{k\to\infty} \|w^{\widehat{\ell}(k)} - w^k\| = 0$$

□

It is now possible to prove the following result on the behaviour of the sequence of points produced by algorithm NMCMA $\{w^k\}$.

Proposition 4.5.7. *Let $\{w^k\}$ be the sequence of points produced by algorithm NMCMA.*

Then,

i) $w^k \in \mathcal{L}(w^0)$ *for all* k*;*

ii) $\{R^k\}$ *and* $\{f(w^k)\}$ *are both convergent and have the same limit* R^**;*

iii) $\lim_{k\to\infty} \|w^{k+1} - w^k\| = 0$*;*

Proof. By Assumption 4.4.2 the function f is Lipschitz continuous and by Assumption 4.4.1, f is bounded below. Recalling Lemma 4.5.6, the fact that f is continuously differentiable and that $\mathcal{L}(w^0)$ is compact, we have that all the assumptions of Lemma 1 in [78] are satisfied. For the sake of completeness, the statement and proof of this Lemma are reported in Lemma 4.5.6. Hence, the proof exactly follows from [78, Lemma 1] except that we have still to prove that

$$\lim_{k\to\infty} R^k = \lim_{k\to\infty} f(w^k).$$

This is proved by noting that $\{f(w^k)\}$ converges to a limit, and by recalling the definition of R^k, i.e. $R^k = \max_{0\le j\le \min\{k,M\}}\{f(w^{k-j})\}$.

□

We now show that the inner stepsize ζ^k goes to zero as $k \to \infty$.

Proposition 4.5.8. *Let* $\{\zeta^k\}$ *be the sequence of steps produced by Algorithm NMCMA, then*

$$\lim_{k\to\infty} \zeta^k = 0.$$

Proof. The first part of the proof is similar to the one of Proposition 4.5.3. In every iteration, either $\zeta^{k+1} = \zeta^k$ or $\zeta^{k+1} = \theta\zeta^k < \zeta^k$. Therefore, the sequence $\{\zeta^k\}$ is monotonically non-increasing. Hence, it results

$$\lim_{k\to\infty} \zeta^k = \bar{\zeta} \ge 0.$$

Let us suppose, by contradiction, that $\bar{\zeta} > 0$. If this were the case, there should be an iteration index $\bar{k} \ge 0$ such that, for all $k \ge \bar{k}$, $\zeta^{k+1} = \zeta^k = \bar{\zeta}$. Namely, for all iterations $k \ge \bar{k}$, step 16 or 7 are always executed.
Let us now denote by

$$\begin{aligned} K' &= \{k :\ k \ge \bar{k},\ \text{and step 7 is executed}\}, \\ K'' &= \{k :\ k \ge \bar{k},\ \text{and step 16 is executed}\}. \end{aligned}$$

Let us first suppose that K' is infinite. Then, for $k \in K'$ and $k \ge \bar{k}$, we would have that infinitely many times

$$\gamma\bar{\zeta} \le R^k - f(w^{k+1}).$$

Then, taking the limit for $k \to \infty$ and $k \in K'$, and recalling the results of Proposition 4.5.7, we get a contradiction with $\bar{\zeta} > 0$.

On the other hand, let us suppose that K'' is infinite. Then, infinitely many times, we would have that, for $k \in K''$,

$$f(w^{k+1}) = f(w^k + \alpha^k d^k) \leq R^k - \gamma(\alpha^k)^2 \|d^k\|^2.$$

By taking the limit in the above relation and recalling the compactness of the level set $\mathcal{L}(w^0)$ and continuity of f, we would obtain

$$\lim_{k\to\infty} (\alpha^k)^2 \|d^k\|^2 = 0.$$

But then, for $k \in K''$ and sufficiently large, it would happen that

$$(\alpha^k)^2 \|d^k\|^2 \leq \tau\bar{\zeta}$$

which means that Algorithm NMCMA would have executed step 14 setting $\zeta^{k+1} = \theta\bar{\zeta}$, thus reducing ζ^{k+1} below $\bar{\zeta}$. This contradicts our initial assumption and concludes the proof. □

We are now ready to provide the convergence result for NMCMA.

Proposition 4.5.9. *Let Assumption 4.4.2 hold and let $\{w^k\}$ be the sequence of points produced by Algorithm NMCMA. Then, $\{w^k\}$ admits limit points and (at least) one of them is stationary.*

Proof. By the compactness of $\mathcal{L}(w^0)$ and considering that $w^k \in \mathcal{L}(w^0)$ for all k, we know that $\{w^k\}$ admits limit points.
Now, let us introduce the following set of iteration indices

$$K = \{k : \ \zeta^{k+1} = \theta\zeta^k\}$$

and note that, since $\lim_{k\to\infty} \zeta^k = 0$, K must be infinite. Let $\bar{w}$ be any limit point of the subsequence $\{w^k\}_K$, then, by Proposition 4.3.4, a subset $\bar{K} \subseteq K$ exists such that

$$\lim_{k\to\infty, k\in\bar{K}} w^k = \bar{w}, \tag{4.30a}$$

$$\lim_{k\to\infty, k\in\bar{K}} d^k = -\nabla f(\bar{w}), \tag{4.30b}$$

$$\lim_{k\to\infty, k\in\bar{K}} \zeta^k = 0 \tag{4.30c}$$

Then, let us now split the set $\bar{K}$ into two subsets, namely,

$$\begin{aligned} K_1 &= \{k \in \bar{K} : \ \|d^k\| \leq \tau\zeta^k\}, \\ K_2 &= \{k \in \bar{K} \setminus K_1 : (\alpha^k)^2\|d^k\|^2 \leq \tau\zeta^k\}. \end{aligned}$$

Note that, K_1 and K_2 cannot be both finite.
First, let us suppose that K_1 is infinite. In this case, (4.30) imply that

$$\lim_{k\to\infty, k\in K_1} \|d^k\| = \|\nabla f(\bar{w})\| = 0$$

thus concluding the proof in this case.
Then, let us suppose that K_2 is infinite. In this case, we proceed by contradiction and assume that $\bar{w}$ is not stationary, i.e. $\|\nabla f(\bar{w})\| > 0$. By the definition of K_2 and considering (4.30c), we obtain

$$\lim_{k\to\infty, k\in K_2} (\alpha^k)^2 \|d^k\|^2 = 0,$$

which, considering (4.30b), yields

$$\lim_{k\to\infty, k\in K_2} \alpha^k = 0. \tag{4.31}$$

Now, let us further partition K_2 into

i) $\bar{K}_2 = \{k \in K_2 : \ \alpha^k = 0\}$

ii) $\widehat{K}_2 = \{k \in K_2 : \ \alpha^k > 0\}$.

Let us first suppose that infinitely many times point (i) happens. This means that

$$\frac{f(w^k) - f(w^k + \zeta^k d^k)}{\zeta^k} < \gamma \zeta^k \|d^k\|^2.$$

Then, by the Mean-Value Theorem, a number $\hat{\alpha}^k \in (0, \zeta^k)$ exists such that

$$-\nabla f(w^k + \hat{\alpha}^k d^k)^T d^k < \gamma \zeta^k \|d^k\|^2.$$

Taking the limit in the above relation for $k \to \infty$, $k \in \bar{K}_2$, recalling (4.30), we obtain

$$\|\nabla f(\bar{w})\|^2 \leq 0$$

i.e. $\|\nabla f(\bar{w})\| = 0$.
Now, let us suppose that point (ii) happens. This means that

$$\frac{f(w^k) - f(w^k + (\alpha^k/\delta) d^k)}{\alpha^k/\delta} < \gamma (\alpha^k/\delta) \|d^k\|^2.$$

Then, reasoning as in the previous case, and recalling (4.31) we again can conclude that $\|\nabla f(\bar{w})\| = 0$, thus concluding the proof. □

4.6 Numerical Experiment

In this section, we report numerical results when comparing the implementations of CMA and NMCMA with full batch and mini-batch methods. We consider problems with different values of n and samples P to check how the algorithmic performance changes varying the size of the optimization problem and the size of the batch.
The aim of the computational experience is twofold. On the one hand, we aim to analyze the additional computational effort required by the CMA and NMCMA with respect to a standard online method and the effectiveness of the updating rules for the stepsize to avoid it to be driven

to zero too fast. On the other hand, the quality of the solution obtained in a limited time in comparison with online methods and classical batch methods.
The present section is organized as follows. In subsection 4.6.1 we provide a detailed description of the experimental testbed. In subsection 4.6.2 we report the main computational results from the main paper [131]. Finally, in subsection 4.6.3, we discuss additional computational tests carried out to compare CMA with state-of-the-art mini-batch optimizers for deep learning, including adaptive methods that, despite having weaker convergence properties, do not require at all the evaluation of $f(w)$.

4.6.1 Experimental testbed

The performances of the algorithms were tested on the minimization problem raised when minimizing the ℓ_2 regularized least square training error of a Deep Neural Network (DNN), namely

$$\min_w f(w) = \frac{1}{P}\sum_{p=1}^{P} \|\tilde{y}(w; x_p) - y_p\|^2 + \rho\|w\|^2$$

where $\{x_p, y_p\}_{p=1}^P$, with $x_p \in \mathbb{R}^d$ and $y_p \in \mathbb{R}^m$ are the training data and $\tilde{y}_p$ is the output of the DNN with weights $w \in \mathbb{R}^n$. The regularization parameter ρ is set equal to 10^{-6}.
The DNN is organized in H hidden layers. Each neuron is characterized by an activation function $g(\cdot)$, that we set to be the sigmoidal function for all the neurons, with the only exception of the output layer which has a linear activation function. Denoting with w_ℓ the weights from layer ℓ to layer $\ell + 1$, the output takes the form reported in (2.23).
We considered eight different neural network architectures classified into small and large sizes as reported in Table 4.1. For the sake of simplicity, we considered only networks with the same number of neurons per hidden layer, and we denote as $[H \times N]$ the neural network with H hidden layers and N neurons per layer. The dimension n of the trainable parameters vector w depends on the number of neurons per layer N^j, $j = 1, \ldots, H$ and on the input dimension d, and in our case is equal to $n = N(d+1) + N^2(H-1)$.

Table 4.1: Deep Network architectures

	small networks				large networks		
	$[1 \times 50]$	$[3 \times 20]$	$[5 \times 50]$	$[10 \times 50]$	$[12 \times 300]$	$[12 \times 500]$	$[30 \times 500]$
H	1	3	5	10	12	12	30
N	50	20	50	50	300	500	500

All the algorithms were tested over a set of eleven problems with a different number of samples P and features d taken from open libraries [3, 7, 36]. The datasets have been divided into two groups: *small-size* datasets, where the number of samples and/or features is small enough to be tackled by standard batch methods; *big-size* datasets, where the number of samples and/or features challenges fully batch methods and motivates the use of mini-batch algorithms.
Table 4.2 provides a description of the datasets by reporting the number of samples in the training set and the number of features, along with the number of variables of each optimization

problem corresponding to different neural architectures $[H \times N]$. Datasets are sorted according to their dimensions, with small problems reported first, and large problems reported later. The order within each group is alphabetical. All the datasets are publicly available, and the source where they can be found is reported as a reference for each dataset.

				n in K						
				small network				large network		
	Dataset	P	d	$[1 \times 50]$	$[3 \times 20]$	$[5 \times 50]$	$[10 \times 50]$	$[12 \times 300]$	$[12 \times 500]$	$[30 \times 500]$
small	Ailerons [3]	10312	41	2.1	1.64	12.1	24.6	935	2557	7567
	Bejing Pm25 [7]	31317	48	2.45	1.78	12.5	25	935	2558	7568
	Bikes Sharing [7]	13034	59	3.0	2.0	13	2.55	936	2559	7569
	California [3]	15480	9	0.5	1.0	10.5	23	933	2556	7556
	Mv [3]	30576	13	0.7	1.08	10.7	23.2	934	2556	7566
big	BlogFeedback [7]	39297	281	14.1	6.44	24.1	36.6	948	2570	7580
	Covtype [7]	435759	55	2.8	1.92	12.8	25.3	936	2558	7568
	Protein [7]	109313	75	3.8	2.32	13.8	26.3	937	2560	7570
	Sido [36]	9508	4933	247	99.5	257	269	1180	2802	7812
	Skin NonSkin [7]	183792	4	0.25	0.9	10.3	22.8	933	2555	7566
	YearPredictionMSD [7]	386508	91	4.6	2.64	14.6	27.1	937	2561	7571

Table 4.2: Dataset description (P = number of training samples; d= number of input features) and n =number of variables (in K) of the optimization problems corresponding to the network architecture $[H \times N]$

4.6.2 CMA vs full-batch methods and Incremental Gradient

Both the neural network structures and the optimization algorithms were implemented in Python version 3.9, leveraging *Pytorch 2.1.0*[1] as a scientific computing library. Numerical tests are carried out on a GPU NVIDIA Quadro P2200 (VCQP2200-SB) 5 GB.

We compared the performance of CMA and NMCMA against a full batch method and an incremental gradient method. All the algorithms run with a maximum computational time of 100 seconds.

Since the problems are highly non-convex, numerical results might be affected by the starting point used to initialize the algorithms. Thus, a *multi-start* strategy was implemented so that for each problem we compared each algorithm over five runs, each starting with a random point w^0 which is the same for the four algorithms.

As a batch method, we considered a limited-memory Quasi-Newton optimization algorithm L-BFGS [161] that can be regarded as one of the state-of-the-art batch algorithms. We used L-BFGS from the standard implementation available in *Pytorch*[2]. Default hyperparameters have been used as they resulted in the best performance.

As a mini-batch method, we implemented an Incremental Gradient (IG) method, $I^k = \{1, \dots P\}$ for all k. As updating rule for the stepsize ζ^k in Algorithm 4.2, we selected the one suggested in [71]

$$\zeta^{k+1} = \zeta^k(1 - \epsilon\zeta^k) \tag{4.32}$$

with $\zeta^0 = 0.5$ and $\epsilon = 10^{-3}$, that gave the best performance out of an hyperparameter tuning phase.

[1] https://pytorch.org/

[2] https://pytorch.org/docs/stable/generated/torch.optim.LBFGS.html

For the sake of fairness, the same IG implementation is used for the `Inner_Cycle` (Algorithm 4.1) in both CMA and NMCMA methods, as we aim to analyze to what extent controlling the stepsize in CMA and NMCMA affects the optimization performance. In CMA and NMCMA the stepsize ζ^k is automatically updated according to the controlling steps in Algorithms 4.5 and 4.7.
The tuning of the algorithmic hyperparameters of CMA and NMCMA has been performed by a grid search procedure. The resulting best values are reported in Table 4.3.

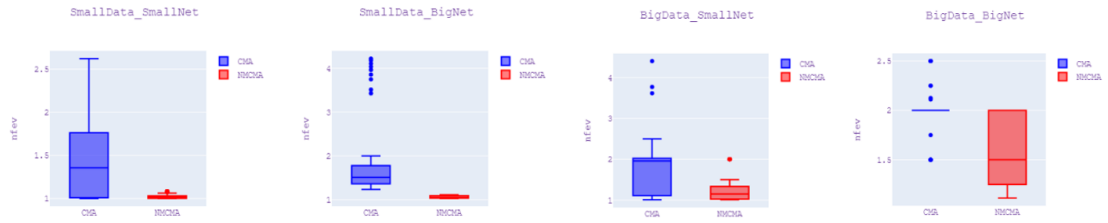
Table 4.3: Hyperparameters settings in CMA and NMCMA

	ζ^0	τ	θ	γ	δ	M
CMA	0.5	10^{-2}	0.5	10^{-6}	0.5	-
NMCMA	0.5	10^{-2}	0.5	10^{-6}	0.5	5

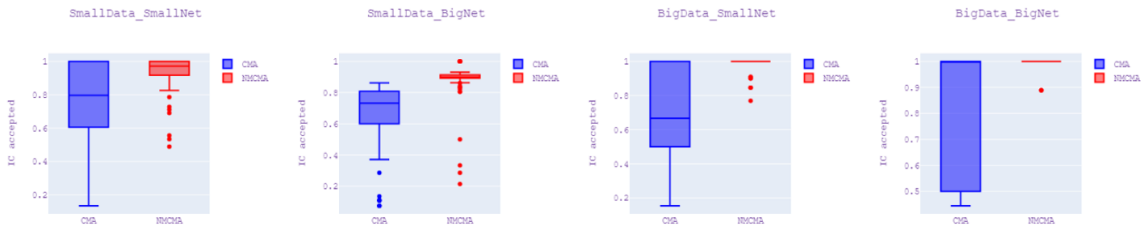
We further analyze the computational behavior of the CMA and NMCMA with the aim of

- assessing the additional effort required over the basic IG method and the effectiveness of the updating rules for the stepsize ζ^k;
- comparing performance with the basic IG and the L-BFGS methods when a restriction on the computational time is given (100 seconds).

We consider the set of 77 test problems $\mathcal{P}$, obtained by considering the 7 network architectures described in Table 4.1 and the 11 different dataset from Table 4.2. Each of these problems has been solved by each of the four algorithms starting from five different random points with a total of 385 runs for each algorithm.
We first observe that both CMA and NMCMA require at least a function evaluation in the trial point $f(\tilde{w}^k)$ at the end of each epoch. The smaller the number of function evaluations, the less the linesearch procedure is used. However, this may also result in failures in accepting the trial point $\tilde{w}^k$ produced by the `Inner_Cycle` algorithm, meaning that all the computations of the `Inner_Cycle` are lost, and the stepsize ζ^k is reduced. To analyze the occurrence of such cases, we construct the boxplots in Figure 5.1 where all the runs of CMA and NMCMA have been considered. In 4.2(a) the average number of function evaluations per iteration is reported for small (upper graph) and large (lower graphs) problems; if it is greater than 1.0, the LS is activated at least once. In 4.2(b) we report the number of times that the trial point $\tilde{w}^k$ has been accepted.
The boxplots of Figures 4.2(a) and 4.2(b) are coherent. Indeed, from Figure 4.2(a), we observe that NMCMA almost always performs a single function evaluation per iteration, which is less than the iterations performed by CMA. This corresponds to the more often acceptance of the trial point, namely the standard iteration of the IG method.
In Figure 4.3(a), we report the average value of the final stepsize for the algorithms CMA, NMCMA, and IG. In general, NMCMA tends to maintain a larger final stepsize than both CMA and IG, which is coherent with the fact that in NMCMA the trial point is accepted more often than in CMA. The three algorithms start with the same initial value of the stepsize ζ^0, however, in IG the stepsize is updated following (4.32) which allows a slower decrease rather

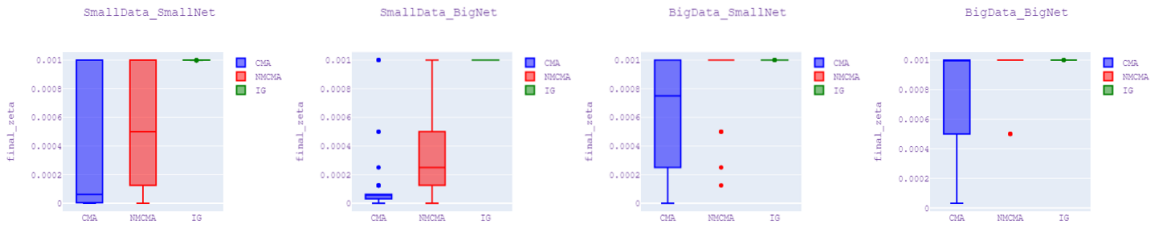


(a) Average number of function evaluations per iteration



(b) Acceptance of the trial point $\tilde{w}^k$

Figure 4.2: Boxplot on all the runs: comparison of CMA and NMCMA on the average number of function evaluations and acceptance rate of $\tilde{w}^k$



(a) Value of the final stepsize ζ: comparison of CMA, NMCMA and IG

(b) Linesearch failures: $\alpha^k = 0$

Figure 4.3: Boxplot on all the runs: stepsize values and restarts

than the simple halving rule of both CMA and NMCMA. This may explain why the average final stepsize of IG is larger than in CMA and NMCMA. Moreover, it is interesting to underline how the range of variability of the final stepsize is bigger for both CMA and NMCMA, suggesting that the two methods are able to tune the stepsize according to the specified instance under consideration.

Figure 4.3(b) reports the number of failures of the LS in CMA and NMCMA, namely the times

that $\alpha^k = 0$ and a restart is needed from the last point w^k so that all the computations of the `Inner_Cycle` are lost. We note that both in CMA and NMCMA, α^k can be set to zero not only in the LS. However, we aim to visualize how the non-monotone rule helps in accepting the stepsize. From the boxplots above, it seems that CMA and NMCMA accept often the trial point of the basic IG iteration, controlling the stepsize reduction, at the cost of extra calculus of objective functions that are not needed in the basic IG method.
We now aim to analyze the performance in terms of the quality of the returned solution (value of the objective function) when the computational time is restricted. In this last case, we compare also with the L-BFGS which in general decreases faster, being a full batch method.
To assess the numerical performance of the algorithms, we considered the performance profile curves [54], with the convergence test proposed in [152] when evaluating the performance of derivative-free solvers with a restricted computational budget. In particular, given a set of solvers $\mathcal{S}$ and a set of test problems $\mathcal{P}$, a problem $p \in \mathcal{P}$ is *solved* by solver $s \in \mathcal{S}$ if it returns a point w such that

$$f(w) \leq f_L + \tau(f(w^0) - f_L), \tag{4.33}$$

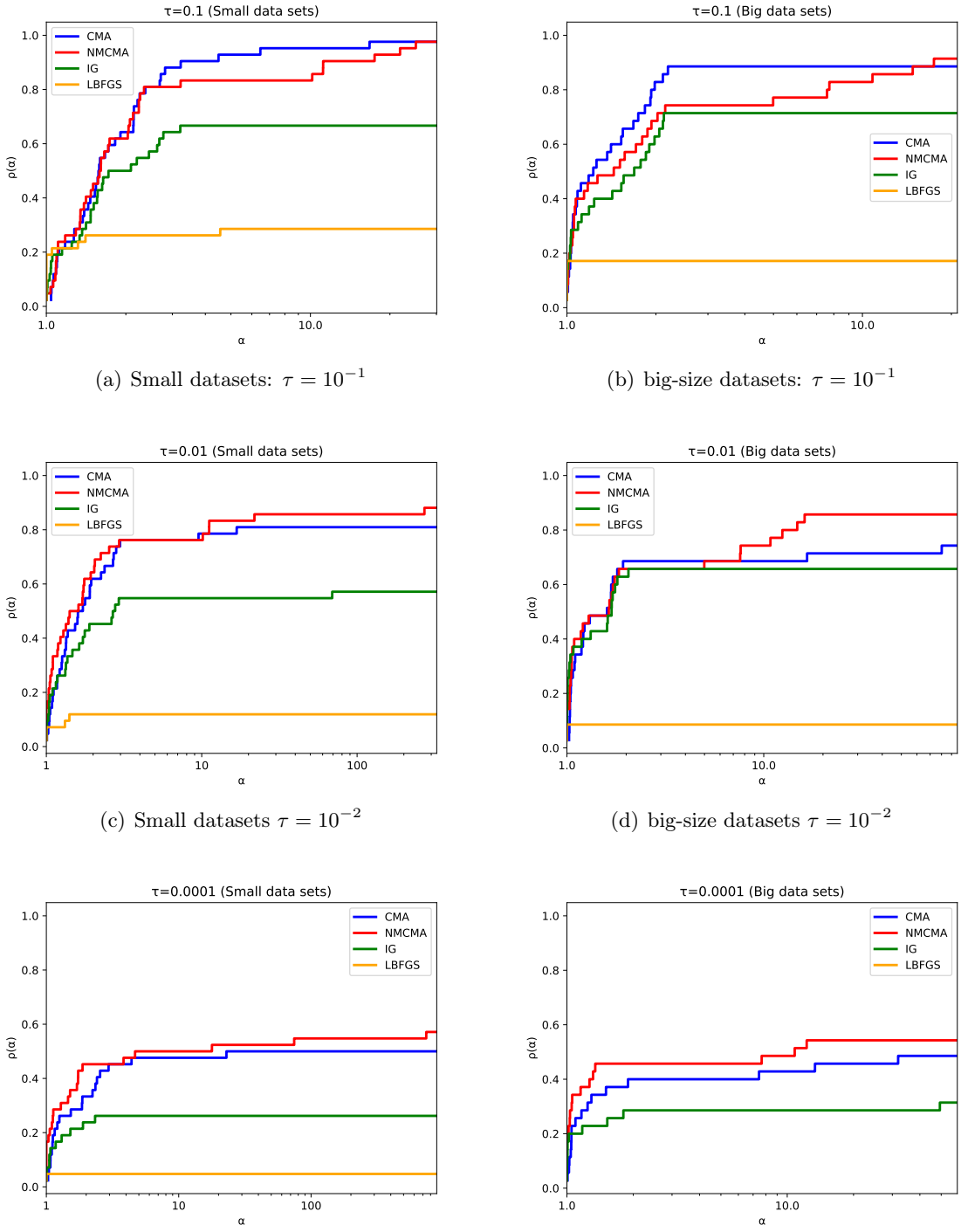
where f_L is the best value returned by all the solvers on the 5 multi-start runs, $f(w^0)$ is the starting point of the solvers (it is the same for all the solvers) and τ is a tolerance. The smaller τ, the more stringent the requirement.
For each $s \in \mathcal{S}$ and $p \in \mathcal{P}$, we define $c_{s,p}$ as the total time needed by solver s to solve problem p and the ratio

$$r_{s,p} = \frac{c_{s,p}}{\min_{s \in S}\{c_{s,p}\}}$$

which represents the performance of solver s on solving problem p compared to the best-performing solver on problem p. Thus $r_{s,p} = 1$ if the solver s solves the problem within $\min_{s \in S}\{c_{s,p}\}$ seconds, and $r_{s,p} > 1$ otherwise. Then, the *performance profile* $\rho_s(\alpha)$ of a solver s is given by

$$\rho_s(\alpha) = \frac{1}{|\mathcal{P}|} \left|\{p \in \mathcal{P} \mid r_{s,p} \leq \alpha\}\right| \qquad \alpha \geq 1, .$$

Performance profiles seek to capture how well the solver performs compared to the other solvers in $\mathcal{S}$ on the set of problems in $\mathcal{P}$. When $\alpha = 1$, $\rho_s(\alpha)$ is the fraction of problems for which solver s performs the best, when α is sufficiently large, it is the fraction of problems solved by s. Solvers with high values for $\rho_s(\alpha)$ are preferable [152].
In particular, in Figure 5.3 we report the performance profiles of the four algorithms when left running for 100 seconds. Results are reported for decreasing values of τ, $10^{-1}, 10^{-2}$ and 10^{-4}, cumulative on all the networks but separating small-size and big-size datasets, with the former on the left and the latter on the right of the figures.
When $\tau = 10^{-1}$, CMA and NMCMA have similar performance profiles and they outperformed both IG and L-BFGS methods on small-size and large-size problems.
When $\alpha = 1$ CMA and NMCMA solve a smaller portion of the problems. For increasing values of α they both reache 100 %, CMA having a slight advantage over its non-monotone counterpart. Conversely, L-BFGS is far away and it solves around one third of the small-size dataset problems and less than 20 % of the large-size ones, which is an expected result. Indeed, despite its faster



(a) Small datasets: $\tau = 10^{-1}$

(b) big-size datasets: $\tau = 10^{-1}$

(c) Small datasets $\tau = 10^{-2}$

(d) big-size datasets $\tau = 10^{-2}$

(e) Small datasets $\tau = 10^{-4}$

(f) big-size datasets $\tau = 10^{-4}$

Figure 4.4: Performance profiles on all the networks for different values of τ. Results for small-size datasets are reported to the left, and for big-size datasets to the right.

theoretical convergence, L-BFGS iteration often requires a large number of function evaluations to become inefficient against mini-batch algorithms. For decreasing values of $\tau = 10^{-2}$ and $\tau = 10^{-4}$, this behaviour is even more evident. When high-precision on large-size dataset is required (see 4.4(f)), L-BFGS never happens to be the best solver. It is interesting to notice that CMA and NMCMA, despite requiring at least one evaluation of the entire objective function

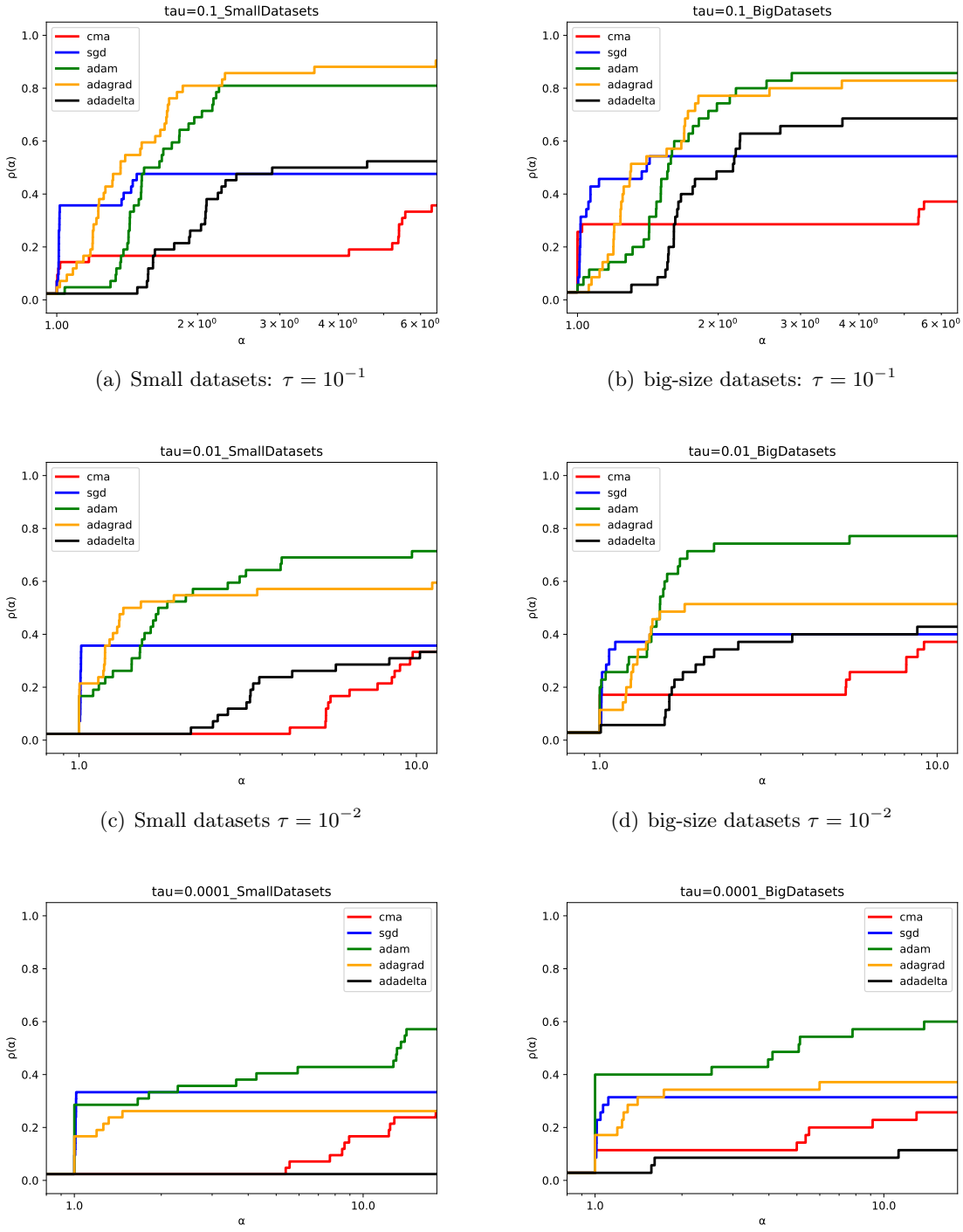
(i.e., on the whole dataset) per iteration, always outperform IG, which never requires to compute the full $f(\omega^k)$. This suggests that checking the quality of the trial direction can in practice speed up the convergence and leading faster to a stationary point. When dealing with larger problems, the advantage of CMA and NMCMA over IG is slightly reduced by the computational burden of evaluating $f(\omega^k)$, but even in that case they both seem to scale up keeping their advantage. Eventually, recalling the bias that can be introduced by the use of the performance profiles to compare more than two solvers [72], we also report in the appendix, in figure A.1, the comparison between CMA and NMCMA, which confirms a slight outperforming of the latter.

4.6.3 CMA vs adaptive methods, towards further research

In addition to the results presented in the previous section, we provide other findings from our attempts to test CMA against state-of-the-art deep learning methods. As extensively discussed in Chapter 3, these methods have weaker convergence properties but are regarded as the most efficient, which is why we used them for comparison with CMA. The results presented here are not included in the main paper [131] because their purpose is to evaluate the trade-off between computing the objective function at least once per iteration with strong convergence guarantees. These tests served also a starting point for the prosecution of this research, with CMA Light algorithm, presented in the following chapter.

For these tests, we have used the same experimental testbed presented in subsection 4.6.1. We tested CMA with the hyper-parameters setting of Table 4.3 against Adam, Adadelta, Adagrad and SGD with weight decay from the `Pytorch` library. As done for the experiments reported in the last subsection, we have fixed a time limit of 100 seconds and carried out 5 multi-start runs per each architecture-dataset pair, resulting in 385 runs for every optimizer. The performance profiles obtained are reported in Figure 4.5.

We observe that the adaptive methods are generally much more efficient than CMA both for small and large datasets. Despite being less stable, they do not need any full evaluation of $f(w)$. This consideration, along with the growing interest in the recently developed Objective Function-Free Optimization (OFFO) framework [73, 74], motivated our continued research on the CMA framework. As we will discuss in the following chapter, identifying a limitation of CMA (specifically, the requirement for at least one evaluation of $f(w)$ per iteration) became the basis for developing a new variant of CMA that eliminates this computational bottleneck by not requiring $f(w)$ at every iteration.



(a) Small datasets: $\tau = 10^{-1}$

(b) big-size datasets: $\tau = 10^{-1}$

(c) Small datasets $\tau = 10^{-2}$

(d) big-size datasets $\tau = 10^{-2}$

(e) Small datasets $\tau = 10^{-4}$

(f) big-size datasets $\tau = 10^{-4}$

Figure 4.5: Performance profiles against Adam, Adagrad, Adadelta, and SGD with weight decay on all the networks for different values of τ. Results for small-size datasets are reported to the left, and for big-size datasets to the right.

4.7 CMA-FD: estimating the complexity of CMA

Differently from most of the currently available research (see Chapter 3), in CMA we develop a unified monotone and non monotone framework, and prove the global convergence to a stationary

point in a fully deterministic way. Our convergence theory holds under the solely assumptions of coercivity and smoothness of the objective function $f(w)$, without any further hypothesis on the search direction. However, this is attained by admitting the possibility that an entire epoch fails in returning a successful point, leading to an undefined amount of wasted computational effort. While creating a novel convergence theory removing any possible assumption on the search direction is out of the scope of this thesis, in this section we present CMA with Fixed Decrease condition (CMA-FD), an adjusted version of CMA to prove $\mathcal{O}(\varepsilon^{-2})$ worst-case complexity. The analysis is straightforward and based on the current available literature.
Although the complexity of CMA is still an open question, our theoretical analysis, along with early computational results, indicates that the efficiency of CMA-FD is comparable to that of CMA against IG and L-BFGS.

4.7.1 Algorithmic scheme and assumptions

The rough scheme of CMA-FD is reported in Algorithm 4.8, while in Algorithm 4.9 we report the extrapolation derivative-free linesearch tailored for CMA-FD.

Algorithm 4.8 CMA-FD

1: Let $\gamma > 0$, $\delta \in (0,1)$, $\theta \in (0,1)$, $\zeta^0 = \alpha^0 > 0$,$\bar{\zeta} > 0$, $w^0 \in \mathbb{R}^n$
2: Set $k = 0$
3: **for** $k = 0, 1, 2 \ldots$ **do**
4: Compute a search direction d_k
5: Compute $\widetilde{w}_k = w_k + \zeta_k d_k$
6: **if** $f(\widetilde{w}_k) \leq f(w_k) - \bar{\zeta}$ **then**
7: Set $w^{k+1} = \widetilde{w}_k$
8: Set $\zeta^{k+1} = \zeta_k$
9: Set $\alpha_k = \zeta_k$
10: **else**
11: Set $\widetilde{\alpha}_k = \texttt{EDFL}(w_k, d_k, \zeta_k, \gamma, \delta)$
12: **if** $\widetilde{\alpha}_k > 0$ **then**
13: Set $\alpha_k = \widetilde{\alpha}_k$
14: Set $\zeta_{k+1} = \zeta_k$
15: **else**
16: Set $\alpha_k = 0$ and $\zeta_{k+1} = \theta \zeta_k$
17: **end if**
18: **end if**
19: Set $w^{k+1} = w_k + \alpha_k d_k$
20: **end for**

In Algorithm 4.8, the Step 4 can be performed with any suitable mini-batch method. However, in order to determine a complexity bound for CMA, we need to make the following assumptions.

Assumption 4.7.1. *The function f is coercive, i.e. all the level sets*

$$\mathcal{L}(w_0) = \{w \in \Re^n : \ f(w) \leq f(w_0)\}$$

are compact for any $w_0 \in \Re^n$.

Algorithm 4.9 EDFL (for CMA-FD)

Input $(w_k, d_k, \zeta_k; \gamma, \delta)$: $w_k \in \mathbb{R}^n, d_k \in \mathbb{R}^n, \zeta_k > 0$, $\gamma \in (0,1), \delta \in (0,1)$
Set $\alpha = \zeta_k$
if $f(w_k + \alpha d_k) > f(w_k) - \gamma\alpha\|d_k\|^2$ **then**
    Set $\alpha_k = 0$ and **return**
end if
Set max number of iteration N_E
while $f(w_k + (\alpha/\delta)d_k) \leq \min\{f(w_k) - \gamma(\alpha/\delta)\|d_k\|^2, f(w_k + \alpha d_k)\}$ **and** $i \leq N_E$ **do**
    Set $\alpha = \alpha/\delta$
end while
Set $\alpha_k = \alpha$ and **return**
Output: α_k

Assumption 4.7.2. *The function f is Lipschitz-smooth and there exists $L > 0$ such that*

$$\|\nabla f(u) - \nabla f(v)\| \leq L\|u - v\| \qquad \forall\ u, w \in \mathbb{R}^n$$

Assumption 4.7.3. *The gradients ∇f_p are Lipschitz continuous for each $p = 1, \ldots, P$, namely, for all $i = 1, \ldots, P$ there exists $L_p > 0$ such that*

$$\|\nabla f_p(u) - \nabla f_p(v)\| \leq L_p\|u - v\| \qquad \forall\ u, w \in \mathbb{R}^n$$

Assumption 4.7.4. *The function f is bounded below, i.e., there exists a value $f^{low} \in \mathbb{R}$ such that:*

$$f(w) \geq f^{low}\ \forall\ w \in \mathbb{R}^n$$

Notice that they are the same assumptions made in 4.4, expect for the boundedness of the objective function (Assumption 4.7.4).
However, to prove the theoretical results, we need to introduce other assumptions on the search direction, which allows to prove the following lemma.

Lemma 4.7.5. *Suppose that Assumptions 4.7.1, 4.7.2, 4.7.3, 4.7.4 hold. Suppose also that there exist two positive constants $c_1, c_2 > 0$ such that, for every iteration $k = 0, 1, \ldots$, d_k is a search direction with the following properties:*

a) $\|d_k\|^2 \geq c_1\|\nabla f(w_k)\|^2$;

b) $-\nabla f(w_k)^T d_k \geq c_2\|d_k\|^2$.

Then, for $\gamma \in (0, c_2)$, any scalar $\alpha \leq \frac{2(c_2-\gamma)}{L}$ satisfies the condition at step 6 of Algorithm 4.9.

Proof. First of all, recalling the integral mean-value theorem, we have that, given a point w, a direction z and a step-size ξ, we can build a function depending on a step-size $\xi \in \mathbb{R}$, $g(\xi) = f(x + z\xi)$, for which it holds:

$$f(x+z) - f(x) = g(1) - g(0) = \int_0^1 \frac{dg}{d\xi}\xi d\xi = \int_0^1 z^T \nabla f(x + \xi z) d\xi \leq \int_0^1 z^T \nabla f(x) d\xi$$

$$+|\int_0^1 z^T[\nabla f(x+\xi z)-\nabla f(x)]d\xi| \leq z^T\nabla f(x)+\|z\|^2\int_0^1 L\xi\|z\|^2 d\xi = z^T\nabla f(x)+\frac{L}{2}\|z\|^2$$

Now, applying this result on our case, we can set $x = w_k$ and $z = d_k\alpha_k$. Thus, it is always true in CMA, that:

$$f(w^{k+1}) - f(w_k) \leq \alpha_k d_k^T \nabla f(w_k) + \frac{L}{2}(\alpha_k)^2\|d_k\|^2 \tag{4.34}$$

In particular, after EDFL, we will get either $\alpha_k = \frac{\zeta_k}{\delta^{ED_k}}$, being ED_k the number of EDFL inner iterations at CMA iteration k, or $\alpha_k = 0$, in case condition at Step 3 of EDFL holds.
Now, we consider that the condition at Step 6 is satisfied if:

$$f(w_k+\alpha d_k) \leq f(w_k) - \gamma\frac{\alpha}{\delta}\|d_k\|^2 \leq f(w_k) - \gamma\alpha\|d_k\|^2 \tag{4.35}$$

Recalling (4.34), this means that Step 6 is satisfied if:

$$\alpha(-d_k^T\nabla f(w_k)) - \frac{L}{2}\alpha^2\|d_k\|^2 \geq \gamma\alpha\|d_k\|^2 \tag{4.36}$$

Now, using condition b), we can write:

$$\alpha(-d_k^T\nabla f(w_k)) - \frac{L}{2}\alpha^2\|d_k\|^2 \geq \alpha c_2\|d_k\|^2 - \frac{L}{2}\alpha^2\|d_k\|^2 \geq \gamma\alpha\|d_k\|^2$$

Dividing by $\alpha\|d_k\|^2$ we get:

$$c_2 - \frac{L}{2}\alpha \geq \gamma$$

which proves the thesis. □

4.7.2 Worst-case complexity in finding an ε-stationary point

To show the convergence rate of Algorithm 4.8 in finding an ε-stationary point for $f(w)$, we firstly need to define the *successful* iteration of CMA-FD with fixed decrease and to prove that there exists a minimal improvement in the value of the objective function for every such iteration. Then, we need to provide a bound for unsuccessful iterations.
Recalling that $f(w)$ is bounded below (Assumption 4.7.4), we only focus on the convergence rate in relation with 4.9, as condition at Step 6 of Algorithm 4.8 can hold only for a finite number of times. We define the following sets of iterations:

- Successful iterations at step k: $\mathcal{S}^k = \{j = 0, 1, \ldots, k : \alpha^j > 0\}$;
- Unsuccessful iterations at step k $\mathcal{U}^k = \{0, 1, \ldots, k\} \setminus \mathcal{S}^k$.

Now, with the following proposition, we can bound the number of successful iterations.

Proposition 4.7.6. *If at iteration k there exists a scalar α_{min} such that $\alpha_j \geq \alpha_{min}$ for all $j = 0, \ldots, k$, then:*

$$k \leq \frac{1}{\log\left(\frac{1}{\theta}\right)}\left[|\mathcal{S}_k|\log\left(\frac{\delta^{N_E}}{\theta}\right) + \log\left(\frac{\alpha_0}{\alpha_{min}}\right)\right] \tag{4.37}$$

Proof. We have that:

$$\alpha_{min} \leq \alpha_k \leq \alpha_0 \delta^{|\mathcal{S}_k| N_E} \theta^{|\mathcal{U}_k|}$$

where the right side of the inequality comes from the case where, at every successful iteration, EDFL is executed, always for the maximum number of iteration N_E.
Taking the logarithm we get:

$$|\mathcal{S}_k| N_E \log(\delta) + |\mathcal{U}_k| \log(\theta) + \log(\alpha_0) \geq \log(\alpha_{min})$$

$$|\mathcal{S}_k| N_E \log(\delta) + |\mathcal{U}_k| \log(\theta) \geq \log\left(\frac{\alpha_0}{\alpha_{min}}\right)$$

Recalling that $|\mathcal{U}_k| = k - |\mathcal{S}_k|$ we get:

$$|\mathcal{S}_k| N_E \log(\delta) - |\mathcal{S}_k| \log(\theta) + k \log(\theta) \geq \log\left(\frac{\alpha_0}{\alpha_{min}}\right)$$

that is:

$$k \log(\theta) \geq \log\left(\frac{\alpha_0}{\alpha_{min}}\right) + |\mathcal{S}_k|(\log(\theta) - N_E \log(\delta)) = \log\left(\frac{\alpha_0}{\alpha_{min}}\right) + |\mathcal{S}_k| \log\left(\frac{\theta}{\delta^{N_E}}\right)$$

Multiplying by -1, we get:

$$k \log\left(\frac{1}{\theta}\right) \leq \log\left(\frac{\alpha_{min}}{\alpha_0}\right) + |\mathcal{S}_k| \log\left(\frac{\delta^{N_E}}{\theta}\right)$$

which concludes the proof. □

We now prove that there is a fixed minimal improvement in the value of $f(w)$ at any successful iteration.

Proposition 4.7.7. *Suppose Assumptions 4.7.1, 4.7.2, 4.7.3, 4.7.4, and hypotheses a) and b) of Lemma 4.7.5 hold. Suppose also that* $\alpha_0 > \frac{2(c_2-\gamma)}{L} > 0$. *Then, at any successful iteration* k *of Algorithm 4.8, it holds that*

$$\min_{j \leq k} \alpha_j > \alpha_{min} = \theta \frac{2(c_2 - \gamma)}{L}$$

Proof. The proof follows by considering that, at every iteration k, there are only three possible outcomes of CMA-FD:

i) The fixed decrease condition at Step 6 holds, which, as said before, can hold only for a finite number of times;

ii) EDFL succeeds in finding a scalar, which satisfies condition (4.36);

iii) EDFL fails, i.e., condition at Step 3 of Algorithm 4.9 is satisfied.

Now, let's admit, by contradiction, that there exists a successful iteration (type i) or ii)) $\bar{k}$ for which $\alpha_{\bar{k}} \leq \theta \frac{2(c_2-\gamma)}{L}$. This means that there was an unsuccessful iteration (type iii)) $\tilde{k} < \bar{k}$, where $\alpha_{\tilde{k}} = \frac{\alpha_{\bar{k}}}{\theta} \leq \frac{2(c_2-\gamma)}{L}$. But this is contradiction with Lemma 1, because if $\alpha_{\tilde{k}}$ was less than the quantity $\frac{2(c_2-\gamma)}{L}$, then the iteration should have been successful, which proves the thesis. □

We can now finally prove the complexity of Algorithm 4.8.

Theorem 4.7.8. *Suppose Assumptions Assumptions 4.7.1, 4.7.2, 4.7.3, and hypotheses a) and b) of Lemma 4.7.4 hold. Suppose that* $\alpha_0 > \frac{2(c_2-\gamma)}{L} > 0$. *Then, the number of iteration of Algorithm 4.8 to obtain a point* w_k *where* $\|\nabla f(w_k)\| \leq \varepsilon$ *is bounded by the following constant:*

$$\#_{it} = \frac{1}{\log\left(\frac{1}{\theta}\right)} \left[\frac{f(w^0) - f_{low}}{\min\{\bar{\zeta}, \gamma \alpha_{min} c_1 \varepsilon^2\}} \log\left(\frac{\delta^{N_E}}{\theta}\right) + \log\left(\frac{\alpha_0}{\alpha_{min}}\right) \right] \tag{4.38}$$

where $\alpha_{min} = \theta \frac{2(c_2-\gamma)}{L}$.

Proof. Let's first identify the minimal improvement in the value of $f(w)$ for each successful iteration k. In case condition at Step 6 of Algorithm 4.8 holds, then this improvement is γ. Conversely, if the EDFL is successful, using the results of Lemma 4.7.5 and Proposition 4.7.7, we find that this improvement is the quantity:

$$\gamma \alpha_k \|d^k\|^2 \geq \theta \gamma \frac{2(c_2-\gamma)}{L} c_1 \|\nabla f(w_k)\|^2$$

.

Let's now suppose that at iteration k, we have that $\min_{j \leq k} \|\nabla f(w_j)\| \geq \varepsilon$. We can now consider that:

$$f(w^0) - f_{low} \geq f(w^0) - f(w_k) = \sum_{j=0}^{k-1} [f(w^j) - f(w^{j+1})] = \sum_{j \in \mathcal{S}^k} f(w^0) - f(w_k) \geq$$

$$\sum_{j \in \mathcal{S}^k} \min\{\bar{\zeta}, \gamma\theta \frac{2(c_2-\gamma)}{L} \|\nabla f(w_k)\|^2\} \geq |\mathcal{S}^k| \min\{\bar{\zeta}, \gamma\theta \frac{2(c_2-\gamma)}{L} c_1 \varepsilon^2\}$$

.

Now, we can name $\alpha_{min} = \theta \frac{2(c_2-\gamma)}{L}$ and we find the bound on the number of successful iterations:

$$|\mathcal{S}^k| \leq \frac{f(w^0) - f_{low}}{\min\{\bar{\zeta}, \gamma \alpha_{min} c_1 \varepsilon^2\}} \tag{4.39}$$

We can now replace $|\mathcal{S}^k|$ in (4.37) and this concludes the proof of (4.38). □

4.7.3 Early results and conclusions

To gain a general understanding of how CMA-FD performs compared to its main competitors, and to compare it with CMA, we repeated the computational tests from section 4.6 on the same problems described in Table 4.2. The performance profiles are reported in the appendix of this thesis, in figure A.2 (CMA-FD against CMA competitors) and in figure A.3 (CMA-FD against CMA). It can be observed that the performance of CMA-FD and CMA are extremely

similar for low precision levels, whereas, for higher precision, CMA-FD tends to perform slightly better, which is also confirmed looking at figure A.3. We argue that this can happen because in CMA-FD the condition for to have a successful iteration is stricter than in CMA, where the sufficient improvement tends to zero with the step-size ζ^k.

Chapter 5

CMA Light: a mini-batch algorithm for non-convex large-scale finite sum optimization

In this chapter we present the enhanced almost objective-function free version of CMA, CMA Light, tailored for applications in training large-scale neural architectures.
The backbone of this chapter comes from the pre-print [43] by C. Coppola, G. Liuzzi and L. Palagi, which is currently under the first cycle of peer review. However, the introduction and some theoretical remarks have been shortened with respect to the original paper in order not to repeat concepts and explanations the reader, coming from the previous chapter, is already familiar with. The notation and the general set-up of this chapter follows the same structure of the previous one, which is often recalled to precise or update results and/or technical details, such as the mechanic of the algorithms. Furthermore, the early computational study presented in section 5.5 is not part of the original paper and is instead the result of Ilaria Ciocci's Master Thesis work, which has been conducted under the supervision of Prof. Laura Palagi, Lorenzo Papa, and Corrado Coppola. This work resulted in the publication of a technical report [42], which contains the extended version of section 5.5.
The remainder of the chapter is organized as follows. After a brief introduction in section 5.1, we begin by laying out the assumptions and background of the CMA Light algorithm in section 5.2. This is followed, in section 5.3, by a detailed description of the algorithm and the proof of the global convergence of CMA Light to a stationary point. Then, in section 5.4 we present our computational experiments. In section 5.5 we present an early computational study on a block-decomposed version of CMA Light, BD-CMAL. Finally, in section 5.6, we discuss our conclusions.

5.1 Introduction

We consider the same non-convex unconstrained minimization problem described in (4.1) arising in deep learning applications, using the same notation:

$$\min_{w \in \mathbb{R}^n} \qquad f(w) = \sum_{p=1}^{P} f_p(w) \tag{5.1}$$

where $f_p : \mathbb{R}^n \to \mathbb{R}$, $p = 1, \ldots, P$ are continuously differentiable and possibly non-convex functions, as already stated in (2.6).

In recent years, the need for efficient and scalable algorithms for non-convex large-scale optimization problems has become increasingly important, particularly in the context of deep learning applications. The focus of this chapter is the development of CMA Light. Our approach builds on the foundational work of the CMA framework, which was introduced in the previous chapter. While CMA/NMCMA offer significant advantages in terms of convergence guarantees, their reliance on at least one evaluation of the objective function across the entire dataset per each epoch limits the applicability in large-scale settings. Furthermore, as shown in section 4.6.3, even on problems with only few millions of variables (see table 4.2), CMA is already outperformed by adaptive methods, in spite of its theoretical superiority.

The purpose of CMA Light is to address the computational bottleneck of the CMA framework by reducing the frequency of full-dataset evaluations, making it more efficient while maintaining equally strong theoretical guarantees. As we will see, this is achieved by replacing the real objective function value with an approximation built on mini-batch gradients.

As we have many times remarked in the previous chapters, mini-batch methods are widely employed in deep learning due to their ability to handle large datasets and deep architectures but their convergence is proved only under strong assumptions (see chapter 3). To the best of our knowledge, only few attempts have been made to prove the convergence of algorithms that avoid the explicit evaluation of the objective function, while not requiring *strong* assumptions, such as convexity or conditions on the search direction. In particular, the OFFO (Objective Function-Free Optimization) framework, introduced in 2023 by Gratton et al. [73], marked for the research community a significant step forward to the idea that evaluating the objective function is not strictly necessary in large-scale unconstrained optimization. The authors solve problem (5.1) relying only on an approximation of $f(w)$ and, assuming that $f \in \mathcal{C}_L^p$ and that it is possible to compute up to the $(p-1)$-th derivatives tensor of the objective function, prove that OFFO converges to an ε-stationary point within $\mathcal{O}(\varepsilon^{-\frac{p+1}{p}})$ iterations. While this bound does not have a direct application to deep learning, where evaluating $\nabla f(w)$ or, in general, $\nabla^p f(w)$, is computationally impracticable, the authors' finding prove that, having $\nabla f(w)$ (i.e., $p = 2$), means converging to an ε-stationary point within $\mathcal{O}(\varepsilon^{-\frac{3}{2}})$ iterations without *any single evaluation* of $f(w)$, which is $\mathcal{O}(\frac{1}{\sqrt{\varepsilon}})$ better than all the standard results we have reviewed in chapter 3. Intuitively, this suggests that having the gradient, or at least a good estimate of it, is much more important than perfectly knowing the value of the objective function. Eventually, in 2024, Gratton et al. [74] have analyzed a particular case of OFFO framework, the ASTR1 algorithm, which generalizes Adagrad as an example of objective function-free first-order method with approximated gradients.

5.2 Initial assumptions and background

Following the CMA framework introduced in the previous chapter, we aim at improving its efficiency making only seldomly necessary to evaluate the objective function on the whole dataset. While this is a common practice in Deep Learning community, especially when dealing with large-scale datasets and deep architectures, to the best of our knowledge no attempts to prove global converge have been made yet. We introduce the following assumptions on the objective function and on its components f_p.

Assumption 5.2.1. *The function $f(w)$ is coercive, i.e. all the level sets*

$$\mathcal{L}(w_0) = \{w \in \Re^n : \ f(w) \leq f(w_0)\}$$

are compact for any $w_0 \in \Re^n$.

Assumption 5.2.2. *The gradients $\nabla f_p(w)$ are Lipschitz continuous for each $p = 1, \ldots, P$, namely there exists $L > 0$ such that*

$$\|\nabla f_p(u) - \nabla f_p(v)\| \leq L\|u - v\| \qquad \forall\ u, w \in \mathbb{R}^n$$

Assumption 5.2.3. *The components f_p are all non-negative and coercive, for each $p = 1, \ldots, P$, namely,*

1. *$f_p(w) \geq 0$ for any $w \in \mathbb{R}^n$*
2. *$\mathcal{L}(w_0) = \{w \in \Re^n : \ f_p(w) \leq f_p(w_0)\}$ is a compact set for each $p = 1, \ldots, P$ and for any $w_0 \in \mathbb{R}^n$*

Although the assumption 5.2.3 was not required in CMA, we note that it holds for at least the most common loss functions used in training deep neural networks, such as mean square error and cross-entropy loss.
Similarly to CMA, our algorithm is based on a mini-batch gradient method iteration embedded in an outer iteration (epoch).
Algorithm 4.1 encompasses not only the Incremental Gradient method [14, 17, 20] but also most of the state-of-the-art optimizers used to train deep neural networks (see 2.2). However, exploiting its deterministic convergence properties, we focus on the Incremental Gradient method. Since IG requires strong properties on the gradient of the objective functions [17], CMA framework has been proposed as an ease-controlled modification of IG which converges under milder assumptions. Nonetheless, the watchdog rule introduced in the CMA framework to force the sequence generated by the algorithm to remain within a compact level set requires at least one evaluation of the objective function per each epoch to verify that $f(w^k) \leq f(w^0)$. While computational effort is still accepTable when training simple neural architectures, evaluating the objective function on the whole dataset is computationally unaffordable for large-scale datasets and state-of-the-art deep networks.
The underlying idea of CMA Light is to replace the real objective function value with an estimation, without introducing any further assumption. For this purpose we define a new `Inner_Cycle`

in Algorithm 5.1, which is by definition an epoch of IG method returning also an approximated value of the objective function $\tilde{f}^k$.

Algorithm 5.1 `Inner_Cycle`

1: **Input**: w^k, ζ^k
2: Set $\widetilde{w}_0^k = w^k$
3: **for** $i = 1, ..., P$ **do**
4: $\quad \tilde{f}_i^k = f_i(\widetilde{w}_{i-1}^k)$
5: $\quad \tilde{d}_i^k = -\nabla f_i^k(\widetilde{w}_{i-1}^k)$
6: $\quad \widetilde{w}_i^k = \widetilde{w}_{i-1}^k + \zeta^k \tilde{d}_i^k$
7: **end for**
8: **Output** $\widetilde{w}^k = \widetilde{w}_P^k$, $d^k = \sum_{i=1}^{P} \tilde{d}_i^k$, $\tilde{f}^k = \sum_{i=1}^{P} \tilde{f}_i^k$

More formally, in CMA Light we avoid evaluating $f(w^k)$ by checking the watchdog rule condition on the following approximated function

$$\tilde{f}^k = \sum_{i=1}^{P} f_i(\tilde{w}_{i-1}^k) = f_1(\tilde{w}_0^k) + f_2(\tilde{w}_1^k) + \cdots + f_P(\tilde{w}_{P-1}^k) \tag{5.2}$$

where the points $\tilde{w}_i^k$ are defined according to the iterations of the `Inner_Cycle`, i.e., we replace the real value of $f(w^k)$ with the sum of the f_p evaluated in the points produced by an IG epoch. According to Lemma 4.3.2 andd Proposition 4.3.3, we can prove the following result, i.e., the approximated objective function converges to the real $f(w)$.

Proposition 5.2.4. *Let $f_i(w) \in \mathcal{C}_L^1$ for all $i = 1, \ldots, P$. Assume that $\{w^k\}$ is bounded and that $\lim_{k\to\infty} \zeta^k = 0$. Let $\{\widetilde{w}_i^k\}$ and $\{d^k\}$ be the sequences of points and directions produced by Algorithm 5.1.*
Then, for any limit point $\bar{w}$ of $\{w^k\}$ a subset of indices K exists such that

$$\lim_{k\to\infty, k\in K} w^k = \bar{w}, \tag{5.3}$$

$$\lim_{k\to\infty, k\in K} \zeta^k = 0, \tag{5.4}$$

$$\lim_{k\to\infty, k\in K} \widetilde{w}_i^k = \bar{w}, \quad \textit{for all } i = 1, \ldots, P \tag{5.5}$$

$$\lim_{k\to\infty, k\in K} d_k = -\nabla f(\bar{w}). \tag{5.6}$$

$$\lim_{k\to\infty, k\in K} \tilde{f}^k = f(\bar{w}). \tag{5.7}$$

Proof. Recalling that 5.3,5.4,5.5,5.6 have already been proved in Proposition 4.3.4, we have, from Algorithm 5.1:

$$\tilde{f}^k = \sum_{i=1}^{P} f_i(\tilde{w}_{i-1}^k) = f_1(\tilde{w}_0^k) + f_2(\tilde{w}_1^k) + \cdots + f_P(\tilde{w}_{P-1}^k)$$

Recalling that $f(w) = \sum_{i=1}^{P} f_i(w)$ and that $f_i(w) \in \mathcal{C}_L^1$ for all $i = 1, \ldots, P$, using 5.5, we have:

$$\lim_{k \to \infty, k \in K} \tilde{f}^k = \sum_{i=1}^{P} f_i(\bar{w}) = f(\bar{w})$$

□

5.3 The convergence of CMA Light

In this section we define the algorithm scheme of CMA Light and prove convergence under the assumptions above. The CMA framework has been developed aiming at improving computational efficiency with respect to batch-methods, while preserving deterministic convergence properties. In the first version of CMA (Algorithm 4.5), the convergence is ensured by a watchdog approach, imposing a condition of sufficient decrease of the objective function (step 5 of Algorithm 4.5). When this condition holds, which is the most frequent case, the point returned by the Inner Cycle is accepted and no other operations are required. Conversely, when the condition is violated, either the step-size is adjusted, or the derivative-free linesearch is executed.
In CMA Light we want to accept as much as possible the points generated by Algorithm 5.1 so that in practice we almost never need to evaluate $f(w^k)$. To further reduce the number of function evaluations, if the sufficient decrease condition is not satisfied, we perform a modified version of Algorithm 4.4, which we name EDFL Light and report in Algorithm 5.2.

Algorithm 5.2 Extrapolation Derivative-Free Linesearch (`EDFL_Light`)

1: Input $(\tilde{f}^k, w^k, d^k, \zeta^k; \gamma, \delta)$: $w^k \in \mathbb{R}^n, d^k \in \mathbb{R}^n, \zeta^k > 0$, $\gamma \in (0,1), \delta \in (0,1)$
2: Set $j = 0, \alpha = \zeta^k, f_j = \tilde{f}^k$
3: **if** $\tilde{f}^k > f(w^k) - \gamma\alpha\|d^k\|^2$ **then**
4: Set $\alpha^k = 0$ and **return**
5: **end if**
6: **while** $f(w^k + (\alpha/\delta)d^k) \leq \min\{f(w^k) - \gamma\alpha\|d^k\|^2, f_j\}$ **do**
7: Set $f_{j+1} = f(w^k + (\alpha/\delta)d^k)$
8: Set $j = j + 1$
9: Set $\alpha = \alpha/\delta$
10: **end while**
11: Set $\alpha^k = \alpha$ and **return**
12: Output: $\alpha^k, f(w^k + \alpha^k d^k)$

It is trivial to verify that the following lemma holds.

Lemma 5.3.1. *Algorithm 5.2 is well defined, i.e. it determines in a finite number of steps a scalar α^k such that*

$$f(w^k + \alpha^k d^k) \leq f(w^k) - \gamma\alpha^k\|d^k\|^2. \tag{5.8}$$

Proof. First of all, we prove that Algorithm 5.2 is well-defined. In particular, we show that the while loop cannot infinitely cycle. Indeed, if this was the case, that would imply

$$f\left(w^k + \frac{\zeta^k}{\delta^j} d^k\right) \leq f(w^k) - \gamma\left(\frac{\zeta^k}{\delta^j}\right)\|d^k\|^2 \quad \forall j = 1, 2, \ldots$$

so that for $j \to \infty$ we would have $(\zeta^k/\delta^j) \to \infty$ which contradicts the compactness of $\mathcal{L}(w^0)$ and continuity of f.
In order to prove (5.8), we consider separately the two cases i) $\alpha^k = 0$ and ii) $\alpha^k > 0$. In the first case, (5.8) is trivially satisfied.
In case ii), the condition at step 6 implies

$$f(w^k + \zeta^k d^k) \leq f(w^k) - \gamma \zeta^k \|d^k\|^2.$$

Then, if $\alpha^k = \zeta^k$, the condition is satisfied. Otherwise, if $\alpha^k > \zeta^k$, the stopping condition of the while loop, along with the fact that we already proved that the while loop could not infinitely cycle, is such that we have

$$f(w^k + \alpha^k d^k) \leq f(w^k) - \gamma \alpha^k \|d^k\|^2$$

which again proves (5.8), thus concluding the proof. □

Therefore, the scheme of CMA Light is provided in 5.3.

Algorithm 5.3 Controlled Minibatch Algorithm Light (CMA Light)

Set $\zeta^0 > 0, \theta \in (0,1), \tau > 0$, $\gamma \in (0,1), \delta \in (0,1)$
Let $w^0 \in \mathbb{R}^n, k = 0$
Compute $f(w^0)$ and set $\tilde{f}^0 = f(w^0)$ and $\phi^0 = \tilde{f}^0$
for $k = 0, 1 \ldots$ **do**
  Compute $(\widetilde{w}^k, d^k, \tilde{f}^{k+1}) =$ `Inner_Cycle`(w^k, ζ^k)
  if $\tilde{f}^{k+1} \leq \min\{\phi^k - \gamma\zeta^k, f(w^0)\}$ **then**
    Set $\zeta^{k+1} = \zeta^k$ and $\alpha^k = \zeta^k$ and $\phi^{k+1} = \tilde{f}^{k+1}$
  else
    if $\|d^k\| \leq \tau\zeta^k$ **then**
      Set $\zeta^{k+1} = \theta\zeta^k$ and $\alpha^k = \begin{cases} \zeta^k & \text{if } \tilde{f}^{k+1} \leq f(w^0) \\ 0 & \text{otherwise} \end{cases}$
      Set $\phi^{k+1} = \phi^k$
    else
      $(\widetilde{\alpha}^k, \hat{f}^{k+1}) =$ `EDFL_Light`$(w^k, d^k, \zeta^k; \gamma, \delta)$
      if $\widetilde{\alpha}^k \|d^k\|^2 \leq \tau\zeta^k$ **then**
        Set $\zeta^{k+1} = \theta\zeta^k$ and $\alpha^k = \begin{cases} \widetilde{\alpha}^k & \text{if } \widetilde{\alpha}^k > 0 \text{ and } \hat{f}^{k+1} \leq f(w^0) \\ \zeta^k & \text{if } \widetilde{\alpha}^k = 0 \text{ and } \tilde{f}^{k+1} \leq f(w^0) \\ 0 & \text{otherwise} \end{cases}$
      else
        Set $\zeta^{k+1} = \zeta^k$, and $\alpha^k = \begin{cases} \widetilde{\alpha}^k & \text{if } \widetilde{\alpha}^k > 0 \text{ and } \hat{f}^{k+1} \leq f(w^0) \\ 0 & \text{otherwise} \end{cases}$
      end if
      Set $\phi^{k+1} = \min\{\hat{f}^{k+1}, \tilde{f}^{k+1}, \phi^k\}$
    end if
  end if
  Set $w^{k+1} = w^k + \alpha^k d^k$
end for

We do not assume any further assumptions on the norm of $\nabla f(w)$ or on the relation between

$f(w^k)$ and its estimate $\tilde{f}^k$ but, in order to prove the convergence, we need to impose conditions that ensure the boundness of the output sequence $\{w^k\}$. The underlying idea of CMA Light is avoiding the evaluation of $f(w^k)$ without losing the control on the behaviour of the output sequence. At step 6 we check that the estimate of the objective function is below its initial value and the same condition is imposed every time the EDFL is executed, in steps 15 and in step 17. This additional control is necessary because we do not assume any relation between $f(w^k)$ (which depends only on the point w^k) and $\tilde{f}^k$, which depends on all the gradients $\nabla f_i(w^k)$ for $i = 1, \ldots, P$. This makes easy to prove the following lemma.

Lemma 5.3.2. *Let $\{w^k\}$ be the sequence of points produced by algorithm CMA Light. Then, for all $k = 0, 1, \ldots,$ at least one of the following conditions is satisfied:*

- $\tilde{f}^k \leq f(w^0)$
- $f(w^k) \leq f(w^0)$

Proof. By definition of algorithm CMA Light and recalling Lemma 5.3.1, for every iteration k, we have that one of the following cases happens:

i) step 7 is executed, i.e., we move to a new point such that $\tilde{f}^k \leq \min\{\phi^{k-1} - \gamma\zeta^k, f(w^0)\}$.

ii) step 10 is executed. We move to a point such that $\tilde{f}^k \leq f(w^0)$, or we set $\alpha^k = 0$, i.e., we do not move from w^k.

iii) step 14 is executed. After the linesearch, we move to a point such that either $\tilde{f}^k \leq f(w^0)$ or $f(w^k + \tilde{\alpha}^k d^k) = f(w^{k+1}) \leq f(w^0)$ or we do not move if neither of the conditions is satisfied.

iv) step 16 is executed. After the linesearch we move to a point such that $f(w^k + \tilde{\alpha}^k d^k) = f(w^{k+1}) \leq f(w^0)$ or we do not move.

The proof follows by induction. Let's consider $k = 0$. For $k = 0$ both the conditions are trivially satisfied. We prove that this is true for $k = 1$.
If step 7 is executed, we find w^1 such that $\tilde{f}^1 < \tilde{f}^0 = f(w^0)$. If step 10 is executed we either move w^1 such that $\tilde{f}^1 \leq f(w^0)$, or we have $w^1 = w^0$, which implies that $f(w^1) = f(w^0)$. If step 14 is executed we either accept the step-size returned by the linesearch and find w^1 such that $f(w^0) \leq f(w^1)$, either we decrease step-size and accept w^1 such that $\tilde{f}^1 \leq f(w^0)$, or we have again $w^1 = w^0$, which implies that $f(w^1) = f(w^0)$. Eventually, if step 16 is executed we either accept the step-size returned by the linesearch and find w^1 such that $f(w^1) \leq f(w^0)$, or again $w^1 = w^0$. So, we have that the lemma holds for $k = 1$.
Let's now assume that the lemma is true for a generic k. Recalling the possible cases, we can easily proof it is true for $k + 1$. In fact, we will find w^{k+1} such that one of the following holds:

a) $f(w^{k+1}) \leq f(w^0)$

b) $\tilde{f}^{k+1} \leq f(w^0)$

c) $w^{k+1} = w^k$

If a) or b) holds, then the lemma holds too by definition. If c) holds, we have $f(w^{k+1}) = f(w^k)$ and $\tilde{f}^{k+1} = \tilde{f}^k$, so that in both case the lemma still holds. □

Now, we can prove the boundness of the output sequence of CMA Light $\{w^k\}$, which, differently from the CMA case, is not guaranteed by the compactness of $\mathcal{L}(w^0)$. In fact, the following lemma proves that $\{w^k\}$ cannot diverge even if for some k it happens that $f(w^k) > f(w^0)$.

Lemma 5.3.3. *Let $\{w^k\}$ be the sequence of points produced by algorithm CMA Light. Then, $\{w^k\}$ is limited.*

Proof. Recalling Lemma 5.3.2, we have that the set of index $\{0, 1, \dots\}$ can be represented with the following partition:

- $K_1 = \{k = 0, 1, \cdots : \tilde{f}^k \leq f(w^0), f(w^k) > f(w^0)\}$
- $K_2 = \{k = 0, 1, \cdots : \tilde{f}^k > f(w^0), f(w^k) \leq f(w^0)\}$
- $K_3 = \{k = 0, 1, \cdots : \tilde{f}^k \leq f(w^0), f(w^k) \leq f(w^0)\}$

Let's now suppose by contradiction that $\{w^k\}$ is not limited, i.e., there exists an infinite index set $K \subseteq \{0, 1, \dots\}$ exists such that

$$\lim_{\substack{k\to\infty \\ k\in K}} ||w^k|| = \infty \tag{5.9}$$

We want to show that $K_1 \cap K$ is an infinite index set. Let us assume by contradiction that this is not the case. If this is the case, when $k \to \infty$, $k \in K$, then it must hold that $k \in K_2$ or $k \in K_3$. This implies that for $k \in K$ and sufficiently large, $k \in K_2 \cup K_3$, that is $w^k \in \mathcal{L}(w^0)$. This is in contrast with (5.9) and assumption 5.2.1.
Thus, we have that:

$$\lim_{\substack{k\to\infty \\ k\in K_1\cap K}} ||w^k|| = \infty$$

Recall that $\{\tilde{f}^k\}_{K_1\cap K}$ is limited both from below and above by non-negativity and by the definition of the set K_1. From the definition of $\tilde{f}_k$ and the non-negativity of f_j for all j, we can write

$$\tilde{f}^k = \sum_{i=1}^{P} f_i(\tilde{w}_{i-1}^k) = f_1(\tilde{w}_0^k) + f_2(\tilde{w}_1^k) + \cdots + f_P(\tilde{w}_{P-1}^k) \geq f_1(\tilde{w}_0^k) \tag{5.10}$$

where $\tilde{w}_0^k = w^k$. Since, by coercivity of f_1,

$$\lim_{\substack{k\to\infty \\ k\in K_1\cap K}} f_1(w^k) = \infty,$$

an index $\bar{k} \in K_1 \cap K$ exists such that $f_1(w^{\bar{k}}) > f(w^0)$. Then, by (5.10), we also have

$$\tilde{f}^{\bar{k}} \geq f_1(w^{\bar{k}}) > f(w^0)$$

which contradicts the fact that, by definition of K_1, $\tilde{f}^k \leq f(w^0)$ for all $k \in K_1 \cap K$ and concludes the proof. □

Thanks to this result, we only need to prove that $\lim_{k\to\infty} \zeta^k = 0$ in order to apply Proposition 5.2.4 for the convergence proof. For this purpose we have introduced the monotonically non-increasing sequence of values $\{\phi^k\}$, which does not affect the computational efficiency of CMA Light but compensates the fact the both $\{\tilde{f}^k\}$ and $\{f(w^k)\}$ are not assumed to be non-increasing.

Proposition 5.3.4. *Let $\{\zeta^k\}$ be the sequence of steps produced by Algorithm 5.3, then*

$$\lim_{k\to\infty} \zeta^k = 0.$$

Proof. In every iteration, either $\zeta^{k+1} = \zeta^k$ or $\zeta^{k+1} = \theta\zeta^k < \zeta^k$. Therefore, the sequence $\{\zeta^k\}$ is monotonically non-increasing. Hence, it results

$$\lim_{k\to\infty} \zeta^k = \bar{\zeta} \geq 0.$$

Let us suppose, by contradiction, that $\bar{\zeta} > 0$. If this were the case, there should be an iteration index $\bar{k} \geq 0$ such that, for all $k \geq \bar{k}$, $\zeta^{k+1} = \zeta^k = \bar{\zeta}$. Namely, for all iterations $k \geq \bar{k}$, step 7 or 16 are always executed.
Let us now denote by

$$\begin{aligned} K' &= \{k:\ k \geq \bar{k},\ \text{and step 7 is executed}\} \\ K'' &= \{k:\ k \geq \bar{k},\ \text{and step 16 is executed}\} \end{aligned}$$

Let first proof that K' cannot be infinite. If this was the case, we would have that infinitely many times

$$\gamma\bar{\zeta} \leq \phi^k - \tilde{f}^{k+1} = \phi^k - \phi^{k+1}$$

since, for $k \in K'$, the instructions of the algorithm imply that $\phi^{k+1} = \tilde{f}^{k+1}$. Note that, by the instructions of the algorithm $\{\phi^k\}$ is a monotonically non increasing sequence. Also, for all k, $\phi^k \geq 0$. Hence, the sequence $\{\phi^k\}$ is bounded from below hence it is convergent to a limit $\bar{\phi}$. Then, every sub-sequence of $\{\phi^k\}$ is converging to the same limit. In particular,

$$\lim_{k\to\infty, k\in K'} \phi_k = \lim_{k\to\infty, k\in K'} \phi^{k+1} = \bar{\phi}.$$

That is to say that

$$\lim_{k\to\infty, k\in K'} \phi^k - \phi^{k+1} = 0$$

which is in contrast with $\bar{\zeta} > 0$.

Thus, an index $\hat{k}$ exists such that, $k \in K''$ for $k \geq \hat{k}$. Thanks to Lemma 5.3.1, for $k \geq \hat{k}$, we have

$$f(w^{k+1}) \leq f(w^k + \alpha^k d^k) \leq f(w^k) - \gamma\alpha^k \|d^k\|^2, \tag{5.11}$$

that is the sequence $\{f(w^k)\}$ is definitely monotonically non-increasing and bounded below since

$f(w) \geq 0$. Hence

$$\lim_{k\to\infty} f(w^k) = \lim_{k\to\infty} f(w^{k+1}) = \bar{f}.$$

Then, taking the limit in relation (5.11) we obtain:

$$\lim_{k\to\infty} \alpha^k \|d^k\|^2 = 0.$$

But then, for $k \geq \hat{k}$ sufficiently large, it would happen that

$$\alpha^k \|d^k\|^2 \leq \tau\bar{\zeta}$$

which means that Algorithm CMA Light would execute step 10 setting $\zeta^{k+1} = \theta\bar{\zeta}$ thus decreasing ζ^{k+1} below $\bar{\zeta}$. This contradicts our initial assumption and concludes the proof. □

Now we remark that all the hypothesis required by Proposition 5.2.4 are satisfied. Thus, with the following theorem, we prove the global convergence of CMA Light to a stationary point.

Proposition 5.3.5. *Let $\nabla f_p(w) \in \mathcal{C}^1_L \quad \forall p = 1, \ldots, P$ and let $\{w^k\}$ be the sequence of points produced by Algorithm 5.3. Then, $\{w^k\}$ admits limit points and at least one of them is stationary.*

Proof. By the fact that $\{w^k\}$ is limited, we know that $\{w^k\}$ admits limit points.
Now, let us introduce the following set of iteration indices

$$K = \{k : \ \zeta^{k+1} = \theta\zeta^k\}$$

and note that, since $\lim_{k\to\infty} \zeta^k = 0$, K must be infinite. Let $\bar{w}$ be any limit point of the sub-sequence $\{w^k\}_K$, then, by Proposition 5.2.4, a subset $\bar{K} \subseteq K$ exists such that

$$\lim_{k\to\infty, k\in\bar{K}} w^k = \bar{w}, \tag{5.12a}$$

$$\lim_{k\to\infty, k\in\bar{K}} d^k = -\nabla f(\bar{w}), \tag{5.12b}$$

$$\lim_{k\to\infty, k\in\bar{K}} \zeta^k = 0 \tag{5.12c}$$

Then, let us now split the set $\bar{K}$ into two further subsets, namely,

$$\begin{aligned} K_1 &= \{k \in \bar{K} : \ \|d^k\| \leq \tau\zeta^k\}, \\ K_2 &= \{k \in \bar{K} \setminus K_1 : \widetilde{\alpha}^k \|d^k\|^2 \leq \tau\zeta^k\}. \end{aligned}$$

Note that, K_1 and K_2 cannot be both finite.
First, let us suppose that K_1 is infinite. Then, the definition of K_1, (5.12b) and (5.12c) imply that

$$\lim_{k\to\infty, k\in K_1} \|d^k\| = \|\nabla f(\bar{w})\| = 0$$

thus concluding the proof in this case.

Now, let us suppose that K_2 is infinite. In this case, we proceed by contradiction and assume that $\bar{w}$ is not stationary, i.e. $\|\nabla f(\bar{w})\| > 0$. By the definition of K_2 and (5.12c), we obtain

$$\lim_{k\to\infty, k\in K_2} \widetilde{\alpha}^k \|d^k\|^2 = 0,$$

which, recalling (5.12b) and the fact that $\|\nabla f(\bar{w})\| > 0$, yields

$$\lim_{k\to\infty, k\in K_2} \widetilde{\alpha}^k = 0. \tag{5.13}$$

Now, let us further partition K_2 into two subsets, namely

i) $\bar{K}_2 = \{k \in K_2 : \ \widetilde{\alpha}^k = 0\}$

ii) $\widehat{K}_2 = \{k \in K_2 : \ \widetilde{\alpha}^k > 0\}$.

Let us first suppose that $\bar{K}_2$ is infinite. This means that, by the definition of algorithm 5.2,

$$\frac{f(w^k) - f(w^k + \zeta^k d^k)}{\zeta^k} < \gamma \|d^k\|^2.$$

Then, by the Mean-Value Theorem, a number $\hat{\alpha}^k \in (0, \zeta^k)$ exists such that

$$-\nabla f(w^k + \hat{\alpha}^k d^k)^T d^k < \gamma \|d^k\|^2.$$

Taking the limit in the above relation, considering (5.12a), (5.12b) and (5.12c), we obtain

$$\|\nabla f(\bar{w})\|^2 \leq \gamma \|\nabla f(\bar{w})\|^2$$

which, recalling that $\gamma \in (0, 1)$, gives

$$\|\nabla f(\bar{w})\| = 0,$$

which contradicts $\|\nabla f(\bar{w})\| > 0$. Now, let us suppose that $\widehat{K}_2$ is infinite. This means that

$$\frac{f(w^k) - f(w^k + (\widetilde{\alpha}^k/\delta) d^k)}{\widetilde{\alpha}^k/\delta} < \gamma \|d^k\|^2.$$

Then, reasoning as in the previous case, and considering (5.13), we again can conclude that $\|\nabla f(\bar{w})\| = 0$, which is again a contradiction and concludes the proof. □

5.4 Numerical experiments

In this section we report the computational experiments we have carried out to assess CMA Light performance against different algorithms.
CMA Light has been implemented on Python 3.9, using the automatic differentiation tools for backward propagation provided by the open-source library *Pytorch 2.0*[1]. As CMA and NM-

[1] https://pytorch.org/

CMA, CMA Light is built as a `torch.Optimizer` object and, jointly with the fully reproducible numerical results, is available on the public Github repository https://github.com/corrado coppola97/CMA_Light. All the tests have been conducted on GPU NVIDIA GP106GL Quadro P2200 5 GB @ 60Hz.
The tests are presented in three different subsections. In subsection 5.4.1, we briefly compare CMA and CMA Light performances in terms of computational efficiency on the same experimental testbed used in section 4.6 for CMA. Notice that the comparison between CMA and CMA Light is presented in a separate subsection because CMA Light is significantly more efficient. Both CMA and CMA Light may require the evaluation of the objective function, whereas their main competitors never do. If the results were reported together, they could be skewed and biased due to the inclusion of CMA, which differs greatly from the other methods. In subsection 5.4.2, we discuss the tests of CMA Light again on the same problems in Table 4.2 against its competitors. Finally, in subsection 5.4.3, we report the tests of CMA Light with cross-entropy loss function (2.20) on the image classification datasets CIFAR10 and CIFAR100[2] using the residual network architectures [88].
As a benchmark for CMA Light we have used four of the most used optimizers in deep learning:

- SGD [176] with decreasing stepsize the initial learning rate set to 0.01, as for CMA Light. The stepsize decreasing rule is taken from the `stepLR` scheduler, implemented in Pytorch 2.0 as an additional tool for SGD [3].
- Adaptive gradient methods Adam [104], Adagard [60], and Adadelta [217] with the Pytorch default hyper-parameters.

In Table 5.1 we report the hyper-parameters setting of CMA Light used for the tests.

	ζ^0	τ	θ	γ	δ
CMA Light	0.01	10^{-2}	0.75	10^{-2}	0.9

Table 5.1: Hyper-parameters settings in CMA Light

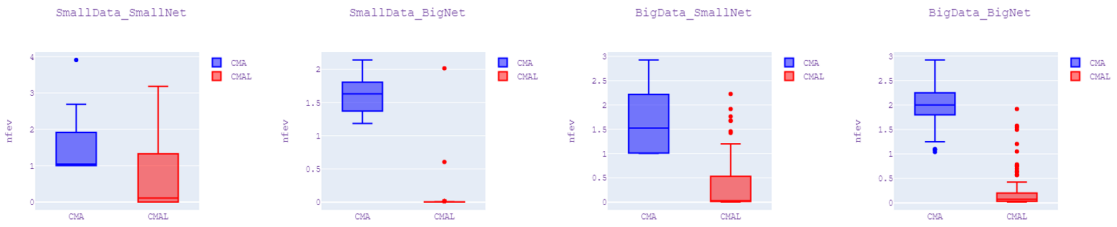
To assess the performance of CMA Light with respect to its competitors, we have followed a two-sided approach.
On the one hand, we are interest in evaluating the efficiency of CMA Light in solving the optimization problem (2.6). For this purpose, we use the performance profiles curve on the training loss, as introduced by [54, 152] and described in detail in section 4.6.
On the other hand, we are also interested in assessing the overall generalization performance on a given task. For the first type of experiments, we measure it by computing the final value of the MSE obtained on the test set. For the second type of experiments, we use the test accuracy on the multi-class image classification tasks.

[2] https://www.cs.toronto.edu/~kriz/cifar.html
[3] https://pytorch.org/docs/sTable/generated/torch.optim.lr_scheduler.stepLR.html

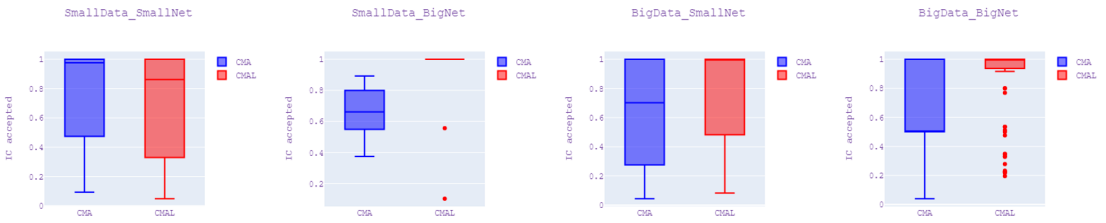
5.4.1 CMA vs CMA Light on regression tasks

We have trained the seven different deep networks reported in Table 4.1 on the 11 open-source datasets from Table 4.2. The training process consisted in the minimization of the MSE loss function. Analogously to what was done in the previous chapter, the numerical tests have been conducted performing 5 multi-start runs and setting a time limit of 300 seconds for each problem, i.e., for each pair (dataset, architecture). This resulted in $11 \times 7 \times 5 = 385$ runs for every optimizer, which means approximately 35 hours of GPU time. We remark that in this case the dimension of problem (2.6) is up to few millions of variables and 500 thousands of samples (see Table 4.2).

Looking at Figure 5.1(a), we observe that, as expected, CMA Light requires significantly less average number of objective function evaluations per epoch, leading to a significant reduction of computational effort. This holds, despite the acceptance rate of the tentative point (box-plot in Figure 5.1(b)) is not always higher in CMA Light than in CMA (notice that for small networks it is even lower). This result is not surprising, since our computational experiments show that the estimate of the objective function (5.2) is often an upper approximation for the real $f(w^k)$. Recalling the definition, $\tilde{f}^k$ is computed using the sequence of points produced by Algorithm 5.1, which means that it is equally influenced by the samples $i = 1, \dots, P$. Our computational experiments show that the anti-gradients $-\nabla f_i$ are often a descent direction even for $f(w)$, which means that the points found in the first inner iteration will correspond to worse value of the objective function.

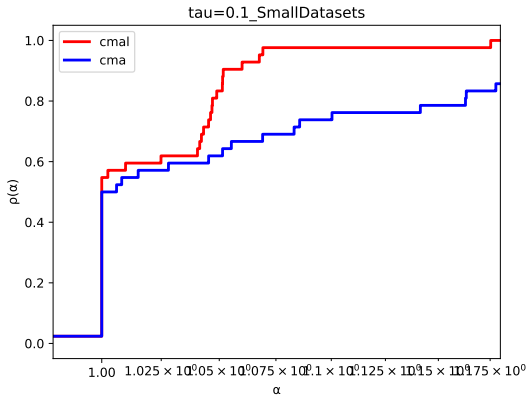


(a) Average number of function evaluations per iteration

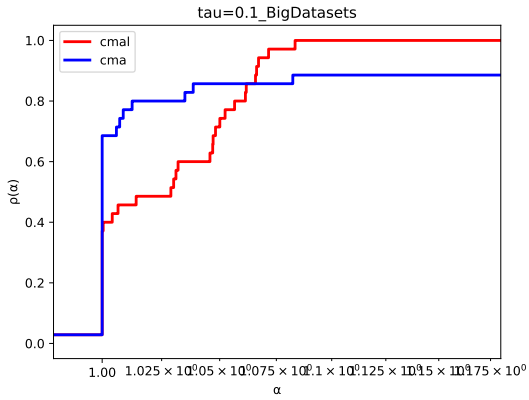


(b) Acceptance of the trial point $\tilde{w}^k$

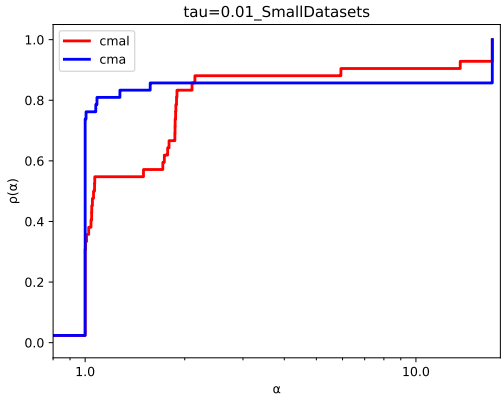
Figure 5.1: Boxplot on all the runs: comparison of CMA Light and CMA on the average number of function evaluations and acceptance rate of $\tilde{w}^k$.



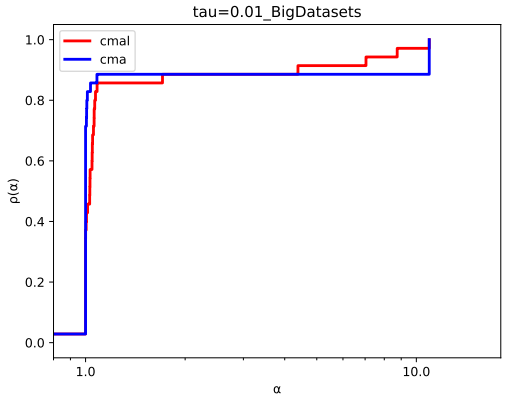
(a) Small datasets: $\tau = 10^{-1}$



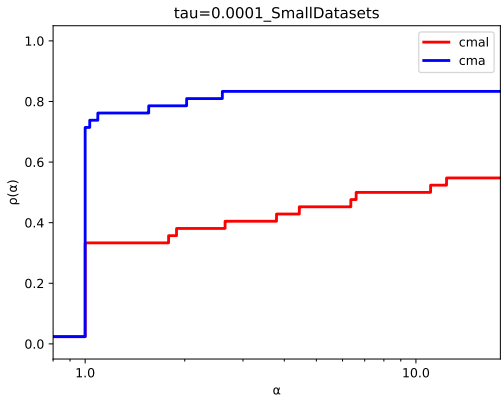
(b) big-size datasets: $\tau = 10^{-1}$



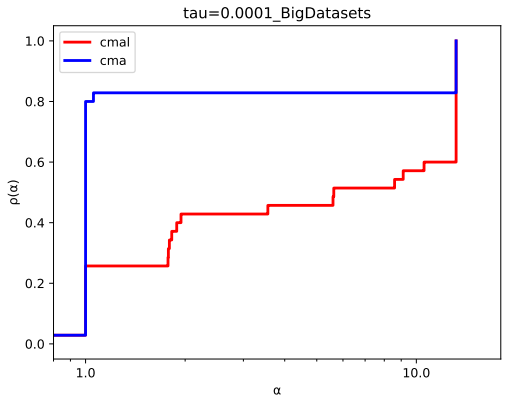
(c) Small datasets $\tau = 10^{-2}$



(d) big-size datasets $\tau = 10^{-2}$



(e) Small datasets $\tau = 10^{-4}$



(f) big-size datasets $\tau = 10^{-4}$

Figure 5.2: Performance profiles of CMA against CMA Light. Results for small datasets are reported to the left, and for big datasets to the right.

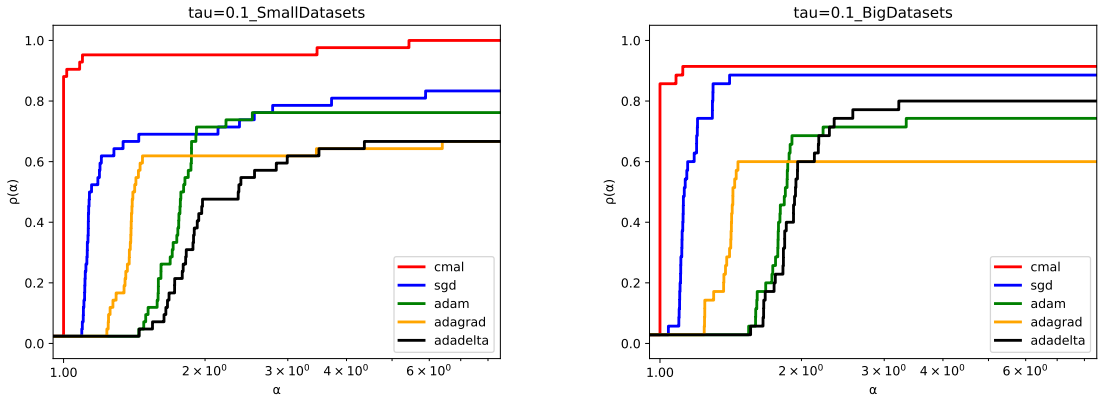
5.4.2 Tests with MSE error function

For this class of tests, the experimental testbed, as well as the training methodology and the multi-start strategy, does not differ from what reported at the beginning of the previous subsection. However, we remind the reader that this resulted in $11 \times 7 \times 5 = 385$ runs for five

optimizer, i.e., a total number of 1925 runs. These tests required approximately 170 hours of GPU time, including the training set-up and the dataset upload.
In Figure 5.3 we report the performance profiles on the training loss for different levels of precision τ. We observe that, for $\tau = 10^{-1}$ and $\tau = 10^{-2}$, CMA Light clearly outperforms the other methods, on both small and big datasets. We also notice that SGD and CMA Light, which involve more simple operations and do not require additional memory storage to accumulate the past gradients, achieve much better efficiency performance than their adaptive competitors. This behaviour is confirmed even for high precision level $\tau = 10^{-4}$. SGD and CMA Light converge faster and have a higher success rate than the other optimizers, which also reflect their stronger convergence properties. Nonetheless, for high precision level, SGD outperforms CMA Light. We suppose this happens do to the evaluations of $f(w^k)$ that are performed when executing the `EDFL_Light`.
For the sake of completeness, in the appendix we also report (in tables A.1 and A.2) the final values of the test loss averaged over the five runs.

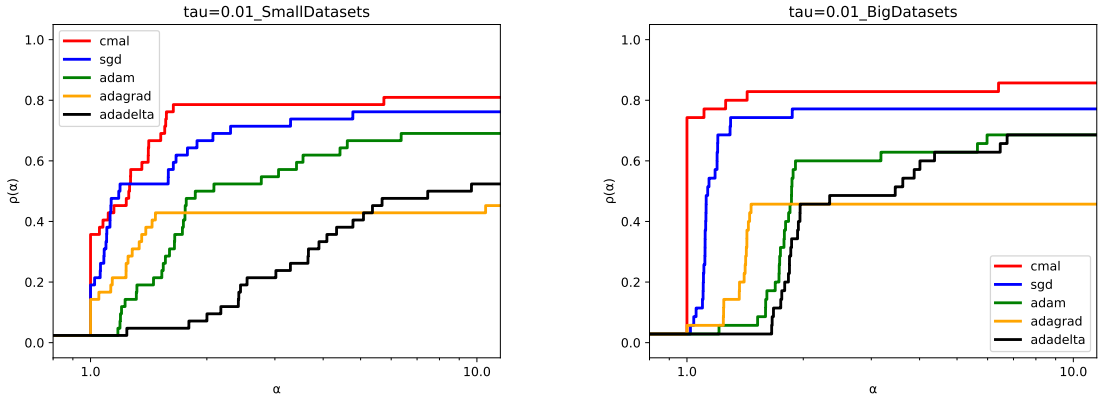
5.4.3 CMA Light on large-scale image-classification tasks

Considering that CMA Light has an increasing advantage over the other algorithms on large-scale problems, we have carried out another class of tests on the image classification datasets CIFAR10 and CIFAR100 with ResNet architectures [88].
CIFAR10 is an image classification dataset with 10 classes. It has 50.000 RGB ($3 \times 32 \times 32$) training images. CIFAR100 has the same structure but the task is harder, since there are 100 different types of objects.
The ResNet architectures are five different convolutional neural networks with residual connections, introduced specifically for the multi-class image classification problem with the aim of solving the vanishing gradient effect [88]. More details on ResNet architecture can be found in Chapter 2 (section 2.3.2).
The target optimization problem, in this case, is the unconstrained minimization of the cross-entropy loss (CEL (2.20)). In Table 5.2 we report the number of variables of problem (2.6), depending on the pair (dataset,architecture). While CIFAR10, and CIFAR100 are still considered basic image classification tasks, the number of variables is between one and two orders of magnitude larger than in datasets in Table 4.2. The main purpose of this test is to assess the scaling capability of CMA Light when increasing the problem dimension.
Conducting these numerical tests required significantly higher computational effort than for the previous ones because we needed the real value of $f(w^k)$ after each epoch to plot the performance profiles. To optimize the consumption of hardware resources, we performed a single-run test using the state-of-the-art network initialization from ImageNet [155, 211] and setting a fixed number of of 50 epochs. The number of epochs has been chosen following the same heuristic approach described by [53, 214], seeking for a minimization of the computational effort required. A single-run train took an average of 4 hours of GPU-time. Considering the five algorithms, the five networks and the two datasets and including the set-up times, the tests presented in this sections required approximately 215 hours of GPU-time.



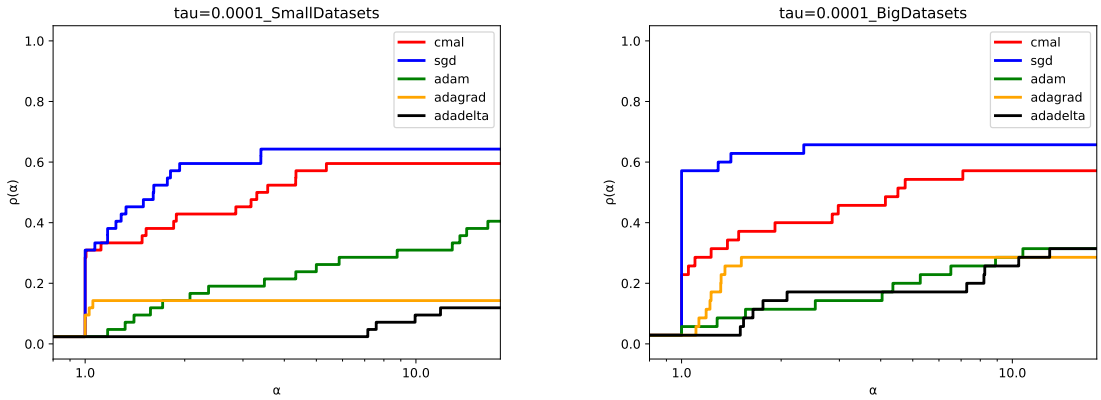
(a) Small datasets: $\tau = 10^{-1}$

(b) big datasets: $\tau = 10^{-1}$



(c) small datasets $\tau = 10^{-2}$

(d) big datasets $\tau = 10^{-2}$



(e) small datasets $\tau = 10^{-4}$

(f) big datasets $\tau = 10^{-4}$

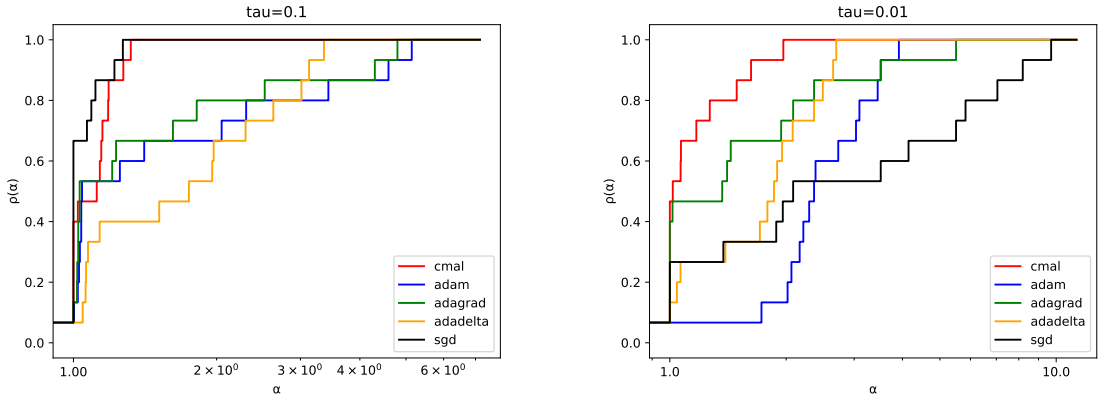
Figure 5.3: Performance profiles on the training loss on all the networks for different values of τ. Results for small datasets are reported to the left, and for big datasets to the right.

As done for the tests on the problems in Table 4.2, in Figure 5.4 we report the performance profiles on the training loss for different levels of precision τ. We observe that, for lower precision levels $\tau = 10^{-1}$ and $\tau = 10^{-2}$ (Figures 5.4(a) and 5.4(b)), CMA Light outperforms its adaptive competitors, which highlights its enhanced computational efficiency in achieving faster an approximated solution of (2.6). However, the advantage of CMA Light decreases for high

Dataset	P	d	n in M ResNet-18	ResNet-34	ResNet-50	ResNet-101	ResNet-152
CIFAR10	50.000	3.072	11.18	21.29	23.53	42.52	58.16
CIFAR100	50.000	3.072	11.17	21.28	23.52	42.51	58.16

Table 5.2: Dataset description (P = number of training samples; d= number of input features) and n =number of variables (in M) of the optimization problems corresponding to the network architecture.

precision levels. We observe in Figure 5.4(c) and 5.4(d) that Adagrad and Adadelta achieves a better efficiency performance. We argue that this happens because, when w^k is close to a stationary point, CMA Light executes the `EDFL_Light` more frequently, meaning the full $f(w)$ is evaluated.



(a) $\tau = 10^{-1}$ (b) $\tau = 10^{-2}$

(c) $\tau = 10^{-3}$ (d) $\tau = 10^{-4}$

Figure 5.4: Performance profiles on the training loss on all the residual networks for different values of τ - CMA Light vs SGD and adaptive methods

Finally, in Table 5.3, we report the test accuracy values obtained at the end of the training process, and we observe that CMA Light is almost always the best-performing optimizer. In the appendix, in figures A.4, A.5,A.6,A.7,A.8 we report the training loss and the test accuracy plot against time during training, observing that CMA Light achieves better level of accuracy faster than its competitors. Seeking for a more reliable assessment of the CMA Light performance

against the most commonly used optimizers in deep learning, we also report the one-by-one comparisons of CMA Light against Adam (figure A.9) and Adagrad (figure A.10).

	Dataset	ResNet-18	ResNet-34	ResNet-50	ResNet-101	ResNet-152
CIFAR10	Adam	0.7429	0.7347	0.7496	0.7163	0.6815
	Adagrad	0.7432	0.7451	0.7141	0.6928	0.7209
	Adadelta	0.7464	0.7446	0.7499	**0.7336**	0.7574
	SGD	0.6202	0.6421	0.6486	0.6434	0.6402
	CMA Light	**0.7580**	**0.7544**	**0.7808**	0.7219	**0.7783**
CIFAR100	Adam	0.4069	0.4309	0.3991	0.3201	0.3280
	Adagrad	0.3804	0.4003	0.3793	0.2944	0.3042
	Adadelta	0.4237	0.4469	0.4614	0.4556	0.4359
	SGD	0.3141	0.3477	0.3541	0.3936	0.3579
	CMA Light	**0.4632**	**0.4724**	**0.5159**	**0.4805**	**0.5045**

Table 5.3: Test accuracy (in %) obtained with the different neural networks on all the image classification datasets with all the optimizers at the end of the training process.

5.5 BD-CMAL: A block-decomposed version of CMA Light

In this section we introduce Block-Decomposed CMA Light (BD-CMAL), a block-decomposition heuristics based on CMA Light, in which at each iteration only a subset of the variables is updated, while the others are kept fixed at their current value. To leverage the additive structure of the objective function, we embed the decomposition scheme into the minibatch framework, exploiting both sample-wise decomposition and block-wise decomposition. In other words, at each inner iteration, a subset of variables is updated using only a mini-batch. The aim of this strategy is to reduce the computational cost of the optimization process by mitigating not only the issues caused by a large number of samples P through mini-batch methods, but also the difficulties posed by a huge number of variables n through variable decomposition.

Notation: For the sake of clearness, we state the specific notation used throughout this section, which slightly differs from the CMA Light notation. $w \in \mathbb{R}^n$ is the vector of all the network parameters, $w_\ell \in \mathbb{R}^{|\ell|}$ is the subvector of the weights belonging to layer $\ell \in \{1, \dots, L\}$. Notice that we use the term *layer* in a broad sense, denoting any subset of network parameters composing a block of variables. The set of all the network layers, denoted by $\mathcal{L} = \{1, \dots, L\}$, is a generic partition of the network parameters. Therefore $\sum_{\ell=1}^{L} |w_\ell| = n$.
$\nabla_w f_i \in \mathbb{R}^n$ represents the estimate of the gradient considering only the samples belonging to batch i, instead of all the P samples. We denote with $\nabla_{w_\ell} f_i \in \mathbb{R}^{|\ell|}$ the partial derivative computed with respect to the block of variables w_ℓ. We denote with $[\cdot]_\ell$ the vector where all the components are set to zero except for those of the ℓ-th layer. Consequently, $\nabla_w f_i$ can be expressed as $\sum_{\ell=1}^{L} [\nabla_{w_\ell} f_i]_\ell$.
In our proposed scheme, we consider the possibility of updating only a subset of the variables at each iteration, denoting the set of layers that are updated at iteration k with $\widetilde{\mathcal{L}}^k$. Consequently,

the direction d^k is evaluated using only the gradient computed with respect to the variables $w_\ell, \ell \in \widetilde{\mathcal{L}}^k$. The set of the layers not updated at epoch k is denoted as $\bar{\mathcal{L}}^k = \mathcal{L} \setminus \widetilde{\mathcal{L}}^k$.

5.5.1 Outline of the method

As mentioned earlier, BD-CMAL cannot be directly compared to CMA or CMA Light as a fully convergent algorithm, since no formal convergence theory has been developed for it yet. However, two element of novelty make this algorithm substantially different from CMA Light: the use of a decomposed version of Algorithm 5.1 for the inner cycle and the introduction of a deactivation heuristic to select the layers to be *frozen* at a given epoch.
To understand BD-CMAL, we first focus on how the standard CMA/CMA Light inner iterations are replaced by a mini-batch method using a Decomposed Inner Cycle, as shown in Algorithm 5.4. In this approach, only a subset of variables is updated during each epoch. During each inner iteration of the Decomposed Inner Cycle, this subset of variables $w_\ell, \ell \in \widetilde{\mathcal{L}}^k$ is iteratively updated using a mini-batch of samples for each step.

Algorithm 5.4 `Decomposed Inner Cycle`

Input: w^k, ζ^k, $\widetilde{\mathcal{L}}^k$
Set $w_0^k = w^k$
Let $I^k = \{h_1^k \dots h_P^k\}$ be a permutation of $\{1, \dots P\}$
for $i = 1, \dots, P$ **do**
    $\widetilde{f}_i^k = f_{h_i}(w_{i-1}^k)$
    $\widetilde{d}_i^k = -\sum_{\ell \in \widetilde{\mathcal{L}}^k} [\nabla_{w_\ell} f_{h_i}^k(w_{i-1}^k)]_\ell$
    $w_i^k = w_{i-1}^k + \zeta^k \tilde{d}_i^k$
end for
Output $w^k = w_P^k$, $d^k = \sum_{i=1}^{P} \widetilde{d}_i^k$ and $\widetilde{f}^k = \sum_{i=1}^{P} \widetilde{f}_i$

Differently from a traditional decomposition method, we do not update the variables in a sequential order or in parallel. Instead, we dynamically select the blocks of variables to be updated at each iteration, while leaving the others unchanged. This can be considered as a training method that leverages the decomposition over variables to enhance computational efficiency performances. At each epoch, after having executed the Decomposed Inner Cycle, we add some additional controls to determine whether it is beneficial to deactivate or reactivate some blocks of variables. This results in an heuristic deactivation strategy.
The selection of the parameters to be updated at each epoch aims to reduce the computational effort by considering only the most relevant variables at that step of the training procedure. When a block is active, its variables are updated during the optimization step of the current epoch. Instead, when convenient, some blocks of variables are left unchanged during one or more epochs, reducing the dimensionality of the optimization problem and decreasing its computational time.
Specifically, at each epoch we consider one candidate block (denoted as b^-) for deactivation and another candidate block (denoted as b^+) for reactivation. These candidate blocks are determined

based on an architectural importance heuristic, which assesses the relevance of each block within the neural network architecture considering its impact on both computational time and accuracy. The process of identifying these candidate blocks is called *ablation test* (see [192] for more details). It involves systematically removing parts of the neural network to evaluate and sort the different impacts on performance metrics. The underlying idea is that if removing a block has minimal impact on the accuracy loss while leading to a significant computational gain, it should be considered as a candidate for deactivation. Conversely, if deactivating a block caused a substantial decrease in accuracy without significant reduction in computational time, it should not be deactivated; rather, it should be a prime candidate for reactivation. These considerations will be reflected in an ordered list prioritizing blocks considering their overall importance, based on their impact on accuracy and computational time.

This decision on whether to deactivate the candidate block b^- or not, is based on a certain deactivation threshold, computed as a function of the norm of the parameter updates of the currently active blocks. If the norm of the update of block b^- is below this threshold, then it can be assumed that these parameters do not play a major role at that training stage and are not significantly contributing to the current training iteration. Therefore, the corresponding variables can be deactivated in order to reduce the computational time. Subsequently, to avoid an excessive loss in accuracy, we compute the validation accuracy at each iteration to determine whether some variables should be reactivated. We use a counter to keep track of the number of consecutive epochs without sufficient improvement in the validation accuracy. If there is no sufficient improvement for a certain number of consecutive epochs (defined by the *patience* parameter), the current set of active blocks may not be sufficient for an effective training. Consequently, the candidate block b^+ is reactivated. This possibility of deactivating and reactivating blocks of parameters based on conditions on the training performance allows to find a good trade-off between reducing the computational cost and maintaining accuracy.

Given $\mathcal{L}$, which represents the set of all the network blocks ordered by their importance, we initialized the sets $\widetilde{\mathcal{L}}^0 = \mathcal{L}$ and $\bar{\mathcal{L}}^0 = \emptyset$, updating all the network parameters for a certain number of epochs determined by the parameter $\bar{k}$. After a number $\bar{k}$ of epochs, in which all the network parameters are updated, we start to check whether it is convenient to deactivate or reactivate some blocks of parameters using the scheme reported in Algorithm 5.5.

When a block is deactivated, it is removed from the list of the blocks to be updated in the subsequent iteration $\widetilde{\mathcal{L}}^{k+1}$ and added to the list of the deactivated blocks $\bar{\mathcal{L}}^{k+1}$. Both of these lists are built containing the blocks in ascending importance order. Consequently, at epoch k, the deactivation candidate b^- is the first block of the list $\widetilde{\mathcal{L}}^k$, while the reactivation candidate b^+ is the last element of the list $\bar{\mathcal{L}}^k$.

In this scheme, A^k represents the validation accuracy computed at iteration k, while β is a parameter used to regulate the inclination to deactivate some blocks of parameters, so that increasing β leads to a higher degree of parameter deactivation. The parameter ρ defines the minimum increase in validation accuracy we aim to achieve at each iteration. Finally, we use the counter *count* to keep track of the number of consecutive epochs without sufficient improvement in the validation accuracy. When the counter reaches the *patience*, the candidate block b^+ is

Algorithm 5.5 `Deactivation/activation scheme over blocks of variables`

Input: $\widetilde{\mathcal{L}}^k, \text{count}^k$
Set deactivation threshold $= \beta \frac{1}{|\widetilde{\mathcal{L}}^k|} \sum_{\ell \in \widetilde{\mathcal{L}}^k} \psi(\|w_\ell^k - w_\ell^{k-1}\|)$
$\bar{\mathcal{L}}^k = \mathcal{L} \setminus \widetilde{\mathcal{L}}^k$
$b^- = \widetilde{\mathcal{L}}^k_1$, $b^+ = \bar{\mathcal{L}}^k_{-1}$
$\widetilde{\mathcal{L}}^{k+1} = \widetilde{\mathcal{L}}^k$
if $\|w_{b^-}^k - w_{b^-}^{k-1}\| <$ deactivation threshold **then**
  $\widetilde{\mathcal{L}}^{k+1} = \widetilde{\mathcal{L}}^k \setminus \{b^-\}$
end if
Compute A^k
if $A^k < \rho A^{k-1}$ **then**
  $\text{count}^k = \text{count}^k + 1$
  $A^k = A^{k-1}$
  if $\text{count}^k =$ patience **then**
    $\text{count}^{k+1} = 0$
    $\widetilde{\mathcal{L}}^{k+1} = \widetilde{\mathcal{L}}^{k+1} \cup \{b^+\}$
  else
    $\text{count}^{k+1} = \text{count}^k$
  end if
else
  $\text{count}^{k+1} = 0$
end if
Output $\widetilde{\mathcal{L}}^{k+1}$, count^{k+1}

reactivated and the counter is set to zero. The deactivation threshold is computed as the average of a function ψ of the norm update of the active blocks, multiplied by the parameter β. In our tests we used ψ as the identity function.

Finally, we report the complete scheme of the algorithm, which incorporate the decomposed inner cycle and the additional controls into CMA Light. The algorithm starts by initializing the parameters and variables. It enters the main loop over the epochs, beginning with the Decomposed Inner Cycle, where the variables belonging to $\widetilde{\mathcal{L}}^k$ are updated. If the iteration count exceeds the number $\bar{k}$ of epochs during which all the parameters are updated, the blocks to be updated in the following epoch are selected through the Deactivation/Activation scheme. In addition, when necessary, the Extrapolation Derivative-Free Linesearch is executed. In the final scheme below, the monotonically non-increasing sequence $\{\phi^k\}$ introduced in CMA Light, is used to check the watchdog condition of sufficient decrease. The use of this sequence compensates the fact that $\{\tilde{f}^k\}$ is not assumed to be non-increasing and was utilized to prove the convergence of CMA Light.

5.5.2 Numerical results

In this subsection, we report the early numerical results comparing the implementations of CMA Light and its block decomposed version described in Algorithm 5.6, denoted as BD-CMAL. As

Algorithm 5.6 `Block-Decomposed CMA Light (BD-CMAL)`

Let $\zeta^0 > 0$, $w^0 \in \mathbb{R}^n$, $\theta \in (0,1)$, $k = 0$, $\bar{k}$, $\text{count}^0 = 0$, β, ρ, patience and $\widetilde{\mathcal{L}}^0 = \mathcal{L}$
for $k = 0, 1, 2, \ldots$ **do**
  Compute $(w^k, d^k) =$ `Decomposed Inner Cycle`$(w^k, \zeta^k, \widetilde{\mathcal{L}}^k)$
  if $k > \bar{k}$ **then**
    Compute $(\widetilde{\mathcal{L}}^{k+1}, \text{count}^{k+1}) =$ `Deactivation/Activation scheme` $(\widetilde{\mathcal{L}}^k, \text{count}^k)$
  end if
  if $\tilde{f}^{k+1} \leq \min\{\phi^k - \gamma\zeta^k, f(\omega^0)\}$ **then**
    Set $\zeta^{k+1} = \zeta^k$, $\alpha^k = \zeta^k$ and $\phi^{k+1} = \tilde{f}^{k+1}$
  else
    Set $(\tilde{\alpha}^k, \hat{f}^{k+1}) =$ `Extrapolation Derivative-Free Linesearch`
    Set $\alpha^k = \begin{cases} \tilde{\alpha}^k & \text{if LS successful} \\ 0 & \text{otherwise} \end{cases}$, $\zeta^{k+1} = \begin{cases} \zeta^k & \text{if LS is successful} \\ \theta\zeta^k & \text{otherwise} \end{cases}$
    Set $\phi^{k+1} = \min\{\hat{f}^{k+1}, \tilde{f}^{k+1}, \phi^k\}$
  end if
  $w^{k+1} = \begin{cases} w^k + \alpha^k d^k & \text{the RR/IG point} \\ w^k & \text{restart} \\ w^k + \tilde{\alpha}^k d^k & \text{longer step along } d_k \end{cases}$
end for

for the previous section, the tests were conducted on the image classification tasks CIFAR10 and CIFAR100 with the five state-of-the-art residual networks (see [88] and section 2.3.2). We used the same hardware GPU NVIDIA Quadro P2200 4 GB and carried out a single 50 epoch run with the default ImageNet weight initialization [110]. The aim of these experiments is to evaluate the efficiency and effectiveness of BD-CMAL when applied to large-scale problems. Our analysis focuses on improvements in terms of computational time and the impact on the performance of training loss, training accuracy and test accuracy. To ensure a fair comparison, we used the default hyperparameters setting for both CMA Light and BD-CMAL, reported in the following Table 5.1. In addition to these settings, the decomposed version of the CMA Light algorithm includes the hyperparameters of the `Deactivation/activation scheme` for the selection of the blocks to be updated. These parameters include the number of epochs in which we first update all the network blocks $\bar{k}$, the parameter β to regulate the deactivation threshold, the parameter ρ to evaluate the sufficient improvement in validation accuracy and the *patience*. The selected values are reported in Table 5.4.

Table 5.4: Deactivation/activation scheme hyperparameter settings

β	ρ	$\bar{k}$	*patience*
1.25	1.005	5	2

The obtained results are summarized in the Table 5.5, which provide performance metrics at epoch 50 of average training loss, test accuracy, training accuracy and computational time.
The obtained results demonstrate the potential of BD-CMAL to achieve significant time gain with only a slight loss in test accuracy. It is important to note that the hyperparameters for block deactivation were determined through a trial and error process on the sole ResNet-18 architecture. Consequently, when applied to other architectures, the block deactivation mechanism

Dataset	Net	Algorithm	Train Loss	Test Acc	Train Acc	Time
CIFAR 10	ResNet-18	CMA Light	2.99e-5	76.01%	99.93%	3313.09
		BD-CMAL	2.23e-4	74.56%	99.81%	1399.01
	ResNet-34	CMA Light	1.68e-5	76.62%	99.97%	5725.39
		BD-CMAL	2.36e-5	76.40%	99.96%	5663.57
	ResNet-50	CMA Light	1.71e-5	77.67%	99.98%	5206.62
		BD-CMAL	2.26e-5	75.83%	99.97%	5085.57
	ResNet-101	CMA Light	9.92e-4	76.74%	99.80%	9460.65
		BD-CMAL	4.34e-5	68.48%	99.55%	8632.54
	ResNet-152	CMA Light	1.06e-5	76.22%	99.98%	10912.27
		BD-CMAL	8.57e-6	78.20%	99.98%	10651.00
CIFAR 100	ResNet-18	CMAL	2.86e-4	46.27%	99.90%	3088.51
		BD-CMAL	3.51e-4	46.49%	99.83%	3052.19
	ResNet-34	CMA Light	1.45e-4	48.17%	99.89%	5178.58
		BD-CMAL	1.91e-4	47.34%	99.51%	5244.39
	ResNet-50	CMA Light	1.02e-4	50.94%	99.91%	4874.15
		BD-CMAL	1.13e-4	51.69%	99.93%	4661.35
	ResNet-101	CMA Light	8.08e-5	50.72%	99.96%	7785.69
		BD-CMAL	6.89e-5	50.48%	99.97%	7832.62
	ResNet-152	CMA Light	7.29e-5	51.63%	99.99%	10250.25
		BD-CMAL	1.05e-4	50.11%	99.97%	10291.73

Table 5.5: Results on CIFAR10 with ResNet-18

did not always work effectively. Looking at ResNet-18, BD-CMAL significantly reduced training time compared to CMA Light, with only a slight drop in test accuracy. This indicates that the block deactivation in BD-CMAL was effective for ResNet-18, demonstrating its potential for improved computational efficiency. However the reduction in terms of computational effort is slighter for the other networks, where some blocks were deactivated only in the latest epochs. This also leads to some inconsistencies in the results, especially on the CIFAR100 dataset where, on ResNet-152 and ResNet-101, BD-CMAL is even slower.

Although these early results indicate the need for hyper-parameter tuning (as shown in Table 5.4), and the configuration found for ResNet-18 may not be fully transferable to other tasks, they still demonstrate a potential efficiency gain when the hyper-parameters are optimized for the specific problem. The efficiency of this approach, which includes ablation testing to rank the importance of different blocks and hyper-parameter optimization, remains uncertain and is the focus of ongoing research. This method may only turn out to be useful when similar tasks are repeated multiple times, e.g., training networks with slight architectural variations [44].

5.6 Conclusions

In this chapter we have presented a novel algorithm of the CMA family, which was deeply inspired by the recent findings in the field of objective function-free optimization. Despite the

use of a deterministic incremental rule for the sample selection, CMA Light outperforms all its competitors in almost all settings in terms both of training efficiency and of generalization performance. While we did not investigate the number of line-search iterations and failed epochs as a function of the probability that the search direction is a "good" descent direction (which can certainly be matter of future research), our experiments show that, for many state-of-the-art tasks, the computational overhead of CMA Light line-search is more than compensated by the increased training efficiency and generalization performance. Moreover, the better generalization performance of networks trained with CMA Light suggests that our algorithm is a more efficient and flexible "teacher". Unlike classical non-adaptive methods, which often require a computationally intensive grid search to tune the step-size, CMA Light automatically adjusts the learning rate during training with the line-search. While quantifying this advantage precisely is challenging, it is important to highlight that CMA Light, through its occasional activation of a line-search, imitates the adaptive mechanism of algorithms like Adam, without needing the same amount of memory to store the past mini-batch gradients. The next step of this research, which we will present in the next chapter, involves enhancing the efficiency of th eline-search procedure itself and extending the CMA Light convergence theory to the case where samples are randomly reshuffled after each epoch.

Chapter 6

Beyond adaptive gradient: Fast-Controlled Minibatch Algorithm for large-scale optimization

In this chapter we introduce the Fast-Controlled Minibatch Algorithm (F-CMA), which is the enhanced version of CMA Light. Differently from CMA Light, F-CMA supports the random reshuffling of the samples, while preserving the same convergence guarantees. The work described in this chapter has been carried out during the first half of 2024 year with the invaluable help of ALCOR Lab, which provided the hardware resources for the massive computational tests on F-CMA and its competitors. The work has been submitted at "IEEE Transactions on Neural Networks and Learning Systems".

6.1 Introduction

In recent years, with the introduction of popular optimizers such as Adam [104], adaptive gradient methods have gained popularity in the deep learning community because of their fast convergence and lower sensitivity to hyper-parameters. Nonetheless, as discussed in Chapter 3, they suffer from drawbacks, including higher memory demands for elements like moving averages and a less investigated convergence theory. In this chapter, we propose F-CMA, a fast-controlled mini-batch algorithm enhanced with backtracking line-search, to solve the unconstrained minimization problem formulated in (2.6).

The exponential increase in the number of model parameters, driven by the advancement in deep learning architectures, such as attention-based and diffusion models [52, 102, 125, 171, 204], combined with the growing adoption of over-parametrization to achieve optimal interpolation [5, 29, 38, 210], yielded the use of traditional first-order methods prohibitively expensive, since they require the full-batch gradient of the objective function. Hence, as we have stated so far, the most largely adopted optimizers are the mini-batch methods, which rely on parameters updating rules that exploit only a small subset of the available dataset. While the underlying idea of mini-batch methods dates back to the early 1950s with [176], plenty of these methods

have been developed in the past decade (see also Chapter 2 and Chapter 3). Moreover, they can be categorized into adaptive and non-adaptive methods based on whether the learning rate is static across all parameters or dynamically adjusted during the training process.
Non-adaptive methods, like Stochastic Gradient Descent (SGD) [26, 176, 180, 198] and its variants such as Incremental Gradient (IG) [16], Incremental Aggregated Gradient (IAG) [81, 200], and random reshuffling (RR) [148] perform parameters update with fixed or decreasing learning rate in the form:

$$w^{t+1} = w^t + \alpha^t d^t \tag{6.1}$$

where d^t is an estimate of $-\nabla f(w^t)$. Despite being simpler to implement with respect to their adaptive competitors, these methods are highly affected by the initial learning rate setting. Furthermore, as highlighted in [212], the optimal fixed learning rate can have strong dependencies on the Lipschitz constant of $f(w)$, while untuned updating rules can often lead to slow convergence or to exploding gradient [11, 113].
Conversely, adaptive methods perform parameter updates using a fixed learning rate and a direction, that is, an aggregated measure (such as the exponential moving average) of the gradient estimates sampled until the current step. Many adaptive methods have been proposed by deep learning researchers [84, 144], each varying the gradient accumulation rule. In this work, seeking for a comparison between F-CMA and the most frequently used optimizers, we consider as benchmark the adaptive methods presented in Chapter 2: Adam [104], Adamax [104], AdamW [134, 135], Adagrad [58], Nadam [57], and Radam [127]. Despite requiring more memory to store the past gradient estimates [191], adaptive methods are generally acknowledged to be more efficient, stable, and robust to the initial learning rate [138, 212, 220]. However, they require stronger assumptions on the objective function to prove convergence, such as strong convexity of $f(w)$ [86, 107, 108] or gradient-related search direction [73, 74], which are often not satisfied in deep learning application [99, 174]. Moreover, most of the available converge theory rely on stochastic results, i.e., convergence in the expected value [26].
In addition to non-adaptive and adaptive methods, there have been multiple attempts to address the problem of learning rate tuning in non-adaptive algorithms such as making SGD adaptive through the use of a line-search procedure [80, 93, 139, 153, 154]. Building on this research path and considering the advantages that the RR strategy has demonstrated in numerous applications compared to its incremental and stochastic counterparts [83, 150, 182, 190, 213], we propose a method that integrates RR [148] (Algorithm 6.1) with a safeguard rule to ensure the sufficient decrease of an approximation, $\tilde{f}$, of $f(w)$. More in detail, if the safeguard rule is not satisfied, inspired by the controlled mini-batch framework proposed in [131], F-CMA automatically adjusts the learning rate by either decreasing it or executing a line-search procedure (Algorithm 6.2). Similarly to what we have seen for CMA Light in the last chapter, the decrease condition on $\tilde{f}$ at each epoch is checked using the aggregated running loss over an epoch. However, F-CMA performs the line-search without the true loss function, and only two full evaluations of $f(w)$ are required, reducing the computational overhead with respect to CMA Light.
Summarizing, the main contributions introduced with F-CMA can be summarized as follows:

- We prove that $\inf_k \|\nabla f(w^k)\| \to 0$, allowing the random reshuffling of the samples at each epoch using a deterministic approach and without requiring the convexity of $f(w)$.
- We introduce a learning rate-driven early stopping rule to reduce the computational effort, lowering the consumption energy required for the training process and its carbon footprint.
- We design a backtracking derivative-free line-search technique that can use any approximation model of $f(w)$ and requires, in the worst case, at most two full evaluations of $f(w)$ while still preserving convergence properties and competitive performances.
- We lighten the individual coercivity assumption of CMA Light on $f_i(w)$ and replace it with gradient clipping to ensure the boundedness of $\|\nabla f(w^k)\|$ and speed up the training process [39, 170].
- We introduce an internal scheduler (η in Algorithm 6.2) to adjust the learning rate ζ^k before the line-search.

The rest of the paper is organized as follows. In section 6.2 we explain in detail the algorithm F-CMA, focusing on the elements of novelties with respect to the others algorithm of the CMA family, and introduce our assumption on the objective function and its gradient. In section 6.3 we prove in a deterministic way that F-CMA globally converges to a stationary point, i.e., the sequence of points produced by F-CMA is limited and at least one of its accumulation points is stationary for the objective function. In section 6.4 we discuss the experimental setup and the hardware used to carry out the computational tests. In section 6.5 we present the results both on the training loss and on the generalization performance attained by the model. Finally, in section 6.6, we briefly report our conclusions.

6.2 Fast-Controlled Mini-batch Algorithm (F-CMA)

In this section, we discuss in detail the proposed method and the convergence properties of F-CMA.

F-CMA, which is reported in Algorithm 6.3, relies on checking after every epoch a sufficient decrease condition on an estimate of the real objective function. This estimate is obtained by summing the batch losses accumulated during an epoch. According to Algorithm 6.1, given a random permutation I^k at iteration k, this estimate be formally written as:

$$\tilde{f}^k = \sum_{p=1}^{P} f_{h_p}(\tilde{w}_{h_{p-1}}^{k-1}) \tag{6.2}$$

where $\{\tilde{w}_{h_{p-1}}^k\}_{p=1,\dots,P}$ is the trajectory of points generated by a RR iteration (Algorithm 6.1). Notice that $\tilde{f}^k$ from (6.2) is the same estimate from (5.2). The notation is different to highlight that in F-CMA we extend the convergence results of CMA Light to the case with random reshuffling.

However, before proceeding to the detailed explanation of F-CMA, we introduce the assumptions we make to prove the global convergence to a stationary point in a deterministic way. These

Algorithm 6.1 `Random Reshuffling iteration`

1: **Input**: w^k, ζ^k
2: Let $I_k = \{h_1, \dots, h_P\}$ be a random permutation of $\{1, \dots, P\}$
3: Set $\widetilde{w}^k_{h_0} = w^k$
4: **for** $i = 1, ..., P$ **do**
5: $\quad \tilde{f}^k_{h_i} = f_{h_i}(\widetilde{w}^k_{h_{i-1}})$
6: $\quad \tilde{d}^k_{h_i} = -\nabla f^k_{h_i}(\widetilde{w}^k_{h_{i-1}})$
7: $\quad \widetilde{w}^k_{h_i} = \widetilde{w}^k_{h_{i-1}} + \zeta^k \tilde{d}^k_{h_i}$
8: **end for**
9: **Output** $\widetilde{w}^k = \widetilde{w}^k_P$, $d^k = \sum_{i=1}^{P} \tilde{d}^k_{h_i}$, $\tilde{f}^k = \sum_{i=1}^{P} \tilde{f}^k_{h_i}$

assumptions are slightly more restrictive than the ones used in the previous chapters for CMA and CMA Light.

Assumption 6.2.1. *The function f is coercive, i.e., all the level sets*

$$\mathcal{L}_f(w_0) = \{w \in \Re^n : \ f(w) \leq f(w_0)\} \tag{6.3}$$

are compact for any $w_0 \in \Re^n$.

Assumption 6.2.2. *$f_p(w) \in \mathcal{C}^1_{L_p}(\mathbb{R}^n)$ for all $p = 1, \dots, P$, meaning that the functions f_p are all continuously differentiable and the gradients ∇f_p are Lipschitz continuous for each $p = 1, \dots, P$, namely there exists finite $L = \max_{p=1,\dots,P} L_p > 0$ such that:*

$$\|\nabla f_p(u) - \nabla f_p(v)\| \leq L\|u - v\| \forall \ u, w \in \mathbb{R}^n \tag{6.4}$$

Notice that, to simplify the notation of the more frequently recurring terms, we will further refer to the Lipschitz constant of $f_p(w)$ as L_{f_p}, while we will indicate the constant of ∇f_p as L_p.

Assumption 6.2.3. *The functions f_p are bounded below for each $p = 1, \dots, P$, that is:*

$$\mathcal{L}_{f_p}(w_0) = \{w \in \Re^n : \ f_p(w) \leq f_p(w_0)\} \text{ is compact for any } w_0 \in \mathbb{R}^n. \tag{6.5}$$

Assumption 6.2.4. *There exist three constants $C, D > 0$ and $M > 0$ such that for each $p = 1, \dots, P$ and for any $w \in \mathbb{R}^n$:*

$$\|\nabla f_p(w)\| \leq C\|\nabla f(w)\| + D \tag{6.6}$$

$$\|\nabla f(w)\| \leq M \tag{6.7}$$

Assumptions 6.2.1 and 6.2.2 are the same required to prove the convergence of CMA and CMA Light, while assumption 6.2.3 is a slight modification of assumption 5.2.3. Assumption 6.2.4, which postulates the boundedness of the norm of the gradient, is more restrictive and was not required in the previous chapters. However, bounded gradient hypothesis is still widely adopted in algorithms that use approximations of the true objective function, such as optimization in federated learning with error [55, 65, 117, 175], adaptive methods in non-convex optimization [50, 208], and stochastic gradient method (e.g., [4, 16] assume bounded variance of $\|\nabla f(w)\|$).

We refer the reader to Chapter 3 for a more detailed discussion of this topic. Furthermore, we remark that the convergence of F-CMA still holds if Assumption 6.2.4 is satisfied for any w produced by F-CMA. In the computational practice this condition can be easily ensured using the gradient clipping option[1], which is also acknowledged to speed up the training process when using non-adaptive methods [39, 170].

Like CMA Light, discussed in the previous chapter, F-CMA (reported in Algorithm 6.3) follows the same structure shared by all algorithms adopting the CMA framework. F-CMA is based on the random reshuffling iteration (Algorithm 6.1). The assumptions stated above, jointly with the conditions at steps 6, 9, and 13 of F-CMA, make it possible to prove the global convergence to a stationary point for smooth objective functions. At each epoch k, after performing an RR cycle and having a tentative point $\tilde{w}^k$, a tentative direction d^k, and an estimate of the objective function in the point $\tilde{w}^k$, $\tilde{f}^{k+1}$ given by (6.2), F-CMA checks at step 6, whether there has been an improvement by at least a factor $\gamma\zeta^k$, being ζ^k the decreasing learning rate. Following the same logic of CMA and CMA Light, if the condition at step 6 is satisfied, which often occurs in the first epochs of training, F-CMA produces the same points that would have been produced by RR. When the condition is no longer satisfied, a safeguard rule on the norm of the direction is checked. If the norm is too small with respect to the hyper-parameter τ (condition at step 9), the learning rate ζ^k is decreased by the factor θ but, if still in the level set of $f(w)$, the tentative point is accepted. Otherwise, the Derivative-Free Linesearch (DFL), reported in Algorithm 6.2 is executed as in CMA Light. However the DFL is performed using any suitable model $\psi(\cdot)$ of the true objective $f(w)$ and the real $f(w)$ is evaluated only to check the initial and final conditions (step 3 and 13 of Algorithm 6.2). The line-search returns an Armijo-like step-size [6, 77, 78] and, according to condition at step 13 of F-CMA, the current step-size α^k can be updated as follows: decreased, set to its returned value, or driven down to zero for the current epoch in case we end up out of the level set (conditions at steps 14.3 or 16.2).

Algorithm 6.2 `Derivative-Free line-search (DFL)`

1: Input $(\tilde{f}^k, w^k, d^k, \zeta^k; \gamma, \delta, \psi(\cdot))$: $w^k \in \mathbb{R}^n, d^k \in \mathbb{R}^n, \zeta^k > 0$, $\gamma \in (0,1), \delta \in (0,1), \eta \in (0,1)$
2: Set $j = 0, \alpha = \zeta^k \eta, f_j = \tilde{f}^k$
3: Compute $f(w^k)$
4: **if** $\tilde{f}^k > f(w^k) - \gamma\alpha\|d^k\|^2$ **then**
5: Set $\alpha^k = 0$ and **return** α^k, f_j
6: **end if**
7: **while** $\psi(w^k + (\alpha/\delta)d^k) \leq \min\{f(w^k) - \gamma\alpha\|d^k\|^2, f_j\}$ **do**
8: Set $f_{j+1} = \psi(w^k + (\alpha/\delta)d^k)$
9: Set $j = j + 1$
10: Set $\alpha = \alpha/\delta$
11: **end while**
12: Compute $f(w^k + \alpha d^k)$
13: **if** $f(w^k + \alpha d^k) \leq f(w^k) - \gamma\alpha\|d^k\|^2$ **then**
14: Set $\alpha^k = \alpha$ and **return** $\alpha^k, f(w^k + \alpha^k d^k)$
15: **else**
16: Set $\alpha^k = 0$ and **return** α^k, f_0
17: **end if**

Although F-CMA shares a similar structure with CMA Light, it differs not only in that it generalizes CMA Light, supporting random reshuffling epoch. In fact, a key innovation in F-

[1] https://pytorch.org/docs/stable/generated/torch.nn.utils.clip_grad_norm_.html

CMA is its derivative-free line search, which introduces an internal scheduler (parameter η in Algorithm 6.2) and utilizes an approximation model ψ for the objective function. Firstly, at the beginning of DFL (step 2), the current learning rate ζ^k is scaled by a factor $\eta \in (0, 1)$ to achieve more stability than in CMA and CMA Light, where computational experience shows that the line-search happens to fail when ζ^k is still too large to move successfully alongside the search direction. Then, DFL performs an extrapolation procedure by dividing the current learning rate by the hyper-parameter $\delta \in (0, 1)$ until the condition at step 7 holds. However, in order to minimize the number of evaluations of the entire objective $f(w)$, we tailored the line-search to support the use of any coercive model ψ to approximate f. In our implementation, given a permutation $I_k = \{h_1, \dots, h_P\}$, ψ is computed using a subset of the available samples, such that:

$$\psi(w) = \sum_{i=1}^{p} f_{h_i}(w) \text{ with } p << P \tag{6.8}$$

More formally, we highlight that the algorithm does not change substantially if using any approximating model ψ which satisfies the following:

Assumption 6.2.5 (Properties of ψ)**.** *The approximating model is a function $\psi : \mathbb{R}^n \to \mathbb{R}$ with the following properties:*

i) ψ is coercive;

ii) there exists a non negative constant $M < \infty$ such that $|\psi(w) - f(w)| \leq M$ for all $w \in \mathbb{R}^n$.

Eventually, the sufficient decrease condition at step 13 is still evaluated on the true objective to *deterministically* prove the global convergence without assuming further hypotheses on the search direction d^k.

6.3 Convergence properties

Proving the global convergence of F-CMA to a stationary point for $f(w)$ is straightforward if Proposition 5.2.4 in the previous chapter can be applied to the random reshuffling case. In fact, the other proofs are mostly by contradiction, and, as seen in the previous chapters, they are all based on the analysis of the possible outcomes of each epoch k of F-CMA.
For the sake of completeness, we reformulate Proposition 5.2.4 using a notation that highlights the random reshuffling of the samples at each epoch. Using the assumptions stated in the previous section, we prove the following result.

Proposition 6.3.1. *Assume that the sequence of points produced by Algorithm 6.1 $\{w_k\}$ is limited and that $\lim_{k\to 0} \zeta_k = 0$. Then, for any limit point $\bar{w}$ of $\{w^k\}$ a subset of indices K exists*

Algorithm 6.3 `F-CMA (Fast-Controlled Minibatch Algorithm)`

Set $\zeta^0 > 0, \theta \in (0,1), \tau > 0, \gamma \in (0,1), \delta \in (0,1), \eta \in (0,1), \alpha_{min} > 0$
Let $w^0 \in \mathbb{R}^n$
Compute $f(w^0)$ and set $\tilde{f}^0 = f(w^0)$ and $\phi^0 = \tilde{f}^0$
for $k = 0, 1 \dots$ **do**
  Compute $(\widetilde{w}^k, d^k, \tilde{f}^{k+1}) =$ `RandomReshuffling`(w^k, ζ^k)
  if $\tilde{f}^{k+1} \leq \min\{\phi^k - \gamma\zeta^k, f(\omega^0)\}$ **then**
    Set $\zeta^{k+1} = \zeta^k$ and $\alpha^k = \zeta^k$ and $\phi^{k+1} = \tilde{f}^{k+1}$
  else
    if $\|d^k\| \leq \tau\zeta^k$ **then**
      Set $\zeta^{k+1} = \theta\zeta^k$ and $\alpha^k = \begin{cases} \zeta^k & \text{if } \tilde{f}^{k+1} \leq f(w^0) \\ 0 & \text{otherwise} \end{cases}$ and $\phi^{k+1} = \phi^k$
    else
      $(\widetilde{\alpha}^k, \hat{f}^{k+1}) =$ `DFL`$(w^k, d^k, \zeta^k; \gamma, \delta, \eta)$
      if $\widetilde{\alpha}^k \|d^k\|^2 \leq \tau\zeta^k$ **then**
        Set $\zeta^{k+1} = \theta\zeta^k$ and $\alpha^k = \begin{cases} \widetilde{\alpha}^k & \text{if } \widetilde{\alpha}^k > 0 \text{ and } \hat{f}^{k+1} \leq f(w^0) \\ \zeta^k & \text{if } \widetilde{\alpha}^k = 0 \text{ and } \tilde{f}^{k+1} \leq f(w^0) \\ 0 & \text{otherwise} \end{cases}$
      else
        Set $\zeta^{k+1} = \max\{\widetilde{\alpha}^k, \alpha_{min}\}$, and $\alpha^k = \begin{cases} \widetilde{\alpha}^k & \text{if } \widetilde{\alpha}^k > 0 \text{ and } \hat{f}^{k+1} \leq f(w^0) \\ 0 & \text{otherwise} \end{cases}$
      end if
      Set $\phi^{k+1} = \min\{\hat{f}^{k+1}, \tilde{f}^{k+1}, \phi^k\}$
    end if
  end if
  Set $w^{k+1} = w^k + \alpha^k d^k$
end for

such that

$$\lim_{k\to\infty, k\in K} w^k = \bar{w}, \tag{6.9}$$

$$\lim_{k\to\infty, k\in K} \zeta^k = 0, \tag{6.10}$$

$$\lim_{k\to\infty, k\in K} \widetilde{w}_i^k = \bar{w}, \quad \textit{for all } i = 1, \dots, P \tag{6.11}$$

$$\lim_{k\to\infty, k\in K} d_k = -\nabla f(\bar{w}). \tag{6.12}$$

$$\lim_{k\to\infty, k\in K} \tilde{f}^k = f(\bar{w}). \tag{6.13}$$

Proof. The first part of this proof is substantially the generalization of Proposition 4.3.4 using a notation that highlights the dependency of the on the sequence from the current random permutation.

Recalling that $\{w^k\}$ is limited and that $\lim_{k\to 0} \zeta_k = 0$, (6.9) and (6.10) follow by definition.

Then, given a permutation $I^k = \{h_1, h_2, \dots, h_P\}$, consider the sequence $\|\widetilde{w}_{h_i}^k - w^k\|$. Recalling that $\widetilde{w}_0^k = w^k$ follows by definition of 6.1 that:

$$\widetilde{w}_{h_1}^k = w^k - \zeta^k \nabla f_{h_1}(w^k)$$

which implies that:

$$\|\widetilde{w}_{h_1}^k - w^k\| \leq \zeta^k \|\nabla f_{h_1}(w^k)\| \tag{6.14}$$

For $i > 1$ we can write:

$$\|\widetilde{w}^k_{h_i} - w^k\| = \zeta^k \|\sum_{j=1}^{i} \nabla f_{h_j}(\widetilde{w}^k_{h_{j-1}})\| \leq \zeta^k \sum_{j=1}^{i} \|\nabla f_{h_j}(\widetilde{w}^k_{h_{j-1}}) + \nabla f_{h_j}(w^k) - \nabla f_{h_j}(w^k)\|$$

Exploiting the triangular inequality, (6.2.2), and (6.2.4), we have that:

$$\zeta^k \sum_{j=1}^{i} \|\nabla f_{h_j}(\widetilde{w}^k_{h_{j-1}}) + \nabla f_{h_j}(w^k) - \nabla f_{h_j}(w^k)\| \leq \zeta^k \sum_{j=1}^{i} L\|\widetilde{w}^k_{h_{j-1}} - w^k\| + \|\nabla f_{h_j}(w^k)\|$$

This proves that:

$$\|\widetilde{w}^k_{h_i} - w^k\| \leq \zeta^k \sum_{j=1}^{i} L\|\widetilde{w}^k_{h_{j-1}} - w^k\| + \|\nabla f_{h_j}(w^k)\| \text{ for all } i = 1, \ldots, P \tag{6.15}$$

Notice that the result in (6.15) is simply the generalization of the thesis of Lemma 4.3.2 with an explicit dependency from I^k.
Hence, we can now reason by induction. If $i = 1$, then, taking the limit for $k \to \infty$ on (6.14) and recalling that $\zeta^k \to 0$, we get:

$$\lim_{k\to\infty, k\in K} \|\widetilde{w}^k_{h_1} - w^k\| = 0$$

Assuming that this is true for $i = 1, 2, \ldots, j-1$, thanks to the boundedness of $\{w^k\}$, (6.11) is proved.
Let's now consider the direction given by 6.1:

$$d^k = -\sum_{i=1}^{P} \nabla f_{h_i}(\widetilde{w}^k_{h_{i-1}})$$

Taking the limit for $k \to \infty$, using (6.11) and assumption 6.2.2, (6.12) is proved, since we have:

$$\lim_{k\to\infty, k\in K} d^k = -\sum_{i=1}^{P} \nabla f_{h_i}(\bar{w}) = -\nabla f(\bar{w})$$

Eventually, using the continuity of $f(w)$ and (6.11), we have:

$$\lim_{k\to\infty, k\in K} \tilde{f}^k = \lim_{k\to\infty, k\in K} \sum_{i=1}^{P} f_{h_i}(\widetilde{w}^k_{h_{i-1}}) = \sum_{i=1}^{P} f_{h_i}(\bar{w}) = f(\bar{w})$$

which concludes the proof of (6.13) □

Having proved that F-CMA can converge to a stationary point if $\{w^k\}$ is bounded and $\zeta^k \to 0$, we need to prove that these conditions actually holds. We begin by proving, in the following proposition, an important property of the DFL (Algorithm 6.2).

Proposition 6.3.2. *Suppose that $\psi(\cdot)$ satisfies assumption 6.2.5. Then, Algorithm 6.2 is well-defined, and it returns within a finite number of steps a scalar α_k such that:*

$$f(w^k + \alpha^k d^k) \leq f(w^k) - \gamma \alpha^k \|d^k\|^2 \tag{6.16}$$

Proof. First, we show by contradiction that the EDFL cannot cycle an infinitely number of times. If this was the case, we would have that, for all $j = 1, 2, \ldots$:

$$\psi(w^k + \frac{\zeta^k \eta}{\delta^j} d^k) \leq f(w^k) - \gamma \frac{\zeta^k \eta}{\delta^j} \|d^k\|^2$$

Taking the limit for $j \to \infty$ and being $\frac{\zeta^k \eta}{\delta^j} \to \infty$, we would get a contradiction with the compactness of the level sets of ψ and with the boundedness of $f(w)$ in 6.2.1.
To prove (6.16), we first consider that it trivially holds if the EDFL fails (step 4) and $\alpha^k = 0$. Conversely, if the condition at step 7 is verified at least one time, we can select a scalar $\alpha > 0$, only if (6.16) holds (step 13). □

Now, since we removed the individual coercivity assumption on f_i, we need to prove that the difference between the approximation of the objective function $\tilde{f}^{k+1}$ and the real value $f(w^k)$ at the beginning of the epoch is bounded by a certain fixed threshold. In the following lemma, we show that this threshold exists and depends on the constants C and M, on the number of samples (or batch of samples) P, as well as on the Lipschitz constant of $f(w)$.

Lemma 6.3.3. *Let $M > 0$ be a scalar such that $\|\nabla f(w)\| \leq M$ for all $w \in \mathbb{R}^n$ (assumption 6.2.4). Then, there exists a constant $\tilde{C} >=$ for which the following relation between the estimate of the objective function $\tilde{f}^k$ and the real objective function $f(w^k)$ holds for every iteration $k = 0, 1, \ldots$ of Algorithm 6.3.*

$$|\tilde{f}^{k+1} - f(w^k)| \leq P^2 L_f \zeta^k \tilde{C} M \tag{6.17}$$

where L_f is the Lipschitz-constant of $f(w)$.

Proof. To avoid an abuse of notation, the proof is written in the case of the trivial random permutation $I^k = \{1, 2, \ldots, P\}$.
Let's consider the difference between the estimate and the real value of the p-th term of $f(w)$. Recalling assumption 6.2.2 and naming $L_f = \max_{p=1,\ldots,P} L_{f_p}$ we can write:

$$|f_p(\tilde{w}^k_{p-1}) - f_p(w^k)| \leq L_f \|\tilde{w}^k_{p-1} - w^k\| = L_f \zeta^k \|\sum_{j=1}^{p} \nabla f_j(\tilde{w}^k_{j-1})\| \leq$$

$$L_f \zeta^k p(C\|\nabla f(\tilde{w}^k_{j-1})\| + D) \leq L_f \zeta^k p(CM + A)$$

where for the last inequalities we exploited Assumption 6.2.4 and the hypothesis on $\|\nabla f(w)\|$. Now we can rewrite:

$$|\tilde{f}^{k+1} - f(w^k)| = |\sum_{p=1}^{P} f_p(\tilde{w}^k_{p-1}) - f_p(w^k)| \leq \sum_{p=1}^{P} |f_p(\tilde{w}^k_{p-1}) - f_p(w^k)|$$

$$\leq \sum_{p=1}^{P} L_f \zeta^k p(CM + A) = L_f \zeta^k (CM + A) \frac{P(P-1)}{2} \leq L_f \zeta^k \tilde{C} P^2$$

with $\tilde{C} = (CM + A)$ which proves the thesis. □

We have now collected all the elements to prove, in the following proposition, that the sequence of points produced by F-CMA is limited.

Proposition 6.3.4. *Let $\{w^k\}$ be the sequence of points produced by the Algorithm 6.3 and let $M > 0$ be a scalar such that $\|\nabla f(w)\| \leq M$ for all $w \in \mathbb{R}^n$. Then, $\{w^k\}$ is limited.*

Proof. The proof begins showing that for each $k = 0, 1 \dots$ at least one of the following conditions holds:

i) $\tilde{f}^{k+1} \leq f(w^0)$

ii) $f(w^k) \leq f(w^0)$

iii) $w^{k+1} = w^k$

By definition of Algorithm 6.3, for every iteration k, we have that one of the following cases happens:

a) step 7 is executed, i.e., we move to a new point such that $\tilde{f}^k \leq \min\{\phi^{k-1} - \gamma\zeta^k, f(\omega^0)\}$.

b) step 10 is executed. We move to a point such that $\tilde{f}^k \leq f(\omega^0)$, or we set $\alpha^k = 0$, i.e., we do not move from ω^k.

c) step 14 is executed. After the line-search, we move to a point such that either $\tilde{f}^k \leq f(\omega^0)$ or $f(\omega^k + \tilde{\alpha}^k d^k) = f(\omega^{k+1}) \leq f(\omega^0)$ or we do not move if neither of the conditions is satisfied.

d) step 16 is executed. After the line-search we move to a point such that $f(\omega^k + \tilde{\alpha}^k d^k) = f(\omega^{k+1}) \leq f(\omega^0)$ or we do not move.

In case a) at least i) holds. In case b) either i) or iii) holds. In case c) either i) or ii) holds if the line-search learning rate is satisfied otherwise iii) holds. In case d), again, ii) holds if the line-search learning rate is satisfied, otherwise iii) holds.
Now, suppose by contradiction that the sequence of points $\{w^k\}$ produced by the Algorithm 6.3 diverges, i.e., an infinite set of index $k \subseteq \{0, 1, 2, \dots\}$ exists such that:

$$\lim_{\substack{k \to \infty \\ k \in K}} ||\omega^k|| = \infty \tag{6.18}$$

Consider the following sets of indexes:

- $K_1 = \{k = 0, 1, \cdots : \tilde{f}^{k+1} \leq f(w^0)\}$
- $K_2 = \{k = 0, 1, \cdots : \tilde{f}(w^k) \leq f(w^0)\}$
- $K_3 = \{k = 0, 1, \cdots : w^{k+1} = w^k\}$

We first show by contradiction that $K \cap (K_1 \cup K_2)$ must be finite.
For any k such that i) holds, recalling (6.17), we have that:

$$f(w^k) \leq \tilde{f}^{k+1} + P^2 L_f \zeta^k \tilde{C} M \leq f(w^0) + P^2 L_f \zeta^k \tilde{C} M$$

If $K \cap (K_1 \cup K_2)$ is not finite, then taking the limit for $k \to \infty$ with $k \in K \cap (K_1 \cup K_2)$ on both sizes and recalling that $f(w)$ is coercive (assumption (6.2.1)) and that $\{\zeta^k\}$ is by definition a non-increasing sequence, we get the following contradiction:

$$\infty \leq f(w^0) + P^2 L_f \bar{\zeta} \tilde{C} M < \infty$$

Conversely, for any k such that ii) holds, if $K \cap (K_1 \cup K_2)$ is not finite, we immediately have a contradiction with assumption (6.2.1), since we would get that $\|w^k\| \to \infty$ but $f(w^k) \leq f(w^0)$, meaning that the level set is not compact.
Hence, since $K \cap (K_1 \cup K_2)$ must be finite, there must be an infinite set of index $\tilde{K} \subseteq K_3 \backslash (K_1 \cup K_2)$ such that:

$$\lim_{\substack{k \to \infty \\ k \in (\tilde{K} \cap K)}} ||\omega^k|| = \infty \tag{6.19}$$

But here we encounter again a contradiction. In fact, let's consider an index $\bar{k} \in (\tilde{K} \cap K)$. If for all $m = \{1, 2, \dots\}$ $\bar{k} + m \in (\tilde{K} \cap K)$, then we get a trivial contradiction with $\|w^k\| \to \infty$. Hence, to ensure (6.18), there must be an index $\tilde{k} > \bar{k}$ such that $w^{\tilde{k}} \neq w^{\bar{k}}$, which means such that iii) does not hold. However, we have proved that if iii) does not hold, at least one between i) and ii) must hold, which means $\tilde{k} \in (K_1 \cup K_2)$. But this is, by definition of $\tilde{K}$, in contradiction with $k \in (\tilde{K} \cap K)$. □

Finally, the following proposition proves that the step-size ζ^k goes to zero, which is the last element we need to prove the convergence of F-CMA. Although the proof is structurally identical to the proof of Proposition 5.3.4, we still report it below because the steps of algorithms F-CMA and CMA Light differ in numeration.

Proposition 6.3.5. *Let $\{\zeta^k\}$ be the sequence of steps produced by Algorithm 6.3, then*

$$\lim_{k \to \infty} \zeta^k = 0.$$

Proof. In every iteration, either $\zeta^{k+1} = \zeta^k$ or $\zeta^{k+1} = \theta \zeta^k < \zeta^k$. Therefore, the sequence $\{\zeta^k\}$ is monotonically non-increasing. Hence, it results

$$\lim_{k \to \infty} \zeta^k = \bar{\zeta} \geq 0.$$

Let us suppose, by contradiction, that $\bar{\zeta} > 0$. If this were the case, there should be an iteration index $\bar{k} \geq 0$ such that, for all $k \geq \bar{k}$, $\zeta^{k+1} = \zeta^k = \bar{\zeta} \geq \alpha_{min}$. Namely, for all iterations $k \geq \bar{k}$, steps 7 or 17 are always executed.

Let us now denote by

$$\begin{aligned} K' &= \{k : \ k \geq \bar{k}, \text{ and step 7 is executed}\} \\ K'' &= \{k : \ k \geq \bar{k}, \text{ and step 17 is executed}\} \end{aligned}$$

Let us first prove that K' cannot be infinite. If this was the case, we would have that infinitely many times

$$\gamma\bar{\zeta} \leq \phi^k - \tilde{f}^{k+1} = \phi^k - \phi^{k+1}$$

However, by the instructions of the algorithm $\{\phi^k\}$ is a monotonically non increasing sequence. Also, by assumption 6.2.3, for all k, $\phi^k \geq 0$. This means that:

$$\lim_{k\to\infty, k\in K'} \phi_k = \lim_{k\to\infty, k\in K'} \phi^{k+1} = \bar{\phi}.$$

That is to say that

$$\lim_{k\to\infty, k\in K'} \phi^k - \phi^{k+1} = 0$$

which is in contrast with $\bar{\zeta} > 0$.

Thus, an index $\hat{k}$ exists such that, $k \in K''$ for $k \geq \hat{k}$. Thanks to Proposition 6.3.2, for $k \geq \hat{k}$, we have

$$f(w^{k+1}) \leq f(w^k + \alpha^k d^k) \leq f(w^k) - \gamma\alpha^k \|d^k\|^2, \tag{6.20}$$

that is the sequence $\{f(w^k)\}$ is definitely monotonically non-increasing and bounded below since $f(w) \geq 0$. Hence

$$\lim_{k\to\infty} f(w^k) = \lim_{k\to\infty} f(w^{k+1}) = \bar{f}.$$

Then, taking the limit, we obtain:

$$\lim_{k\to\infty} \alpha^k \|d^k\|^2 = 0.$$

But then, for $k \geq \hat{k}$ sufficiently large, it would happen that

$$\alpha^k \|d^k\|^2 \leq \tau\bar{\zeta}$$

which means that step 10 would be executed, setting $\zeta^{k+1} = \theta\bar{\zeta}$ thus decreasing ζ^{k+1} below $\bar{\zeta}$, which contradicts our initial assumption and concludes the proof. □

Following the last proof, it is now possible to show that F-CMA converges to a stationary point, i.e., that $\lim_{k\to\infty} \inf_k \|\nabla f(w^k)\| = 0$.

Theorem 6.3.6. *Let $\{w^k\}$ be the sequence of points produced by the Algorithm 6.3. Then, $\{w^k\}$ admits limit points, and at least one of them is a stationary point for $f(w)$.*

Proof. $\{w^k\}$ is limited by Proposition 6.3.4, thus it admits limit points. Furthermore, recalling

Proposition 6.3.5, the set of index:

$$K = \{k = 0, 1, \cdots : \zeta^{k+1} = \theta\zeta^k\}$$

must be infinite and, by Proposition 6.3.1, there exists an infinite set $\bar{K} \subseteq K$ such that

$$\lim_{k\to\infty, k\in\bar{K}} w^k = \bar{w}$$

and

$$\lim_{k\to\infty, k\in\bar{K}} d^k = -\nabla f(\bar{w})$$

Now, recalling conditions at Step 9 and at Step 13, we have that for all $k \in K$ either $\|d^k\|^2 \leq \tau\zeta^k$, or $\tilde{\alpha}^k\|d^k\| \leq \tau\zeta^k$. Hence, we can partition $\bar{K}$ as follows:

$$\begin{aligned} \bar{K}' &= \{k : \text{ condition at step 9 is satisfied}\} \\ \bar{K}'' &= \{k : \text{ condition at step 13 is satisfied}\} \end{aligned}$$

At least one between $\bar{K}'$ and $\bar{K}''$ must be infinite.
If K' is infinite, then:

$$\lim_{k\to\infty, k\in\bar{K}'} d^k = -\nabla f(\bar{w}) \leq \lim_{k\to\infty, k\in\bar{K}'} \tau\zeta^k = 0$$

and the thesis is proved.
If $\bar{K}''$ is infinite, then:

$$\lim_{k\to\infty, k\in\bar{K}'} d^k\tilde{\alpha}^k \leq \lim_{k\to\infty, k\in\bar{K}'} \tau\zeta^k = 0$$

Recalling that $\tilde{\alpha}^k \geq 0$ and using Assumption 6.2.4, this means:

$$\lim_{k\to\infty, k\in\bar{K}'} d^k\tilde{\alpha}^k \leq \lim_{k\to\infty, k\in\bar{K}'} PC\nabla f(w^k)\tilde{\alpha}^k = 0$$

If $\|\nabla f(w^k)\| \to 0$, the thesis is proved. Otherwise, it must hold that $\tilde{\alpha}^k \to 0$.
But in this last case, considering Proposition 6.3.2, we have that, for all $k \in \bar{K}''$,

$$f(w^k + \tilde{\alpha}^k d^k) \leq f(w^k) - \gamma\tilde{\alpha}^k\|d^k\|^2$$

Applying the Mean Value theorem for all $k \in \bar{K}''$, we know that there exists a scalar $\xi^k \in (0, \tilde{\alpha}^k)$ such that:

$$\frac{f(w^k + \xi^k d^k) - f(w^k)}{\xi^k} = \nabla f(w^k)^T d^k \leq \gamma\|d^k\|^2$$

Taking the limit for $k \to \infty, k \in K''$, we obtain:

$$\|\nabla f(\bar{w})\|^2 \leq \gamma\|\nabla f(\bar{w})\|^2$$

Recalling that $\gamma \in (0, 1)$, this is possible only if $\|\nabla f(\bar{w})\| = 0$, which proves the proposition. □

6.4 Experimental setup

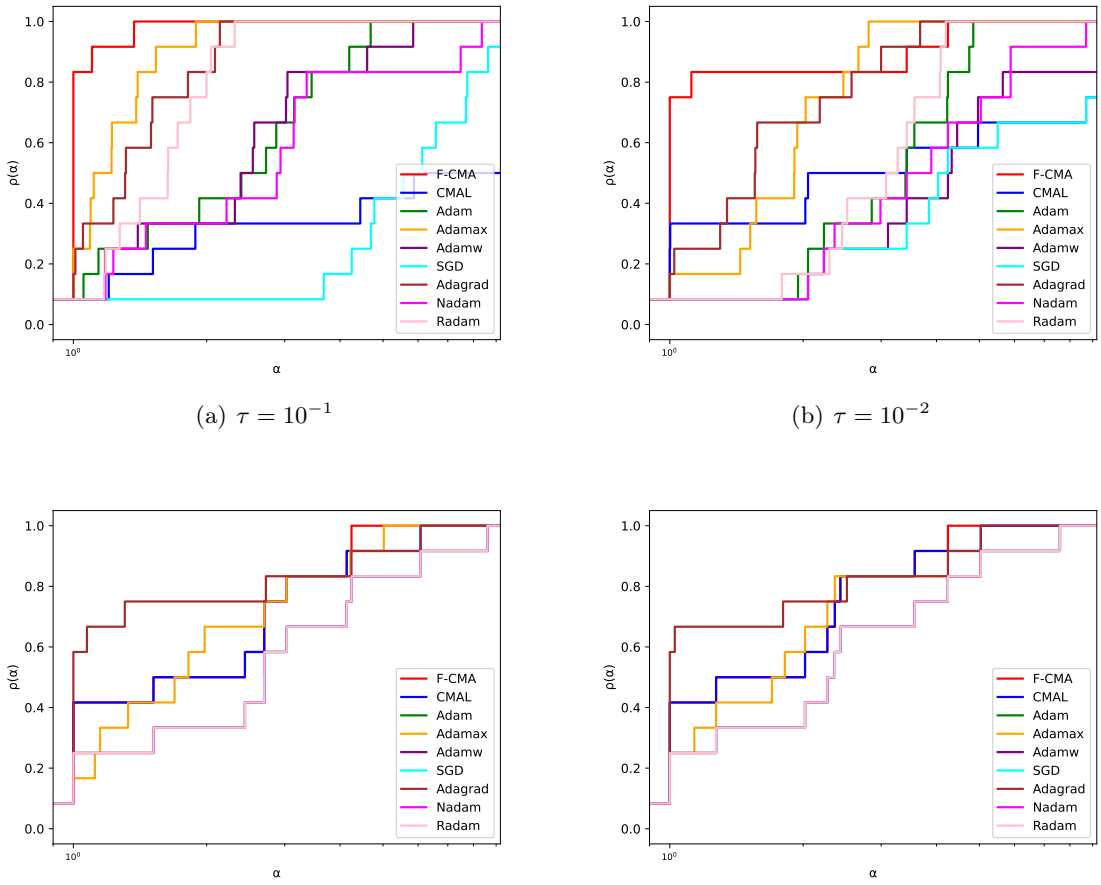
This section gives a detailed description of the experimental setup, including training hyper-parameters, benchmark datasets, and evaluation metrics.
F-CMA has been implemented using PyTorch deep learning API. In order to investigate the convergence behavior and estimation performances achieved by F-CMA, we select again the two reference benchmark datasets, i.e., CIFAR10 and CIFAR100 [109], and compare over eight optimizers and six well-known neural networks. As said in the previous chapters, each dataset is composed of $60K$ images, using the common train/test split composed of $50K/10K$ images, at a resolution of 32×32 pixels with 10 and 100 classes, respectively. More in detail, we select as reference optimizers the original CMA Light configuration (see table 5.1), Adam, Adamax, AdamW, SGD, Adagrad, Nadam, and Radam using their default hyperparameters, i.e., the ones reported in their original papers with the exception of the learning rate of SGD that has been reduced to 10^{-2} due to the lack of convergence in its original settings. Moreover, we compare the optimizers performances over five convolutional and a single vision transformer neural networks, precisely, we chose as reference models ResNet-18, ResNet-50, ResNet-152 [88], Wide ResNet-50 [216], MobileNet V2 [183] and Swin-B [129]. The target problem is always the unconstrained minimization (see (2.6)) of the cross-entropy loss function, as in (2.20). All the models have been initialized on the ImageNet [51] dataset and sequentially trained with each optimizer, i.e., in 54 different combinations, with a batch size of 128 images for 250 epochs (N_{ep}). Due to the large number of optimizers to be compared and the presence of the ultra-deep transformer architecture Swin-B, the tests have been carried out on a high-performing GPU NVIDIA GeForce RTX 3090 24 GB in collaboration with ALCOR Lab[2] at the Department of Computer, Control and Management Engineering (DIAG) in Rome, Italy.

6.5 Computational results

Once the training phase is concluded, we are interest both in the optimization performance and in the generalization capability of the model. Following the same methodological approach of the previous chapters, we evaluate each model-optimizer combination with performance profiles [54, 152] on the cross-entropy training loss and with comparative tables on test accuracy values to determine the architectures performance on the classification tasks.
We firstly report, in Figure 6.1, the performance profiles for different levels of precision. For low and medium level of precision, in figures 6.1(a) and 6.1(b), we observe that F-CMA strongly outperforms all the competitors in terms of efficiency. F-CMA, thanks to the random reshuffling and the reduced computational burden of the derivative-free linesearch, seems to achieve here what CMA Light only partially achieved in 5.4. In particular, we invite the reader to recall figure 5.4(a), where we show that, for low precision level, SGD outperforms CMA Light. Conversely, F-CMA outperforms both SGD and adaptive methods for $\tau = 10^{-1}$. According to our interpretation, this result has an obvious explanation. On the one hand, F-CMA incorporates the efficiency of SGD (it does not need for any stored information such as averaged gradients

[2] https://alcorlab.diag.uniroma1.it/

in adaptive methods) and almost the same theoretical guarantees of CMA Light. On the other hand, the new linesearch makes F-CMA much more efficient, because it only twice needs the real value of $f(w)$ and also because, starting with a smaller step-size $\eta\zeta^k < \zeta^k$, it is easier to attain the sufficient decrease condition. Concerning high-precision levels, we observe, in figures 6.1(c) and 6.1(d), that F-CMA is no more the best-performing optimizer. Nonetheless, for $\tau = 10^{-3}$ and $\tau = 10^{-4}$, all the algorithms tend to have more similar performance than with lower precision levels and F-CMA is clearly not under the average. Furthermore, it can be noticed that F-CMA still outperforms CMA Light, as well as some adaptive optimizers.



(a) $\tau = 10^{-1}$ (b) $\tau = 10^{-2}$

(c) $\tau = 10^{-3}$ (d) $\tau = 10^{-4}$

Figure 6.1: Performance profiles on the training loss on all the architectures for different values of τ - F-CMA vs CMA Light, SGD and adaptive methods

Now, focusing on the test accuracy results, we report in table 6.1 the test accuracy achieved, the number of epochs (N_{ep}) needed to perform a complete train, the average time needed to perform a single epoch (T/Ep) measured in seconds, and the epoch in which the highest value of the test accuracy has been reached (labeled as k^*). Based on the reported results, it can be observed that in most cases, F-CMA achieves the highest accuracy, with a maximum boost of up to +5% compared to other optimizers and a total training time of up to −68% shorter. More in detail, we note that in terms of the time needed to perform a single epoch, SGD is the fastest optimizer, due to its limited amount of operations. However, compared to F-CMA, SGD

achieves an average boost of only 1 second per epoch, and its convergence is usually slower, requiring an average of +145 additional epochs to reach k^*.
Furthermore, by observing the Acc and k^* columns in the table 6.1, it can be noticed that F-CMA reaches higher accuracy values faster than all other optimizers in $\approx 85\%$ of cases; consequently, the addition of the learning rate-driven early stopping rule significantly reduces the total training time. More in detail, taking the Swin-B architecture as an example, it can be noticed that F-CMA achieves accuracy values comparable to CMA Light, i.e., -0.1% and $+0.9\%$ respectively on CIFAR10 and CIFAR100 datasets with faster training time, i.e., a T/Ep equal to -4.0 and -3.9 seconds and a k^* achieved in -30 and -78 epochs, leading to a reduction of -36.5% and -55.8% in the training time needed to converge (k^*). Moreover, in this example, it can also be noticed that Nadam, with its default hyperparameters, is not able to converge (k^* = -). It is also remarkable that in the cases in which the optimizers reach a k^* value in a range lower than 40/50 epochs, this phenomenon usually leads to low estimation accuracy values; for example, this behavior can be noticed in the following optimizer/architecture combinations: CMA Light with ResNet-152 and Nadam with ResNet-18 on the CIFAR100 dataset.
Eventually, we report, in the additional material appendix (figures A.13, A.14, A.15, A.16, A.17, and A.18), the plots of the training loss and test accuracy against computational time for each dataset-network pair. The figures confirm what already concluded by analyzing table 6.1. Trying to better assess the performance of F-CMA against the most commonly used optimizers in deep learning, we also report the one-by-one comparisons of F-CMA against Adam (figure A.12) and Adagrad (figure A.12). F-CMA is not only the best-performing algorithm in terms of test accuracy achieved but it often achieves this result significantly faster than other algorithms. Finally, looking at the plots, in particular at the training on the CIFAR100 dataset, which is an harder and larger task, we observe how F-CMA presents a much more stable behaviour than its adaptive competitors (see, e.g., figure A.17).

6.6 Conclusions

Looking at the results we have presented and discussed, we can assess that in the analyzed classification tasks, F-CMA exhibits a faster convergence behavior than well-established algorithms while attaining higher accuracy values when compared to the same deep learning model. This is evident both analyzing the performance profiles on the training loss and the test accuracy achieved. Jointly with a highly stable behaviour, these performances also represent a promising advancement towards enhancing the efficiency of the training process, reducing the computational effort required to attain a certain level of precision. This results not only in a potential cost reduction when training deep neural architectures but also in lowering the energy consumption and the carbon footprint while avoiding architectural modification and/or potential performance losses [166, 221].

Table 6.1: Model-optimizer performances over classification tasks. The proposed method is highlighted in gray, the best results are in bold, and the second bests are underlined.

Net	Optimizer	CIFAR 10				CIFAR 100			
		Acc ↑	T/Ep ↓	N_{ep}	k^*↓	Acc ↑	T/Ep ↓	N_{ep}	k^*↓
ResNet-18	Adam	81.67	6.46	250	139	52.30	6.55	250	31
	Adamax	82.69	7.75	250	228	53.14	7.79	250	86
	AdamW	81.24	6.52	250	**78**	52.46	6.49	250	35
	SGD	79.27	**5.96**	250	201	51.82	**5.85**	250	139
	Adagrad	82.02	6.51	250	106	53.03	6.38	250	250
	Nadam	81.27	6.63	250	181	51.33	6.65	250	**20**
	Radam	82.02	6.62	250	114	52.78	6.69	250	29
	CMA Light	82.83	7.45	250	119	54.95	7.64	250	179
	F-CMA	**83.43**	6.52	109	**78**	**56.64**	6.46	119	79
ResNet-50	Adam	82.34	12.65	250	242	49.76	12.67	250	102
	Adamax	83.64	15.45	250	142	56.65	15.53	250	**15**
	AdamW	81.67	12.75	250	77	52.02	12.76	250	27
	SGD	81.74	**11.95**	250	227	56.63	**11.91**	250	191
	Adagrad	83.01	12.57	250	223	50.91	12.53	250	201
	Nadam	81.43	13.04	250	198	49.17	13.05	250	81
	Radam	82.86	12.92	250	187	54.76	12.91	250	37
	CMA Light	84.96	13.90	250	195	59.58	13.52	250	70
	F-CMA	**85.55**	12.70	93	**75**	**60.51**	12.57	96	62
ResNet-152	Adam	77.89	25.58	250	205	46.78	25.79	250	226
	Adamax	82.91	33.05	250	139	54.51	33.11	250	47
	AdamW	79.57	25.91	250	239	45.61	25.90	250	227
	SGD	80.94	**23.77**	250	243	55.53	**23.77**	250	205
	Adagrad	82.55	25.31	250	192	45.08	25.24	250	41
	Nadam	80.83	26.53	250	202	46.73	26.46	250	168
	Radam	81.72	26.34	250	245	53.82	26.33	250	69
	CMA Light	81.38	27.38	250	**99**	47.06	29.34	250	**6**
	F-CMA	**83.98**	25.07	105	**99**	**58.90**	25.01	111	103
Wide ResNet-50	Adam	82.20	13.77	250	186	48.27	13.74	250	241
	Adamax	84.58	16.18	250	178	56.79	16.31	250	**53**
	AdamW	81.78	13.91	250	114	49.56	13.87	250	247
	SGD	83.06	**12.30**	250	232	57.96	**12.35**	250	221
	Adagrad	82.33	13.36	250	240	49.44	13.31	250	173
	Nadam	81.10	14.16	250	231	47.10	14.03	250	80
	Radam	83.37	14.25	250	137	55.46	14.16	250	147
	CMA Light	85.18	14.57	250	**61**	60.18	14.01	250	227
	F-CMA	**85.93**	13.10	85	70	**61.71**	12.99	98	59
Mobilenet V2	Adam	81.33	11.43	250	222	52.98	11.46	250	141
	Adamax	**83.50**	14.47	250	84	56.21	14.57	250	**43**
	AdamW	82.27	11.67	250	204	53.16	11.55	250	90
	SGD	80.88	**10.87**	250	245	54.99	**10.79**	250	248
	Adagrad	80.98	11.58	250	89	52.41	11.37	250	**43**
	Nadam	81.61	11.85	250	239	47.48	11.85	250	193
	Radam	82.22	11.59	250	188	55.25	11.68	250	50
	CMA Light	82.13	12.45	250	227	54.90	12.55	250	229
	F-CMA	83.01	11.68	78	**41**	**57.64**	11.39	122	98
Swin-B	Adam	85.68	55.96	250	249	59.23	55.82	250	102
	Adamax	86.59	57.92	250	248	62.47	57.74	250	241
	AdamW	81.53	56.42	250	238	58.95	56.03	250	84
	SGD	87.07	**53.29**	250	239	65.91	**53.41**	250	250
	Adagrad	86.65	55.27	250	243	63.93	55.09	250	243
	Nadam	14.38	56.44	250	-	56.15	56.61	250	71
	Radam	85.30	56.63	250	178	60.07	56.76	250	**64**
	CMA Light	**89.77**	59.31	250	94	69.91	59.05	250	148
	F-CMA	89.67	55.32	102	**64**	**70.81**	55.16	120	70

Chapter 7

Conclusion and further perspectives: enhanced line-search techniques on generative tasks

In this chapter, we present the main conclusions from this nearly three years doctoral research and provide a preview of future developments in this work. The chapter is structured as follows. In section 7.1, we review the key achievements of the controlled line-search-based approach, highlighting potential areas for further improvement. In section 7.2, we discuss the instance generation problem, which is significantly more challenging than the already analyzed regression and classification tasks, and introduce the diffusion model architecture. Next, in section 7.3, we summarize the initial results of a case study where we applied the line-search approach to image generation. In particular, we describe the Enhanced Armijo Line-search (EAL), an heuristic method within the CMA-framework developed in collaboration with MS student Lorenzo Ciarpaglini. The convergence theory remains to be fully explored but early results are presented as the starting point of a forthcoming research. Eventually, we make the final conclusions in 7.4.

7.1 The line-search approach: advantages and limitations

The CMA framework introduced in this doctoral research, as represented in the related literature, has shown increasingly better results, both in terms of efficiency and generalization performance. On one hand, we believe this work is valuable to the optimization community as it provides methods with strong convergence properties under mild assumptions, along with fully deterministic proofs of convergence, not relying on the expected value and not requiring bounds on the variance (see section 3.1). On the other hand, this approach appears to be equally valuable to the deep learning research community, particularly because the algorithms presented (especially CMA Light and, to an even greater extent, F-CMA) achieve better generalization performance than their competitors, with significantly reduced training times. This leads to lower computational costs, in terms of memory usage and in terms of energy consumption for the training, with positive consequences on the budget needed and on the carbon footprint of the

training itself. The use of the extrapolation line-search technique has proven to be an effective way to adapt the step-size, imitating the adaptive methods mechanism, but without the need to store past gradients in memory, which is a significant advantage. As highlighted in earlier chapters, we are effectively combining the strengths of both adaptive and non-adaptive methods while avoiding their respective drawbacks.
Nonetheless, this represents only part of the broader picture. There remain several promising areas for improvement with this newly developed method. For instance, while the deterministic proofs are valuable as they demonstrate convergence to a stationary point of the objective function, they depend on the step-size eventually decreasing to zero (monotonically or not), making it impossible to establish worst-case complexity bounds, as it is done in literature with other stochastic methods.
Moreover, the deterministic nature of the current approach prevents CMA-framework from being entirely objective-function-free, as some objective function evaluations are still required, even in F-CMA, at the beginning and end of the line-search, to ensure that the sequence of points generated by the algorithm remains within the level set of the objective function. Incorporating a stochastic analysis using estimators for the objective function is certainly an interesting future research direction.
Additionally, it must be underlined that these methods perform optimally when the computational overhead of line-search is minimal relative to the total number of epochs. In highly unstable environments, such as byzantine learning [118] with heavily corrupted information, the algorithms would still converge but at the cost of an immense computational effort. Extending our results to scenarios where only estimators are used would therefore be highly beneficial.
Lastly, the tasks we have tackled thus far are somewhat limited. While competitive from a research perspective, they are not yet comparable to the industrial-scale AI systems that involve models with billions or trillions of trainable parameters. The computational resources required to train such models are far beyond the reach of most research institutions. A particularly relevant example is generative machine learning, where the number of parameters needed is enormous.
Following the beginning of this research path towards the enhancement of the CMA-framework itself and the extension to stochastic settings, we further present the early results on a case-study involving the reconstruction of the MNIST[1] dataset images.

7.2 The challenge of GenAI

Generative tasks in supervised deep learning, differently from what we have seen so far, are inherently computationally challenging, independently from the dimension of the model, i. e., from the number of variables of problem (2.6). In the field of computer vision, image and video generation tasks play a structural role in many different applications [32, 63, 122]. Furthermore, even considering the NLP world, it is remarkable that most large-language models have been integrating features that enable image generation from caption.

[1] https://yann.lecun.com/exdb/mnist/

However, for the deep learning community instance generation still represents one of the hardest and most advanced research areas. As deep learning researchers are trying to improve the state-of-the-art following different paths, often working on architectural modifications on the models, it is clear that training efficiency is a first-order priority for this emerging domain. In our opinion, the development of novel and more efficient optimizers can give a contribution to this field as well and is somehow the next step of our research.

7.2.1 Instance generation: an overview

Recalling what we have discussed in section 2.3 presenting the main types of neural architectures for classical supervised learning tasks, we can immediately get an idea of why instance generation is a particularly hard task.
Firstly, in generative tasks, the model must generate complex and high-dimensional outputs, such as images, text, or audio, from a simpler input representation. This is significantly harder than predicting a single scalar (regression) or a class label (classification), where the possible outputs are far fewer and easier to define. The generative model has to learn not only the key features but also how to compose them to generate new, plausible examples from the learned distribution. For instance, if we consider our standard datasets CIFAR10, in a classification task the output is a probability vector whose dimension is determined by the number of classes in the dataset. Conversely, if the task is to generate images of objects from CIFAR10, the output is a $3 \times 32 \times 32$ images, which means, mathematically speaking, a vector in $\mathbb{R}^{3072}$. Considering high-resolution images, it is clear how the output dimension is remarkably higher than in classification tasks.
Secondly, while in classification and regression tasks, the goal is to approximate a mapping from input to a well-defined output space, generative models must learn the underlying data distribution itself, capturing all possible variations in the data (e.g., different ways a cat can appear in an image). The generative process inherently involves uncertainty, as there are many valid outputs for a single input. Capturing this complexity requires more sophisticated techniques and a slightly different mathematical approach.
The mathematical framework, as well as the formulation of problem (2.6) in the case of generative tasks can significantly vary depending on the architectural setting adopted. Reviewing in detail the main architectural solutions developed in this field is out of the scope of this doctoral research and the reader can find a thorough comparison of all existing methods in chapters 15,16,17, and 18 of [169]. However, the most commonly used state-of-the-art architectures for generative tasks in computer vision are VAEs (Variational Auto-Encoders) and diffusion models. They are both based on a primitive statistical concept - the maximization of the likelihood.
More formally, given the input data set $\chi = (x_1, x_2, \ldots, x_P)$, building a generative model for χ means approximating the unknown probability distribution $\mathcal{P}(x)$ from which χ is generated. If $\varphi_\chi(w)$ is a parametrized probability distribution to approximate $\mathcal{P}(x)$, then training the generative model φ means solving the following problem:

$$\min_{w \in \mathbb{R}^n} \mathcal{L}\left[\mathcal{P}(x), \varphi_\chi(w)\right] \tag{7.1}$$

where $\mathcal{L}(p, q)$ is a metric measuring the distance between probability distributions p and q (e.g., the Kullback-Leibler divergence).
Since we do not have any closed expression for the input data distribution $\mathcal{P}(x)$, the problem 7.1 must be solved indirectly and there are different approaches depending on the single task and the architecture. For the purpose of presenting the early findings of this research, we only explain the underlying idea of diffusion models, which are among the largest and best performing generative models used in deep learning.

7.2.2 The diffusion model architecture

The diffusion models, following the same theoretical approach of VAEs [105, 106], are based on solving the problem 7.1 by maximizing the *likelihood* of having the current data points. It is intuitive to understand, that, given a data set χ, the most straightforward way to infer the unknown data distribution from a parametrized distribution $\varphi_\chi(w)$ is to choose:

$$w^* \in \arg\max_{w \in \mathbb{R}^n} \prod_{x \in \chi} \texttt{Prob}(x|w) \tag{7.2}$$

where $\texttt{Prob}(x|w)$ is the probability that, when the network parameters are w, the generated instances are exactly the ones observed in the data set we want to learn the distribution from.

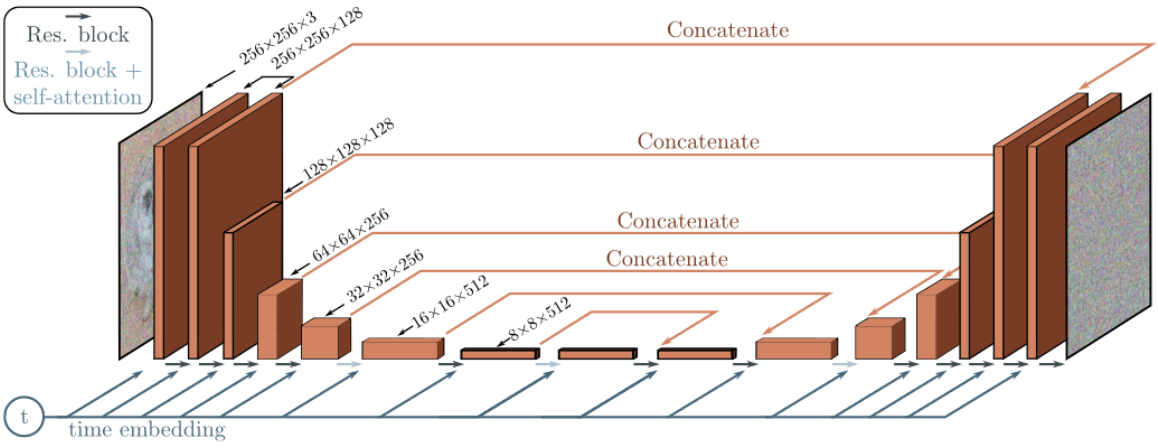


Figure 7.1: U-Net diffusion model architecture, as schematically represented in [169].

Finding w^* through (7.2) involves solving a max-likelihood well-known problem in statistics. The architecture of the diffusion model, illustrated schematically in figure 7.1 with the U-Net design [179], which is also utilized in our experiments, consists of an encoder and a decoder. Given an input x, the encoder (i.e., the left side of the network in figure 7.1, progressively through T layers adds Gaussian noise to the data in the following recursive form:

$$z_1 = \sqrt{1-\beta_1}x + \sqrt{\beta_1}\varepsilon_1$$
$$z_t = \sqrt{1-\beta_t}z_{t-1} + \sqrt{\beta_t}\varepsilon_t \quad \text{for } t = 2, \ldots, T$$

where T is the number of encoder layers, β_t are hyper-parameters, and ε_t represents standard

Gaussian noise added at each layer. We remark that, as in [169], we adopt the t-letter notation, as the diffusion models replicates the statistical diffusion process in time-steps, which is typical in discrete-time Markov chains. Given an input $z_0 = x$ and given T encoder layers, it is straightforward to verify that the conditioned probability distribution of each intermediate output is $q(z_t|z_{t-1}) \sim \texttt{Normal}\left[\mu = \sqrt{1-\beta_t}z_{t-1}, \sigma^2 = \beta_t\mathbf{I}\right]$, being $\mathbf{I}$ the identity matrix. Hence, the joint distribution of the final output conditioned to the data point x can be written as:

$$q(z_1, z_2, \ldots, z_T|x) = q(z_1|x)\prod_{t=2}^{T} q(z_t|z_{t-1}) \tag{7.3}$$

Now, naming $\alpha_t = \prod_{j=1}^{t}(1-\beta_j)$, it is possible to derive (see section 18.2.1 of [169]) the distribution of the marginal conditioned probabilities $q(z_t|x) \sim \texttt{Normal}\left[\mu = \sqrt{\alpha_t}x, \sigma^2 = (1-\alpha_t)\mathbf{I}\right]$. The second part (right part of figure 7.1) of the diffusion model, the decoder, is tasked with re-constructing the instance x from the final embedding z_T. To this aim, we firstly need to explicit the conditioned distribution $q(z_{t-1}|z_t, x)$, which is possible using Bayes' formula and marginalization rule, as explained in detail in section 18.2.4 of [169]. We can find that:

$$q(z_{t-1}|z_t, x) \sim \texttt{Normal}\left[\mu = \frac{1-\alpha_{t-1}}{1-\alpha_t}\sqrt{1-\beta_t}z_t + \frac{\sqrt{\alpha_{t-1}}}{1-\alpha_t}x, \sigma^2 = \frac{1-\alpha_{t-1}}{1-\alpha_t}\beta_t\mathbf{I}\right] \tag{7.4}$$

Now, we observe that we can formalize the re-construction process as a *supervised de-noising* task, where the target distribution at each step is given by (7.4). More formally, in computer vision applications (as in the case of U-Net) the real unconditioned distribution $q(z_{t-1}|z_t)$ is approximate with a Gaussian variable and the decoder consists in T convolutional blocks (recall section 2.3.2 for the details) that approximate the mean values of these distributions at each step of the diffusion process as follows:

$$\texttt{Prob}(z_T) = \texttt{Normal}\left[\mu = \mathbf{0}, \sigma^2 = \mathbf{I}\right]$$

$$\texttt{Prob}(z_{t-1}|z_t, x; w_t) = \texttt{Normal}\left[\mu = \varphi_t(z_t; w_t), \sigma^2 = \sigma_t^2\mathbf{I}\right]$$

$$\texttt{Prob}(x|z_1; w_1) = \texttt{Normal}\left[\mu = \varphi_1(z_1; w_1), \sigma^2 = \sigma_1^2\mathbf{I}\right]$$

where w_t is the sub-vector of the trainable parameters belonging to the t-th layer of the decoder and φ_t is the output of this layer.
It is now possible to rewrite problem (7.2) in a more tractable way as the maximization of the log-likelihood of the dataset, given the network parameters, and the Kullback-Leibler divergence between the approximated distributions $\texttt{Prob}(z_{t-1}|z_t, x; w_t)$ and the real distributions $q(z_{t-1}|z_t, x)$ from (7.4). For a dataset of P samples and with T diffusion steps we obtain that the loss function is:

$$w^* \in \arg\max_{w\in\mathbb{R}^n} \sum_{i=1}^{P}\left(\log\left[\texttt{Prob}(x_i|z_1; w_1)\right] - \sum_{t=2}^{T} D_{KL}\left[q(z_{t-1}|z_t, x_i)\|\texttt{Prob}(z_{t-1}|z_t, x_i; w_t)\right]\right) \tag{7.5}$$

To achieve a better computational performance, in most applications, as in U-Net, the network is re-parametrized such that the predicted outputs are not directly the means of normal distributions, but instead the Gaussian noises that have been added by the encoder. This makes the optimization problem less heavy from a computational perspective because the problem (7.5) is reformulated as the mean squared error between the predicted noise and the one actually added by the encoder, removing the dependency from the probability distributions.
Without entering the implementation details of the diffusion models, we remark that the challenging nature of problem (7.5) is not only related to the number of trainable parameters, as for the other tasks we have already tackled in this thesis. The diffusion and de-noising process are *not* the equivalent of the forward process in traditional neural networks. More specifically the diffusion process does not only require matrix dot-products but relies on dynamically sampling all the intermediate outputs $z_1, \dots, z_T$ from given distribution. This process is inherently iterative and not parallelizable, making computationally unaffordable the evaluation of the objective function on the entire dataset.
Eventually, it is intuitive to understand that the larger the number of diffusion steps T, the better the model can approximate the real data distribution. In real-world application, T can easily be in the order of thousands. The diffusion process in this case is extremely heavy and has a high impact on the cost on the carbon footprint of the training [166].

7.3 Re-constructing MNIST - a case study

The iterative nature of the diffusion process, where distributions are computed sequentially, combined with the increasing complexity of diffusion models and generative tasks, presents a significant challenge in adapting the CMA framework to these tasks. The need to compute the full objective function over all data points, even if seldomly, would make this approach computationally prohibitive.
F-CMA offers a more practical alternative, providing a feasible direction for applying such techniques in these scenarios. However, even in F-CMA, the line-search procedure requires evaluating the entire objective function $f(w)$ at least once, before accepting a candidate point to ensure it lies within the level set $\mathcal{L}(w^0) = \{w \in \mathbb{R}^n : f(w) \leq f(w^0)\}$.
The central aim of this computational study was to develop a heuristic framework that integrates F-CMA into a fully objective function-free optimization paradigm. Accepting to evaluate the initial training loss $f(w^0)$, we propose to replace algorithm 6.2 with a new line-search heuristic, based on the Armijo conditions [6], using a stochastic estimator ψ, as defined in equation (6.8), which computes the loss over only a subset of samples in the dataset.
The line-search heuristic, which we named Enhanced Armijo Line-search (EAL) is reported in algorithm 7.1. The algorithm is divided into two stages. In the first stage (cycle at step 3), the line-search follows the standard Armijo condition mechanism, where the true gradient of the objective function is replaced by the search direction $d^k = -\sum_{i=1}^{P} \nabla f_i(\widetilde{w}_{i-1}^k)$ obtained from the RR epoch (algorithm 6.1). Once an α satisfying the condition at step 3 is found, the algorithm continues to decrease the step size as long as the estimate of the objective function

Algorithm 7.1 Enhanced Armijo Linesearch (EAL)

Input: $(w^k, d^k, \alpha^k, \gamma, \zeta)$: $w^k \in \mathbb{R}^n, d^k \in \mathbb{R}^n, \alpha^k > 0$, $\gamma \in (0,1)$, $\zeta \in (0,1)$
Set $\alpha_0 = \alpha^k$, $\delta = 1 - \zeta$
while $\psi(w^k + \alpha_i d^k) \leq \psi(w^k) - \gamma\alpha_i \|d^k\|^2$ **do**
  Set $\alpha_{i+1} = \alpha_i\zeta$, $i = i + 1$
end while
while true **do**
  if condition is satisfied **then**
    break
  end if
  if $\psi(w^k + \alpha_i d^k) < \psi(w^k + \alpha_{i-1} d^k)$ **then**
    if $\|\psi(w^k + \alpha_i d^k) - \psi(w^k + \alpha_{i-1} d^k)\| < \epsilon$ **then**
      break
    end if
    Set $\alpha_{i+1} = \alpha_i\zeta$, $i = i + 1$
  else
    Update $\alpha_i = \alpha_{i-1}$, $\delta = \delta/2$, $\zeta = \zeta + \delta$, $\alpha_{i+1} = \alpha_i\zeta$, $i = i + 1$
  end if
end while
Output: α_i

value continues to decrease. When this estimate stops decreasing, the step size is reverted to the previous value and reduced by a smaller amount. This process is iterated for a fixed number of times where ψ ceases to decrease, indicating that the we are approaching a neighborhood of a stationary point. The underlying idea behind this heuristic is to improve the accuracy of the solution and help navigate closer to stationary points.

To quantify the possible benefits of this approach from a mathematical perspective, we need to evaluate the average maximum squared error that can be made when using ψ instead of f, namely $E_{max}(w) = \mathbb{E}[(\psi(w) - f(w))^2]$. Intuitively, we can argue that $E_{max}(w)$, being directly related to the second moment of $\psi(w)$, can be bounded depending on the fraction $\kappa \in (0, 1]$ of data points used to estimate f. Furthermore, it is trivial to verify that if $\kappa = 1$, then $E_{max}(w) = 0$ for any w.

We believe that bounding this error is possible and can lead to novel convergence results and this is the first research direction we are already working on.

However, despite not having a developed theory yet, we present to the reader a computational study that has been made on MNIST dataset, training U-Net diffusion model with F-CMA, where the extrapolation line-search has been replaced by EAL as in algorithm 7.1. This computational study is a result of a joint work with MS student in Artificial Intelligence Lorenzo Ciarpaglini and has been possible thanks to the use of high-performing GPUs made avaliable by ALCOR Lab.

7.3.1 Computational experiments and early results

MNIST dataset contains 50,000 training and 10,000 test black-and-white images of handwritten digits of dimension $1\times28\times28$. We have performed five multi-start run, training for 100 epochs the

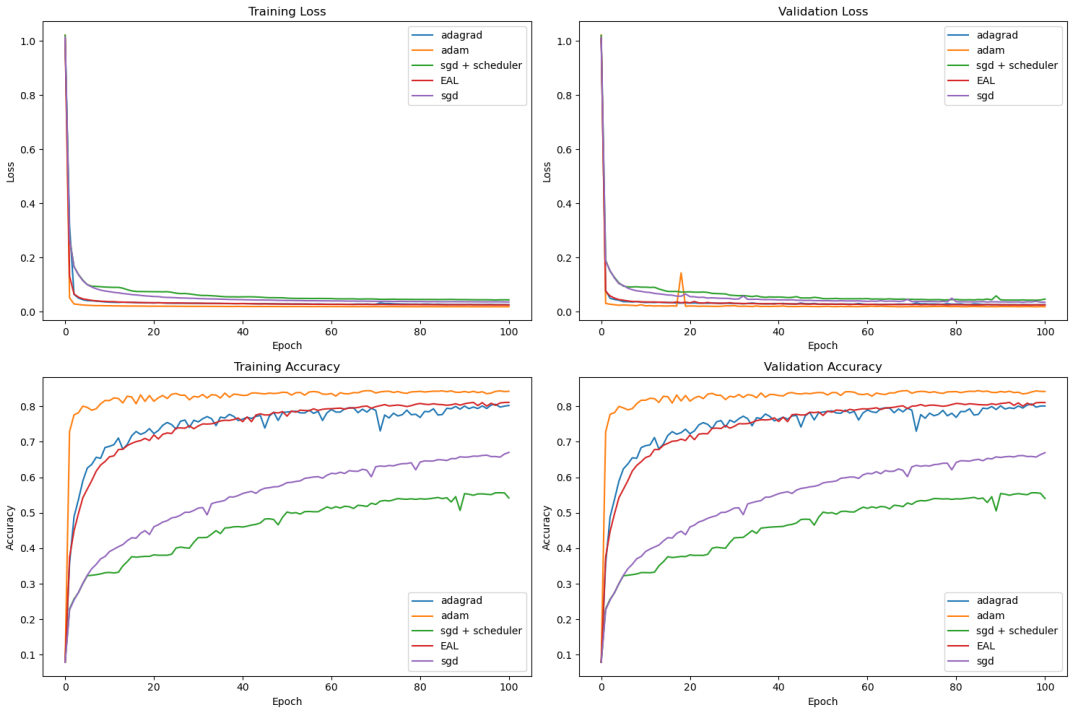


Figure 7.2: Performance comparison of different optimization algorithms on the image generation task, using a fraction $\kappa = 0.5$ of the dataset to compute the stochastic objective function.

U-Net architecture (about 1.7×10^8 trainable parameters) on generating images of handwritten digits from Gaussian black-and-white noise. We have tested F-CMA with EAL against SGD with fixed step-size tuned after a grid search, SGD with decreasing step-size according to a pre-fixed schedule[2], Adam and Adagrad.

As shown in figure 7.1, the Adam optimizer outperforms the other methods, achieving the highest accuracy in the shortest time. However, F-CMA with EAL ranks second across all performance metrics, surpassing other algorithms such as SGD, SGD with scheduler, and adaptive methods like Adagrad.

When increasing the fraction κ of the dataset used to compute the estimate of the objective function in the EAL, the computational results seem to confirm our hypothesis on the relation between κ and the error. In figure 7.3, we can indeed observe that, while Adam remains the best performing algorithm, F-CMA with EAL is more stable in terms of both accuracy and training loss. It is also remarkable that in any setting, the new algorithm consistently outperforms SGD both with a tuned fixed learning rate and with decreasing step-size.

Finally, in figure 7.4 we present the results of image generation obtained using the same model architecture, but with different optimization algorithms. It is evident that the choice of the optimizer can significantly impact the quality of the generated images, even when the underlying model remains unchanged. In particular, when looking at figure 7.4, we observe that, within the fixed number of 100 epochs, only F-CMA with EAL, besides Adam, can train U-Net sufficiently to generate realistic images of handwritten digits. Other algorithms would require much more

[2]it has been used the StepLR scheduler from https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.StepLR.html

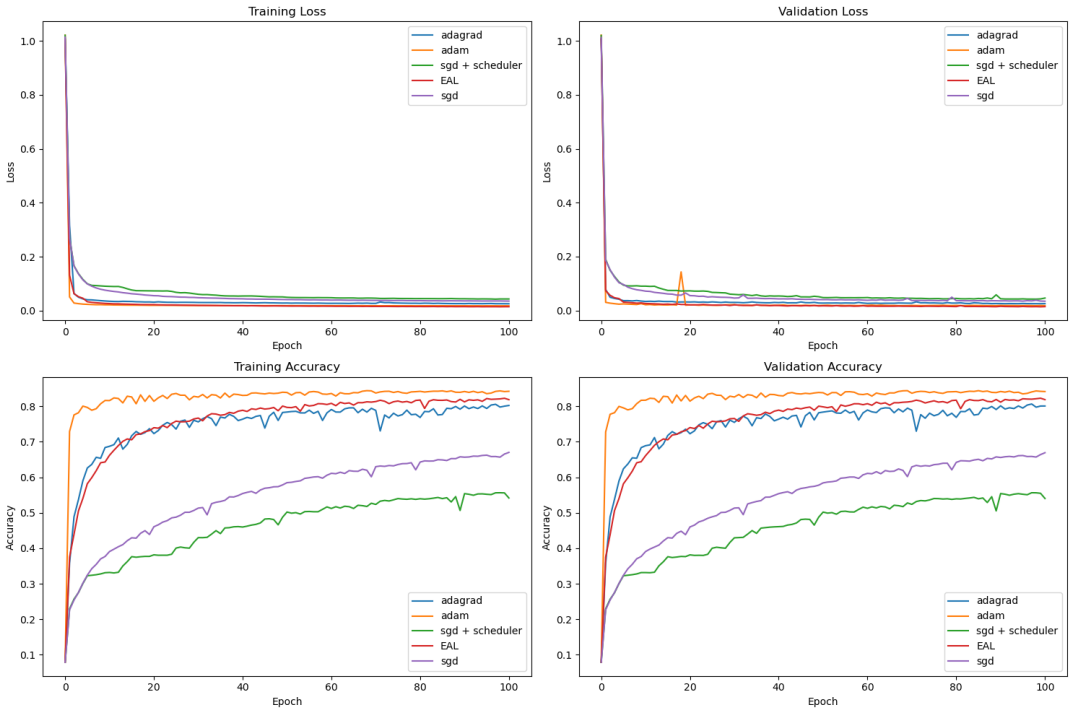


Figure 7.3: Performance comparison of different optimization algorithms on the image generation task, using a fraction $\kappa = 0.75$ of the dataset to compute the stochastic objective function.

training epochs.

7.4 Conclusion

As remarked at the beginning of this chapter, we believe that the CMA-framework represents a valuable contribution to the optimization-driven innovation in the field of deep learning. Although the idea of replacing the adaptive mechanism with a line-search comes with limitation and bottlenecks as well, the algorithms of the CMA family proved themselves as possible alternatives to standard off-the-shelf optimizers. Furthermore, the computational study we have performed on the image generation task using a CMA-based heuristic approach lead to remarkable results both in terms of training efficiency (albeit still not outperforming Adam) and in terms of optimization theory. We believe that F-CMA with EAL can be the starting point of a forthcoming research, centered on the extension of the CMA theory to a stochastic setting, reformulating our initial target problem (2.6) as a stochastic optimization problem. Having a stochastic counterpart of the deterministic theory of CMA would be also interesting in terms of convergence rate and worst-case complexity analysis, in order to better assess the guarantees of our framework compared to the most recent state-of-the-art presented in chapter 3.
Beyond the merely heuristic nature of the proposal presented in this last chapter, the results obtained so far and the incremental improvements we have been observing for the past two years strengthen our professional and personal trust in this research direction - *multi pertransibunt et augebitur scientia.*

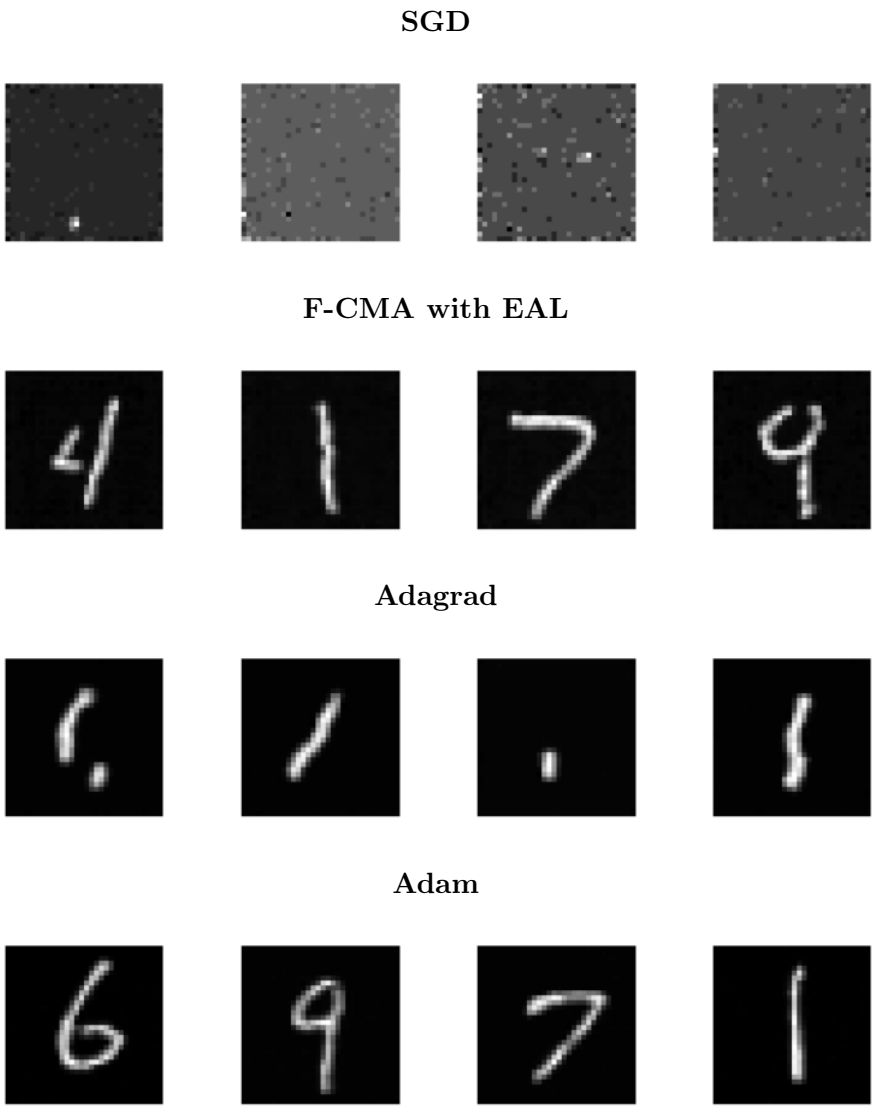


Figure 7.4: Generated images using the same architecture but different optimization algorithms.

Appendices

Appendix A

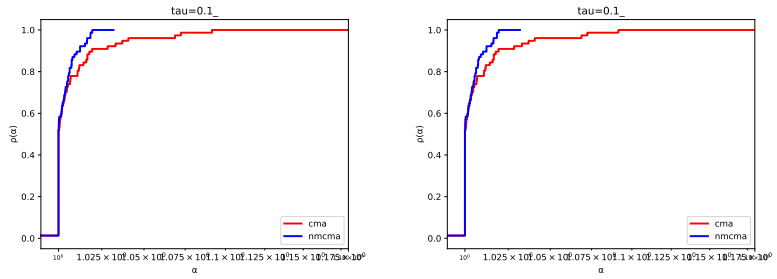
Additional material

In this Appendix we report figures and tables that are recalled in the present thesis to enforce and/or highlight the validity of our claims. The choice of moving figures and tables in the appendix has two main reasons:

- some figures and tables are too long and would take too much space if inserted within the text, leading the reader to confusion;
- the additional material reported here is not strictly necessary to understand and verify the claims in the main text.

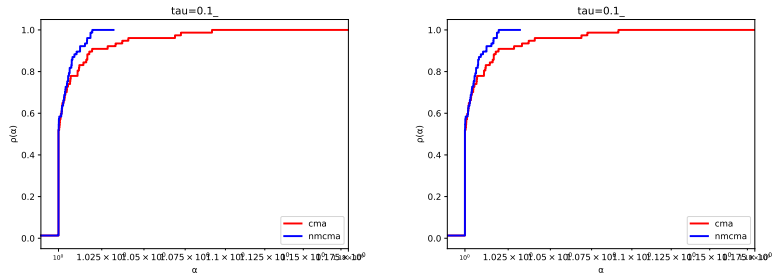
Furthermore, trying to make it to the reader easier to find figures and/or tables, the additional material is divided in sections corresponding to the chapters of this thesis.

A.1 Additional figures and tables from chapter 4 (CMA)



(a) $\tau = 10^{-1}$

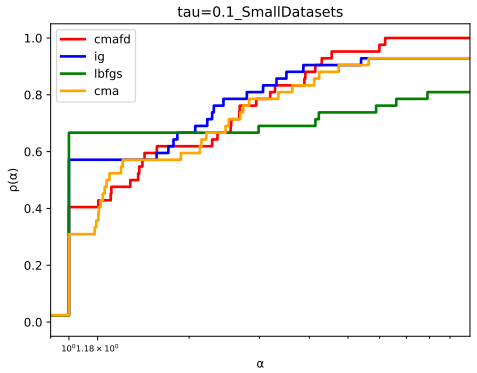
(b) $\tau = 10^{-2}$



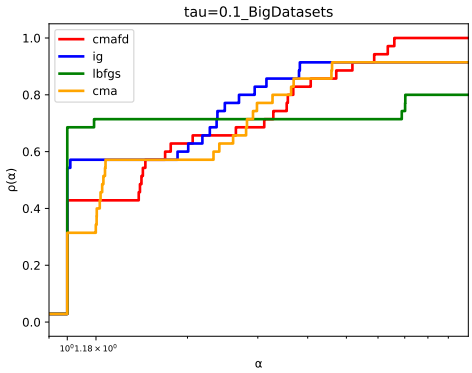
(c) $\tau = 10^{-3}$

(d) $\tau = 10^{-4}$

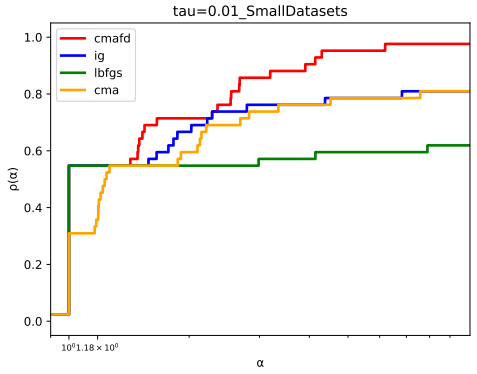
Figure A.1: Performance profile of CMA on the training loss against NMCMA with decreasing step-size.



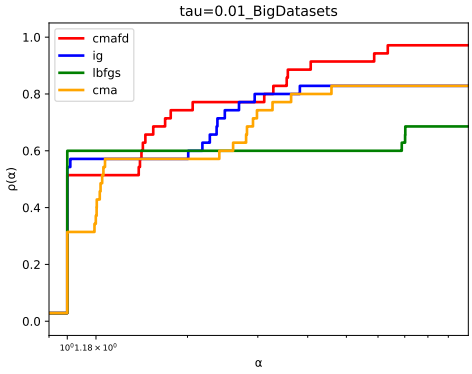
(a) Small datasets: $\tau = 10^{-1}$



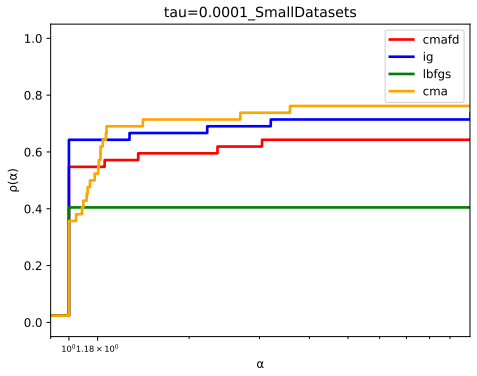
(b) big-size datasets: $\tau = 10^{-1}$



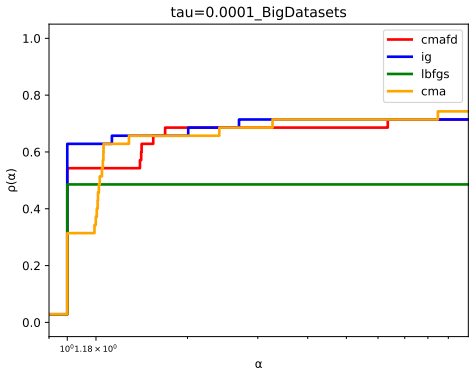
(c) Small datasets $\tau = 10^{-2}$



(d) big-size datasets $\tau = 10^{-2}$

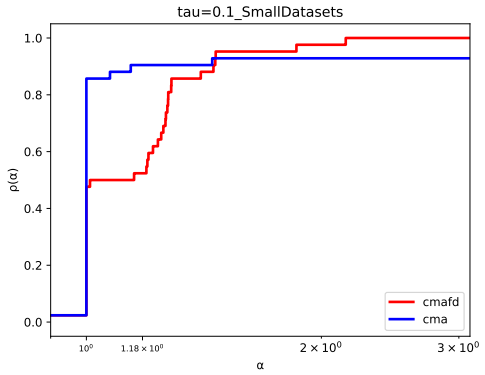


(e) Small datasets $\tau = 10^{-4}$

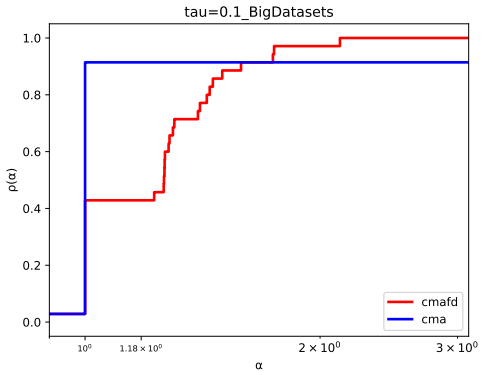


(f) big-size datasets $\tau = 10^{-4}$

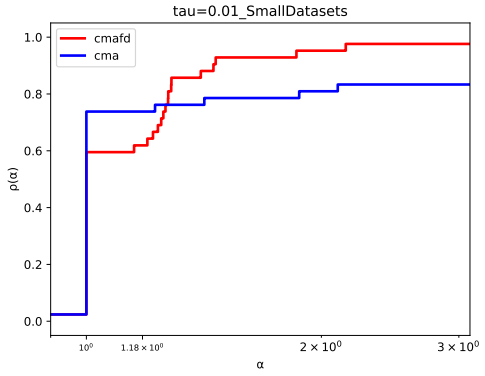
Figure A.2: Performance profiles of CMA-FD against L-BFGS and IG on all the networks for different values of τ. Results for small-size datasets are reported to the left, and for big-size datasets to the right.



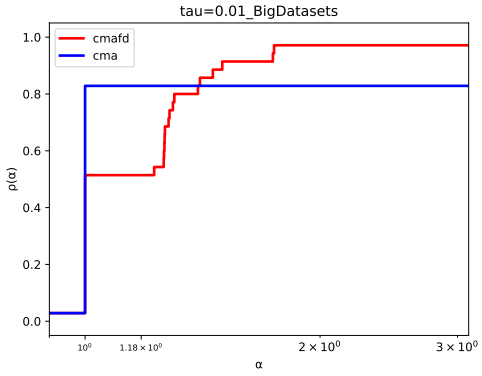
(a) Small datasets: $\tau = 10^{-1}$



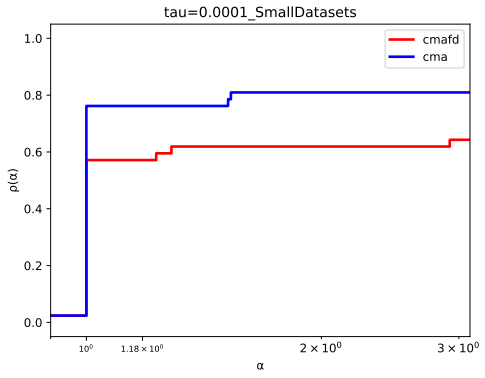
(b) big-size datasets: $\tau = 10^{-1}$



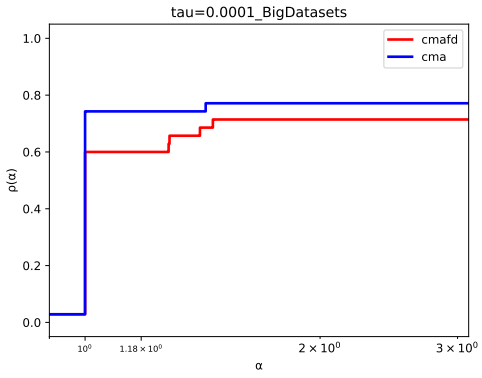
(c) Small datasets $\tau = 10^{-2}$



(d) big-size datasets $\tau = 10^{-2}$



(e) Small datasets $\tau = 10^{-4}$



(f) big-size datasets $\tau = 10^{-4}$

Figure A.3: Performance profiles of CMA-FD against CMA on all the networks for different values of τ. Results for small-size datasets are reported to the left, and for big-size datasets to the right.

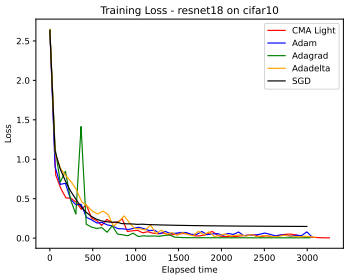
A.2 Additional figures and tables from chapter 5 (CMA Light)

Table A.1: Test Loss on the problems in table 4.2 with the five tested algorithms. The loss reported is the average over the five multistart runs.

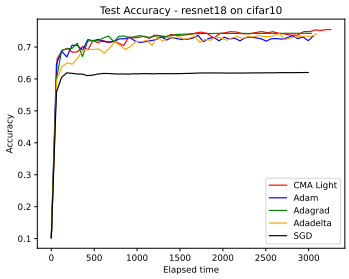
Ailerons	CMA Light	Adam	Adagrad	Adadelta	SGD
S	2.49e-3	8.27e-4	9.09e-4	9.04e-4	4.60e-3
M	4.44e-3	7.48e-4	1.08e-3	1.59e-3	5.21e-3
L	4.43e-3	7.38e-4	8.00e-4	4.43e-3	4.43e-3
XL	4.43e-3	4.43e-3	8.01e-4	4.43e-3	4.43e-3
XXL	4.45e-3	4.43e-3	4.44e-3	4.43e-3	4.43e-3
XXXL	4.44e-3	4.43e-3	4.44e-3	4.43e-3	4.43e-3
4XL	4.45e-3	4.43e-3	4.44e-3	4.43e-3	4.43e-3
Beijing PM25					
S	2.41e-3	1.59e-3	1.67e-3	2.29e-3	2.82e-3
M	2.75e-3	1.45e-3	1.71e-3	2.67e-3	2.93e-3
L	2.74e-3	1.46e-3	1.53e-3	2.86e-3	2.74e-3
XL	2.74e-3	2.74e-3	2.15e-3	2.86e-3	2.74e-3
XXL	2.80e-3	2.74e-3	2.72e-3	2.77e-3	2.74e-3
XXXL	2.83e-3	2.74e-3	2.84e-3	2.77e-3	2.74e-3
4XL	2.85e-3	2.74e-3	2.84e-3	2.77e-3	2.74e-3
Bikes Sharing					
S	7.57e-3	2.14e-3	3.72e-3	3.65e-3	1.21e-2
M	1.18e-2	1.70e-3	2.96e-3	3.35e-3	1.20e-2
L	1.18e-2	1.14e-3	2.87e-3	1.18e-2	1.18e-2
XL	1.18e-2	1.18e-2	2.79e-3	1.18e-2	1.18e-2
XXL	1.18e-2	1.18e-2	8.29e-3	1.18e-2	1.18e-2
XXXL	1.18e-2	1.18e-2	1.18e-2	1.19e-2	1.18e-2
4XL	1.18e-2	1.18e-2	1.18e-2	1.19e-2	1.18e-2
BlogFeedback					
S	1.93e-4	1.42e-4	1.57e-4	1.95e-4	3.04e-4
M	2.18e-4	1.03e-4	1.29e-4	2.07e-4	3.75e-4
L	2.18e-4	9.78e-5	1.02e-4	2.18e-4	2.18e-4
XL	2.18e-4	2.18e-4	1.20e-4	2.18e-4	2.19e-4
XXL	2.18e-4	2.18e-4	1.52e-4	2.18e-4	2.18e-4
XXXL	2.18e-4	2.18e-4	2.29e-4	2.18e-4	2.18e-4
4XL	2.18e-4	2.18e-4	2.32e-4	2.18e-4	2.18e-4
California					
S	1.37e-2	6.89e-3	7.40e-3	1.07e-2	1.95e-2
M	1.90e-2	6.41e-3	7.27e-3	1.03e-2	1.99e-2
L	1.90e-2	5.93e-3	6.71e-3	1.90e-2	1.90e-2
XL	1.90e-2	1.90e-2	6.91e-3	1.90e-2	1.90e-2
XXL	1.91e-2	1.90e-2	1.90e-2	1.90e-2	1.90e-2
XXXL	1.91e-2	1.90e-2	1.90e-2	1.90e-2	1.90e-2
4XL	1.91e-2	1.90e-2	1.90e-2	1.90e-2	1.90e-2

Table A.2: Test Loss on the problems in table 4.2 with the five tested algorithms. The loss reported is the average over the five multistart runs.

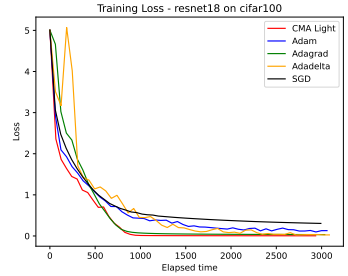
Covtype	CMA Light	Adam	Adagrad	Adadelta	SGD
S	1.25e-2	9.68e-3	1.20e-2	1.02e-2	1.80e-2
M	1.79e-2	1.05e-2	1.17e-2	1.13e-2	1.83e-2
L	1.81e-2	1.10e-2	1.16e-2	1.80e-2	1.80e-2
XL	1.81e-2	1.80e-2	1.17e-2	1.80e-2	1.80e-2
XXL	1.81e-2	1.80e-2	1.50e-2	1.80e-2	1.80e-2
XXXL	1.81e-2	1.80e-2	1.71e-2	1.80e-2	1.80e-2
4XL	1.82e-2	1.80e-2	1.82e-2	1.80e-2	1.80e-2
Mv					
S	3.96e-3	5.09e-5	1.80e-3	2.77e-4	1.87e-2
M	1.85e-2	1.25e-5	4.99e-4	4.54e-4	1.99e-2
L	1.85e-2	1.78e-5	2.51e-4	1.87e-2	1.85e-2
XL	1.85e-2	1.12e-2	1.34e-4	1.87e-2	1.85e-2
XXL	1.87e-2	1.85e-2	7.50e-3	1.87e-2	1.85e-2
XXXL	1.88e-2	1.85e-2	1.86e-2	1.87e-2	1.85e-2
4XL	1.88e-2	1.85e-2	1.86e-2	1.88e-2	1.85e-2
Protein					
S	2.24e-3	9.44e-4	1.58e-3	1.03e-3	2.81e-3
M	2.78e-3	8.62e-4	1.06e-3	1.20e-3	2.95e-3
L	2.78e-3	8.80e-4	9.06e-4	2.78e-3	2.78e-3
XL	2.78e-3	2.78e-3	9.92e-4	2.78e-3	2.78e-3
XXL	2.78e-3	2.78e-3	2.79e-3	2.78e-3	2.78e-3
XXXL	2.79e-3	2.78e-3	2.79e-3	2.78e-3	2.78e-3
4XL	2.80e-3	2.78e-3	2.80e-3	2.78e-3	2.78e-3
Sido					
S	8.33e-3	8.20e-3	7.77e-3	8.14e-3	1.17e-2
M	1.20e-2	7.60e-3	7.58e-3	7.99e-3	1.23e-2
L	1.20e-2	7.79e-3	7.72e-3	1.24e-2	1.20e-2
XL	1.20e-2	1.21e-2	8.76e-3	1.24e-2	1.20e-2
XXL	1.21e-2	1.20e-2	9.65e-3	1.20e-2	1.20e-2
XXXL	1.21e-2	1.20e-2	1.21e-2	1.20e-2	1.20e-2
4XL	1.21e-2	1.20e-2	1.21e-2	1.20e-2	1.20e-2
Skin Non Skin					
S	2.64e-2	4.81e-3	2.37e-2	4.06e-3	5.50e-2
M	2.86e-2	2.13e-3	7.73e-3	5.13e-3	5.65e-2
L	5.53e-2	4.64e-3	6.22e-3	5.86e-3	5.53e-2
XL	5.53e-2	4.72e-2	1.30e-2	5.53e-2	5.53e-2
XXL	5.53e-2	5.54e-2	4.56e-2	5.53e-2	5.53e-2
XXXL	5.53e-2	5.54e-2	5.53e-2	5.53e-2	5.53e-2
4XL	5.53e-2	5.54e-2	5.53e-2	5.53e-2	5.53e-2
YearPredictionMSD					
S	4.20e-3	3.76e-3	3.92e-3	3.87e-3	5.04e-3
M	5.05e-3	3.78e-3	4.03e-3	4.04e-3	6.39e-3
L	5.04e-3	3.69e-3	3.93e-3	5.05e-3	5.04e-3
XL	5.04e-3	5.05e-3	3.88e-3	5.08e-3	5.04e-3
XXL	5.05e-3	5.05e-3	5.04e-3	5.07e-3	5.04e-3
XXXL	5.06e-3	5.05e-3	5.04e-3	5.08e-3	5.04e-3
4XL	5.07e-3	5.05e-3	5.05e-3	5.14e-3	5.04e-3



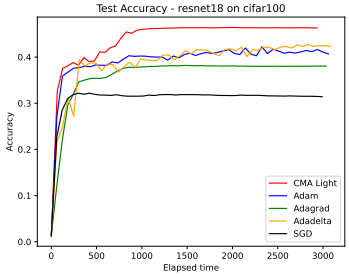
(a) CIFAR 10 - Training Loss



(b) CIFAR 10 - Test Accuracy

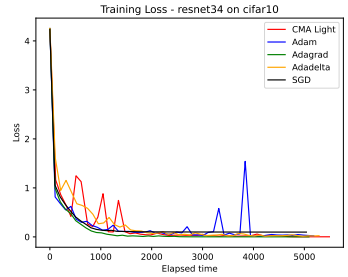


(c) CIFAR 100 - Training Loss

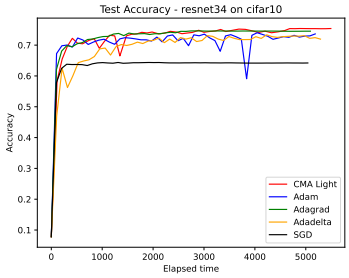


(d) CIFAR 100 - Test Accuracy

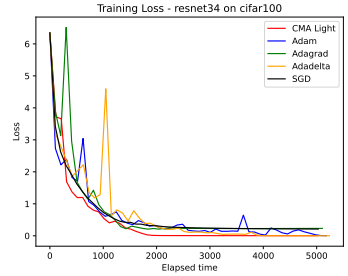
Figure A.4: Training loss and test accuracy plot during the training of CMA Light and its competitors in Chapter 5 on the ResNet-18 architecture.



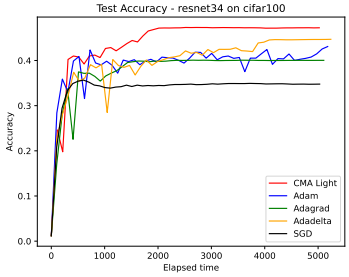
(a) CIFAR 10 - Training Loss



(b) CIFAR 10 - Test Accuracy

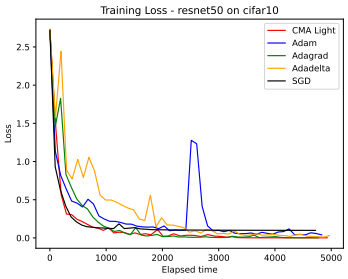


(c) CIFAR 100 - Training Loss

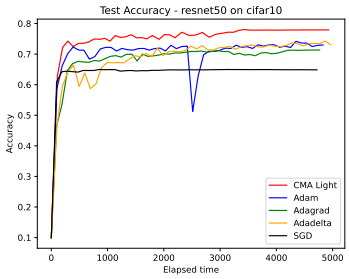


(d) CIFAR 100 - Test Accuracy

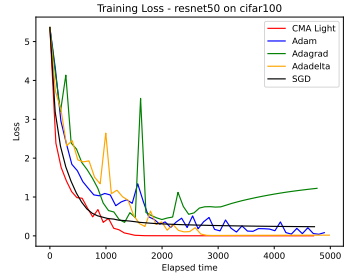
Figure A.5: Training loss and test accuracy plot during the training of CMA Light and its competitors in Chapter 5 on the ResNet-34 architecture.



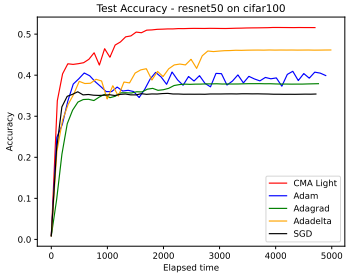
(a) CIFAR 10 - Training Loss



(b) CIFAR 10 - Test Accuracy

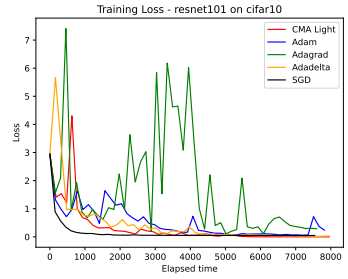


(c) CIFAR 100 - Training Loss

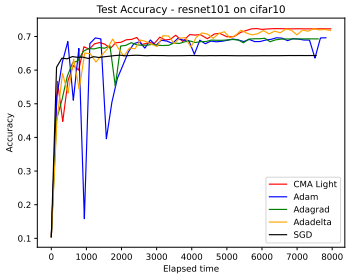


(d) CIFAR 100 - Test Accuracy

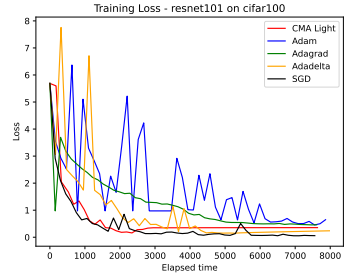
Figure A.6: Training loss and test accuracy plot during the training of CMA Light and its competitors in Chapter 5 on the ResNet-50 architecture.



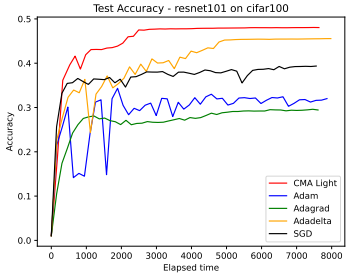
(a) CIFAR 10 - Training Loss



(b) CIFAR 10 - Test Accuracy

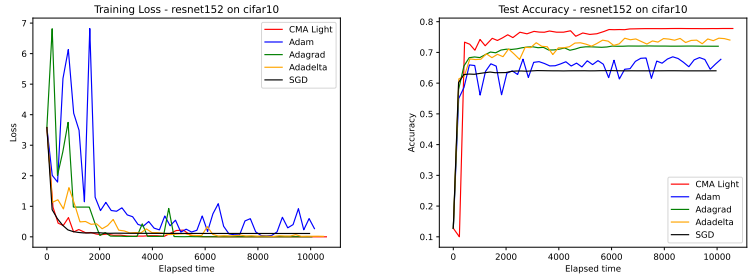


(c) CIFAR 100 - Training Loss

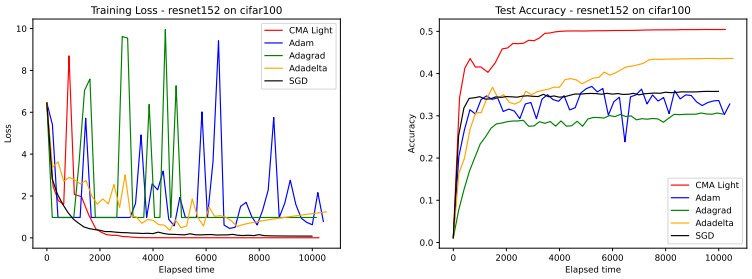


(d) CIFAR 100 - Test Accuracy

Figure A.7: Training loss and test accuracy plot during the training of CMA Light and its competitors in Chapter 5 on the ResNet-101 architecture.

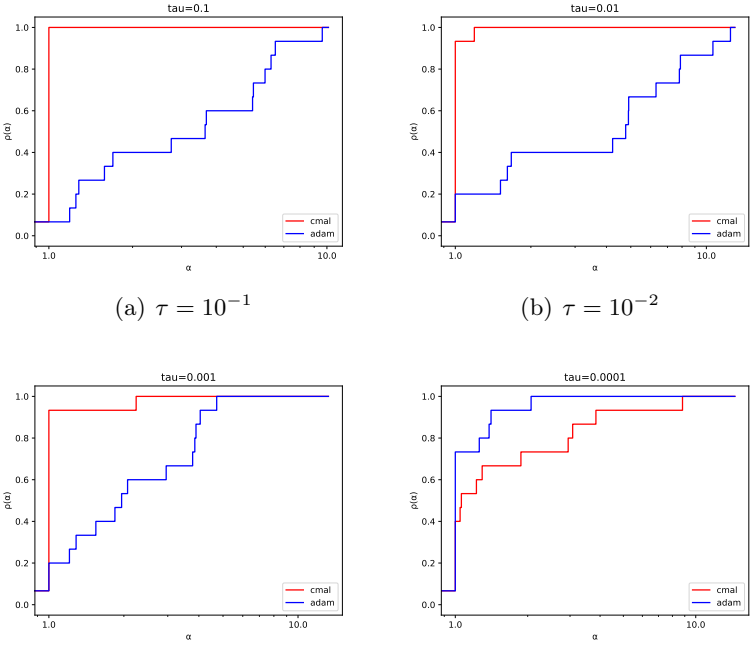


(a) CIFAR 10 - Training Loss (b) CIFAR 10 - Test Accuracy



(c) CIFAR 100 - Training Loss (d) CIFAR 100 - Test Accuracy

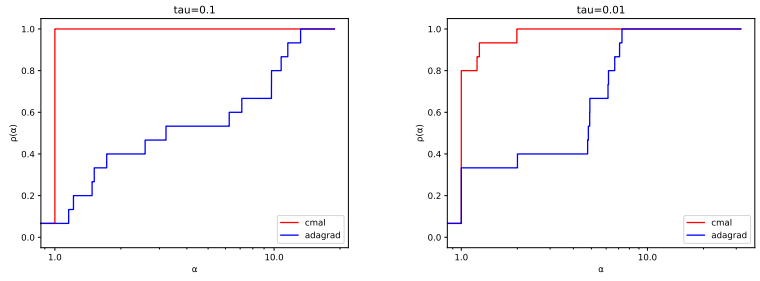
Figure A.8: Training loss and test accuracy plot during the training of CMA Light and its competitors in Chapter 5 on the ResNet-152 architecture.



(a) $\tau = 10^{-1}$ (b) $\tau = 10^{-2}$

(c) $\tau = 10^{-3}$ (d) $\tau = 10^{-4}$

Figure A.9: Performance profile of CMA Light on the training loss against Adam.

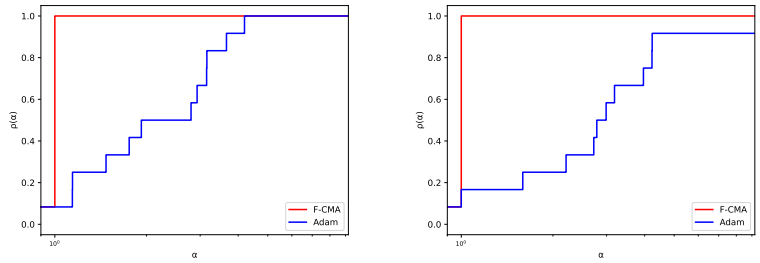


(a) $\tau = 10^{-1}$ (b) $\tau = 10^{-2}$

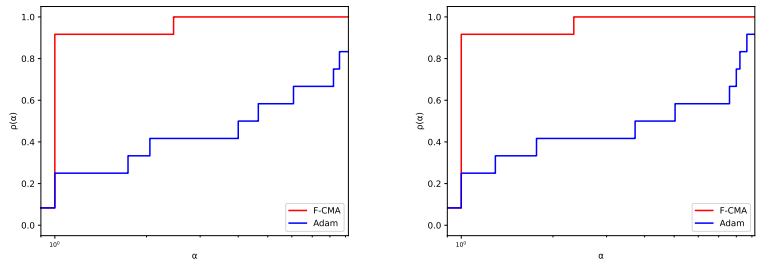
(c) $\tau = 10^{-3}$ (d) $\tau = 10^{-4}$

Figure A.10: Performance profile of CMA Light on the training loss against Adagrad.

A.3 Additional figures and tables from chapter 6 (F-CMA)

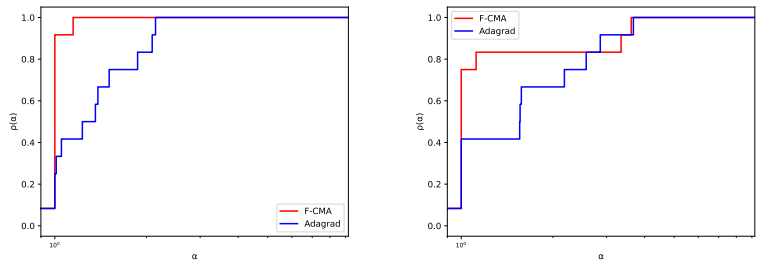


(a) $\tau = 10^{-1}$ (b) $\tau = 10^{-2}$

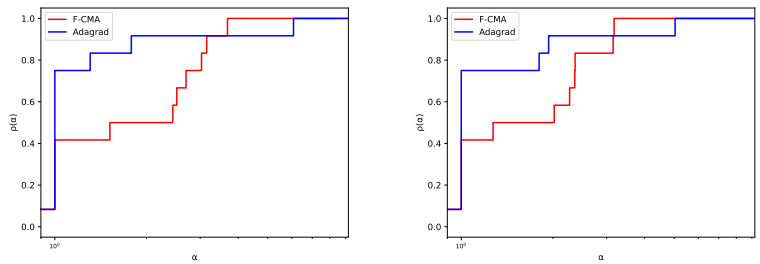


(c) $\tau = 10^{-3}$ (d) $\tau = 10^{-4}$

Figure A.11: Performance profile of F-CMA on the training loss against Adam.

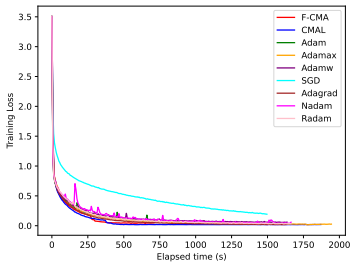


(a) $\tau = 10^{-1}$ (b) $\tau = 10^{-2}$

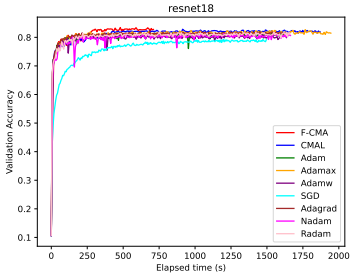


(c) $\tau = 10^{-3}$ (d) $\tau = 10^{-4}$

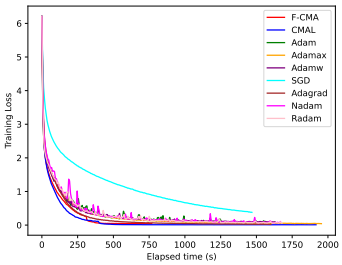
Figure A.12: Performance profile of F-CMA on the training loss against Adagrad.



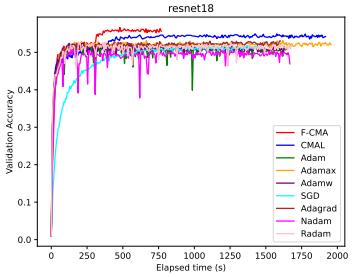
(a) CIFAR 10 - Training Loss



(b) CIFAR 10 - Test Accuracy

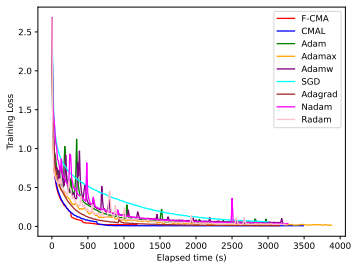


(c) CIFAR 100 - Training Loss

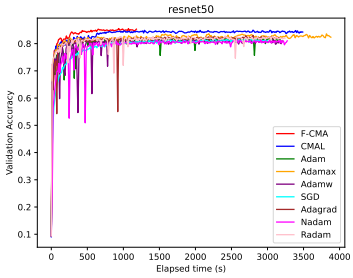


(d) CIFAR 100 - Test Accuracy

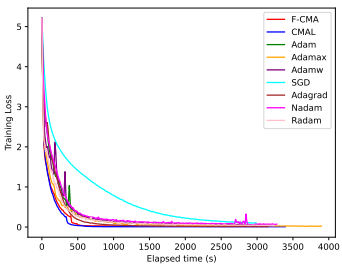
Figure A.13: Training loss and test accuracy plot during the training of F-CMA and its competitors in Chapter 6 on the ResNet-18 architecture.



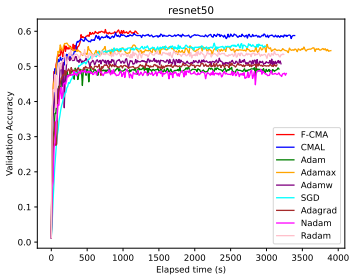
(a) CIFAR 10 - Training Loss



(b) CIFAR 10 - Test Accuracy

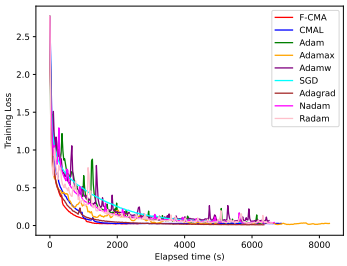


(c) CIFAR 100 - Training Loss

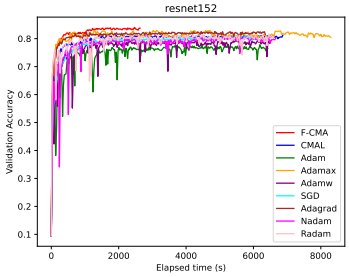


(d) CIFAR 100 - Test Accuracy

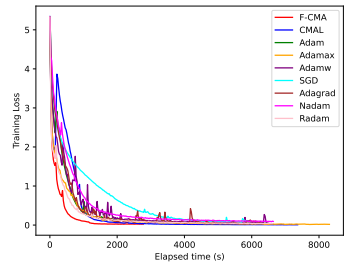
Figure A.14: Training loss and test accuracy plot during the training of F-CMA and its competitors in Chapter 6 on the ResNet-50 architecture.



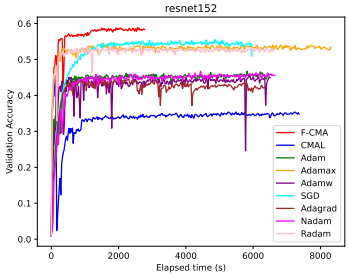
(a) CIFAR 10 - Training Loss



(b) CIFAR 10 - Test Accuracy

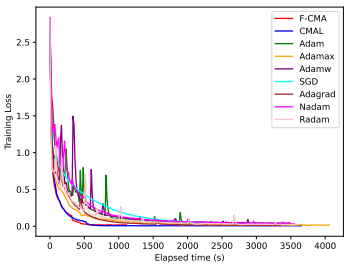


(c) CIFAR 100 - Training Loss

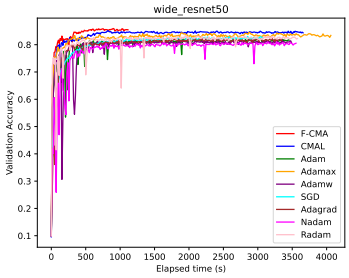


(d) CIFAR 100 - Test Accuracy

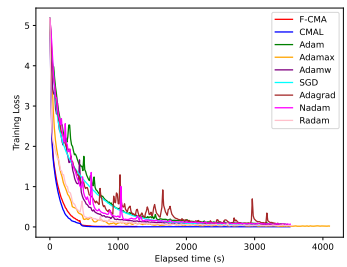
Figure A.15: Training loss and test accuracy plot during the training of F-CMA and its competitors in Chapter 6 on the ResNet-152 architecture.



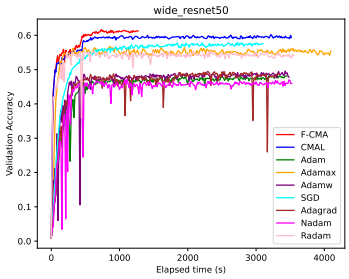
(a) CIFAR 10 - Training Loss



(b) CIFAR 10 - Test Accuracy

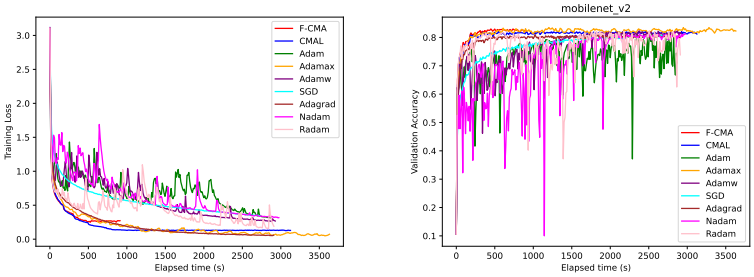


(c) CIFAR 100 - Training Loss

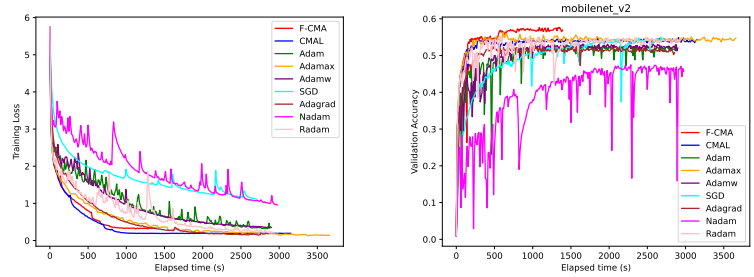


(d) CIFAR 100 - Test Accuracy

Figure A.16: Training loss and test accuracy plot during the training of F-CMA and its competitors in Chapter 6 on the Wide ResNet-50 architecture.

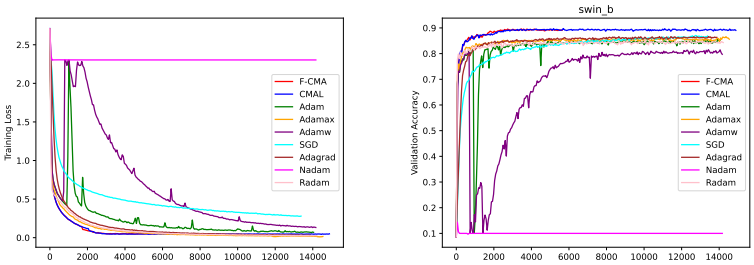


(a) CIFAR 10 - Training Loss (b) CIFAR 10 - Test Accuracy

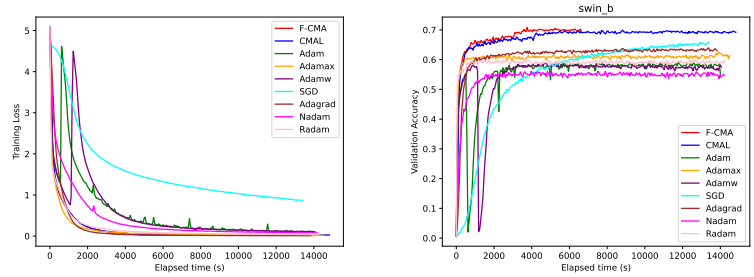


(c) CIFAR 100 - Training Loss (d) CIFAR 100 - Test Accuracy

Figure A.17: Training loss and test accuracy plot during the training of F-CMA and its competitors in Chapter 6 on the Mobilenet V2 architecture.



(a) CIFAR 10 - Training Loss (b) CIFAR 10 - Test Accuracy



(c) CIFAR 100 - Training Loss (d) CIFAR 100 - Test Accuracy

Figure A.18: Training loss and test accuracy plot during the training of F-CMA and its competitors in Chapter 6 on the SwinB architecture.

Bibliography

[1] K. Ahn, C. Yun, and S. Sra. Sgd with shuffling: optimal rates without component convexity and large epoch requirements. *Advances in Neural Information Processing Systems*, 33:17526–17535, 2020.

[2] S. B. Akintoye, L. Han, X. Zhang, H. Chen, and D. Zhang. A hybrid parallelization approach for distributed and scalable deep learning. *IEEE Access*, 10:77950–77961, 2022.

[3] J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, and F. Herrera. Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework. *Journal of Multiple-Valued Logic & Soft Computing*, 17, 2011.

[4] Z. Allen-Zhu and E. Hazan. Variance reduction for faster non-convex optimization. In *International conference on machine learning*, pages 699–707. PMLR, 2016.

[5] Z. Allen-Zhu, Y. Li, and Z. Song. A convergence theory for deep learning via over-parameterization. In *International conference on machine learning*, pages 242–252. PMLR, 2019.

[6] L. Armijo. Minimization of functions having lipschitz continuous first partial derivatives. *Pacific Journal of mathematics*, 16(1):1–3, 1966.

[7] A. Asuncion and D. Newman. Uci machine learning repository, 2007.

[8] M.-F. F. Balcan, M. Khodak, D. Sharma, and A. Talwalkar. Learning-to-learn non-convex piecewise-lipschitz functions. *Advances in Neural Information Processing Systems*, 34:15056–15069, 2021.

[9] Y. Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.

[10] Y. Bengio, I. Goodfellow, and A. Courville. *Deep learning*, volume 1. MIT press Cambridge, MA, USA, 2017.

[11] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[12] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

[13] L. Bernstein, A. Sludds, R. Hamerly, V. Sze, J. Emer, and D. Englund. Freely scalable and reconfigurable optical hardware for deep learning. *Scientific reports*, 11(1):3144, 2021.

[14] D. P. Bertsekas. Incremental gradient, subgradient, and proximal methods for convex optimization: A survey. *Optimization for Machine Learning*, 2010(1-38):3, 2011.

[15] D. P. Bertsekas. Nonlinear programming. In *Nonlinear Programming*. Scientific, Athena, third edition edition, 2016.

[16] D. P. Bertsekas and J. N. Tsitsiklis. Gradient convergence in gradient methods with errors. *SIAM Journal on Optimization*, 10(3):627–642, 2000.

[17] D. P. Bertsekas and J. N. Tsitsiklis. Gradient Convergence in Gradient methods with Errors. *SIAM Journal on Optimization*, 10(3):627–642, 2000.

[18] L. Bethune, T. Boissin, M. Serrurier, F. Mamalet, C. Friedrich, and A. Gonzalez Sanz. Pay attention to your loss: understanding misconceptions about lipschitz neural networks. *Advances in Neural Information Processing Systems*, 35:20077–20091, 2022.

[19] C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[20] D. Blatt, A. O. Hero, and H. Gauchman. A convergent incremental gradient method with a constant step size. *SIAM Journal on Optimization*, 18(1):29–51, 2007.

[21] S. Bock and M. Weiß. A proof of local convergence for the adam optimizer. In *2019 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2019.

[22] R. Bollapragada, R. Byrd, and J. Nocedal. Adaptive sampling strategies for stochastic optimization. *SIAM Journal on Optimization*, 28(4):3312–3343, 2018.

[23] L. Bottou. Curiously fast convergence of some stochastic gradient descent algorithms. In *Proceedings of the symposium on learning and data science, Paris*, volume 8, pages 2624–2633. Citeseer, 2009.

[24] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *in COMPSTAT*, 2010.

[25] L. Bottou. *Stochastic Gradient Descent Tricks*, pages 421–436. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[26] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

[27] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

[28] L. Bouza, A. Bugeau, and L. Lannelongue. How to estimate carbon footprint when training deep learning models? a guide and review. *Environmental Research Communications*, 5(11):115014, 2023.

[29] R.-D. Buhai, Y. Halpern, Y. Kim, A. Risteski, and D. Sontag. Empirical study of the benefits of overparameterization in learning latent variable models. In *International Conference on Machine Learning*, pages 1211–1219. PMLR, 2020.

[30] C. J. Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[31] R. H. Byrd, G. M. Chin, J. Nocedal, and Y. Wu. Sample size selection in optimization methods for machine learning. *Mathematical programming*, 134(1):127–155, 2012.

[32] H. Cai, C. Bai, Y.-W. Tai, and C.-K. Tang. Deep video generation, prediction and completion of human action sequences. In *Proceedings of the European conference on computer vision (ECCV)*, pages 366–382, 2018.

[33] L. Cannelli, F. Facchinei, V. Kungurtsev, and G. Scutari. Asynchronous parallel algorithms for nonconvex optimization. *Mathematical Programming*, 184(1):121–154, 2020.

[34] E. Carrizosa, C. Molero-Río, and D. Romero Morales. Mathematical optimization in classification and regression trees. *Top*, 29(1):5–33, 2021.

[35] C. Cartis and K. Scheinberg. Global convergence rate analysis of unconstrained optimization methods based on probabilistic models. *Mathematical Programming*, 169:337–375, 2018.

[36] C. C. . Causation and Prediction. http://www.causality.inf.ethz.ch/data/SIDO.html, 2008.

[37] R. Chamberlain, M. Powell, C. Lemarechal, and H. Pedersen. The watchdog technique for forcing convergence in algorithms for constrained optimization. *Algorithms for Constrained Minimization of Smooth Nonlinear Functions*, pages 1–17, 1982.

[38] X. Chang, Y. Li, S. Oymak, and C. Thrampoulidis. Provable benefits of overparameterization in model compression: From double descent to pruning neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 6974–6983, 2021.

[39] X. Chen, S. Z. Wu, and M. Hong. Understanding gradient clipping in private sgd: A geometric perspective. *Advances in Neural Information Processing Systems*, 33:13773–13782, 2020.

[40] K. Chowdhary and K. Chowdhary. Natural language processing. *Fundamentals of artificial intelligence*, pages 603–649, 2020.

[41] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

[42] I. Ciocci, C. Coppola, L. Palagi, and L. Papa. Block layer decomposition applied to a watchdog controlled minibatch algorithm. Technical report, DIAG - Sapienza University of Rome, 2024.

[43] C. Coppola, G. Liuzzi, and L. Palagi. Cma light: a novel minibatch algorithm for large-scale non convex finite sum optimization. *arXiv preprint arXiv:2307.15775*, 2023.

[44] C. Coppola, L. Papa, M. Boresta, I. Amerini, and L. Palagi. Tuning parameters of deep neural network training algorithms pays off: a computational study. *TOP*, pages 1–42, 2024.

[45] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20:273–297, 1995.

[46] F.-A. Croitoru, V. Hondru, R. T. Ionescu, and M. Shah. Diffusion models in vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(9):10850–10869, 2023.

[47] G. Damaskinos, R. Guerraoui, R. Patra, M. Taziki, et al. Asynchronous byzantine machine learning (the case of sgd). In *International Conference on Machine Learning*, pages 1145–1154. PMLR, 2018.

[48] S. De, A. Yadav, D. Jacobs, and T. Goldstein. Automated inference with adaptive batches. In *Artificial Intelligence and Statistics*, pages 1504–1513, 2017.

[49] C. M. De Sa. Random reshuffling is not always better. *Advances in Neural Information Processing Systems*, 33:5957–5967, 2020.

[50] A. Défossez, L. Bottou, F. Bach, and N. Usunier. A simple convergence proof of Adam and Adagrad. *arXiv preprint arXiv:2003.02395*, 2020.

[51] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[52] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[53] G. I. Diaz, A. Fokoue-Nkoutche, G. Nannicini, and H. Samulowitz. An effective algorithm for hyperparameter optimization of neural networks. *IBM Journal of Research and Development*, 61(4/5):9–1, 2017.

[54] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.

[55] R. Dorfman, N. Yehya, and K. Y. Levy. Dynamic byzantine-robust learning: Adapting to switching byzantine workers. *arXiv preprint arXiv:2402.02951*, 2024.

[56] T. Dozat. Incorporating nesterov momentum into adam. 2016.

[57] T. Dozat. Incorporating nesterov momentum into Adam. In *ICLR Workshop*, 2016.

[58] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7):2121–2159, 2011.

[59] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011.

[60] J. C. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. In *J. Mach. Learn. Res.*, pages 2121–2159, 2010.

[61] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern classification*. John Wiley & Sons, 2001.

[62] M. Eirinaki, J. Gao, I. Varlamis, and K. Tserpes. Recommender systems for large-scale social networks: A review of challenges and solutions, 2018.

[63] M. Elasri, O. Elharrouss, S. Al-Maadeed, and H. Tairi. Image generation: A review. *Neural Processing Letters*, 54(5):4609–4646, 2022.

[64] W. Fang, D.-J. Han, and C. G. Brinton. Submodel partitioning in hierarchical federated learning: Algorithm design and convergence analysis. In *ICC 2024-IEEE International Conference on Communications*, pages 268–273. IEEE, 2024.

[65] O. Fercoq and P. Richtárik. Accelerated, parallel, and proximal coordinate descent. *SIAM Journal on Optimization*, 25(4):1997–2023, 2015.

[66] Y. Freund and L. Mason. The alternating decision tree learning algorithm. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 124–133, 1999.

[67] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.

[68] M. P. Friedlander and M. Schmidt. Hybrid deterministic-stochastic methods for data fitting. *SIAM Journal on Scientific Computing*, 34(3):A1380–A1405, 2012.

[69] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[70] S. Ghadimi and G. Lan. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM journal on optimization*, 23(4):2341–2368, 2013.

[71] I. Goodfellow, Y. Bengio, and A. Courville. Deep learning. In *Deep Learning*. MIT Press, 2016. urlhttp://www.deeplearningbook.org.

[72] N. Gould and J. Scott. A note on performance profiles for benchmarking software. *ACM Transactions on Mathematical Software (TOMS)*, 43(2):1–5, 2016.

[73] S. Gratton, S. Jerad, and P. L. Toint. Convergence properties of an objective-function-free optimization regularization algorithm, including an complexity bound. *SIAM Journal on Optimization*, 33(3):1621–1646, 2023.

[74] S. Gratton, S. Jerad, and P. L. Toint. Complexity of a class of first-order objective-function-free optimization algorithms. *Optimization Methods and Software*, pages 1–31, 2024.

[75] L. Grippo. A class of unconstrained minimization methods for neural network training. *Optimization Methods and Software*, 4(2):135–150, 1994.

[76] L. Grippo, F. Lampariello, and S. Lucidi. A nonmonotone line search technique for Newton's method. *SIAM journal on Numerical Analysis*, 23(4):707–716, 1986.

[77] L. Grippo, F. Lampariello, and S. Lucidi. Global convergence and stabilization of unconstrained minimization methods without derivatives. *Journal of Optimization Theory and Applications*, 56(3):385–406, 1988.

[78] L. Grippo and M. Sciandrone. Nonmonotone derivative-free methods for nonlinear equations. *Computational Optimization and applications*, 37:297–328, 2007.

[79] L. Grippo and M. Sciandrone. *Introduction to Methods for Nonlinear Optimization*, volume 152. Springer Nature, 2023.

[80] C. Gulcehre, J. Sotelo, M. Moczulski, and Y. Bengio. A robust adaptive stochastic gradient method for deep learning. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 125–132. IEEE, 2017.

[81] M. Gurbuzbalaban, A. Ozdaglar, and P. A. Parrilo. On the convergence rate of incremental aggregated gradient algorithms. *SIAM Journal on Optimization*, 27(2):1035–1048, 2017.

[82] M. Gurbuzbalaban, A. Ozdaglar, and P. A. Parrilo. Convergence rate of incremental gradient and incremental newton methods. *SIAM Journal on Optimization*, 29(4):2542–2565, 2019.

[83] M. Gürbüzbalaban, A. Ozdaglar, and P. A. Parrilo. Why random reshuffling beats stochastic gradient descent. *Mathematical Programming*, 186:49–84, 2021.

[84] S. H. Haji and A. M. Abdulazeez. Comparison of optimization techniques based on gradient descent algorithm: A review. *PalArch's Journal of Archaeology of Egypt/Egyptology*, 18(4):2715–2743, 2021.

[85] K. Han, Y. Wang, H. Chen, X. Chen, J. Guo, Z. Liu, Y. Tang, A. Xiao, C. Xu, Y. Xu, et al. A survey on vision transformer. *IEEE transactions on pattern analysis and machine intelligence*, 45(1):87–110, 2022.

[86] F. Hanzely, P. Richtarik, and L. Xiao. Accelerated bregman proximal gradient methods for relatively smooth convex optimization. *Computational Optimization and Applications*, 79:405–440, 2021.

[87] M. Hardt and T. Ma. Identity matters in deep learning. *arXiv preprint arXiv:1611.04231*, 2016.

[88] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[89] G. Hinton, N. Srivastava, and K. Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8):2, 2012.

[90] S. Hochreiter and J. Schmidhuber. Flat minima. *Neural computation*, 9(1):1–42, 1997.

[91] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[92] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[93] X. Jiang and S. U. Stich. Adaptive sgd with polyak stepsize and line-search: Robust convergence and variance reduction. *Advances in Neural Information Processing Systems*, 36, 2024.

[94] T. Johnson, P. Agrawal, H. Gu, and C. Guestrin. Adascale sgd: A user-friendly algorithm for distributed training. In *International Conference on Machine Learning*, pages 4911–4920. PMLR, 2020.

[95] M. I. Jordan and T. M. Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[96] P. Kairouz, M. R. Diaz, K. Rush, and A. Thakurta. (nearly) dimension independent private erm with adagrad rates. In *Conference on Learning Theory*, pages 2717–2746. PMLR, 2021.

[97] S. P. Karimireddy, L. He, and M. Jaggi. Learning from history for byzantine robust optimization. In *International Conference on Machine Learning*, pages 5311–5319. PMLR, 2021.

[98] A. Kavis, K. Y. Levy, and V. Cevher. High probability bounds for a class of nonconvex algorithms with adagrad stepsize. *arXiv preprint arXiv:2204.02833*, 2022.

[99] K. Kawaguchi. Deep learning without poor local minima. *Advances in neural information processing systems*, 29, 2016.

[100] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016. ICLR 2017.

[101] A. Khaled and P. Richtárik. Better theory for SGD in the nonconvex world. *arXiv preprint arXiv:2002.03329*, 2020.

[102] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah. Transformers in vision: A survey. *ACM computing surveys (CSUR)*, 54(10s):1–41, 2022.

[103] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[104] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[105] D. P. Kingma et al. Variational inference & deep learning. *A New Synthesis. University of Amsterdam*, 2017.

[106] D. P. Kingma, M. Welling, et al. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

[107] D. Kovalev, E. Gasanov, A. Gasnikov, and P. Richtarik. Lower bounds and optimal algorithms for smooth and strongly convex decentralized optimization over time-varying networks. *Advances in Neural Information Processing Systems*, 34:22325–22335, 2021.

[108] D. Kovalev, A. Salim, and P. Richtárik. Optimal and practical algorithms for smooth and strongly convex decentralized optimization. *Advances in Neural Information Processing Systems*, 33:18342–18352, 2020.

[109] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[110] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[111] G. Lan. *First-order and stochastic optimization methods for machine learning*. Springer, New York, 2020.

[112] F. Latorre, P. Rolland, and V. Cevher. Lipschitz constant estimation of neural networks via sparse polynomial optimization. *arXiv preprint arXiv:2004.08688*, 2020.

[113] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[114] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer, 2002.

[115] C. Lemaréchal. A view of line-searches. *Optimization and Optimal Control*, pages 59–78, 1981.

[116] K. Levy. Online to offline conversions, universality and adaptive minibatch sizes. *Advances in Neural Information Processing Systems*, 30, 2017.

[117] K. Levy, A. Kavis, and V. Cevher. Storm+: Fully adaptive sgd with recursive momentum for nonconvex optimization. *Advances in Neural Information Processing Systems*, 34:20571–20582, 2021.

[118] K. Y. Levy. M-sgd: Stable stochastic optimization via a double momentum mechanism. *arXiv preprint arXiv:2304.04172*, 2023.

[119] H. Li, A. Rakhlin, and A. Jadbabaie. Convergence of adam under relaxed assumptions. *Advances in Neural Information Processing Systems*, 36, 2024.

[120] X. Li, A. Milzarek, and J. Qiu. Convergence of random reshuffling under the Kurdyka-Łojasiewicz inequality. *arXiv preprint arXiv:2110.04926*, 2021.

[121] X. Li and F. Orabona. On the convergence of stochastic gradient descent with adaptive stepsizes. In *The 22nd international conference on artificial intelligence and statistics*, pages 983–992. PMLR, 2019.

[122] J. Liang, Y. Fan, K. Zhang, R. Timofte, L. Van Gool, and R. Ranjan. Movideo: Motion-aware video generation with diffusion model. In *European Conference on Computer Vision*, pages 56–74. Springer, 2025.

[123] P. Liang, Y. Tang, X. Zhang, Y. Bai, T. Su, Z. Lai, L. Qiao, and D. Li. A survey on auto-parallelism of large-scale deep learning training. *IEEE Transactions on Parallel and Distributed Systems*, 34(8):2377–2390, 2023.

[124] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403, 2021.

[125] T. Lin, Y. Wang, X. Liu, and X. Qiu. A survey of transformers. *AI open*, 3:111–132, 2022.

[126] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

[127] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.

[128] Y. Liu, S. Liu, Y. Wang, F. Lombardi, and J. Han. A survey of stochastic computing neural networks for machine learning applications. *IEEE Transactions on Neural Networks and Learning Systems*, 32(7):2809–2824, 2020.

[129] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

[130] Z. Liu, T. D. Nguyen, A. Ene, and H. Nguyen. On the convergence of adagrad (norm) on r^ d: Beyond convexity, non-asymptotic rate and acceleration. In *International Conference on Learning Representations*. International Conference on Learning Representations, 2023.

[131] G. Liuzzi, L. Palagi, and R. Seccia. Convergence under lipschitz smoothness of ease-controlled random reshuffling gradient algorithms. *arXiv preprint arXiv:2212.01848*, 2022.

[132] W.-Y. Loh. Classification and regression trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(1):14–23, 2011.

[133] N. Loizou, S. Vaswani, I. H. Laradji, and S. Lacoste-Julien. Stochastic polyak step-size for sgd: An adaptive learning rate for fast convergence. In *International Conference on Artificial Intelligence and Statistics*, pages 1306–1314. PMLR, 2021.

[134] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[135] I. Loshchilov and F. Hutter. Fixing weight decay regularization in adam. 2018.

[136] S. Lucidi and M. Sciandrone. On the global convergence of derivative-free methods for unconstrained optimization. *SIAM Journal on Optimization*, 13(1):97–116, 2002.

[137] Z.-Q. Luo. On the convergence of the lms algorithm with adaptive learning rate for linear feedforward networks. *Neural Computation*, 3(2):226–245, 1991.

[138] C. Ma, L. Wu, and E. Weinan. A qualitative study of the dynamic behavior for adaptive gradient algorithms. In *Mathematical and Scientific Machine Learning*, pages 671–692. PMLR, 2022.

[139] M. Mahsereci and P. Hennig. Probabilistic line searches for stochastic optimization. *Journal of Machine Learning Research*, 18(119):1–59, 2017.

[140] G. Malinovsky, K. Mishchenko, and P. Richtárik. Server-side stepsizes and sampling without replacement provably help in federated optimization. *arXiv preprint arXiv:2201.11066*, 2022.

[141] G. Malinovsky, A. Sailanbayev, and P. Richtárik. Random reshuffling with variance reduction: New analysis and better rates. *arXiv preprint arXiv:2104.09342*, 2021.

[142] J. Martens et al. Deep learning via hessian-free optimization. In *Icml*, volume 27, pages 735–742, 2010.

[143] D. Q. Mayne, E. Polak, and A. Heunis. Solving nonlinear inequalities in a finite number of iterations. *Journal of Optimization Theory and Applications*, 33(2):207–221, 1981.

[144] H. B. McMahan. A survey of algorithms and analysis for adaptive online learning. *Journal of Machine Learning Research*, 18(90):1–50, 2017.

[145] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, S. Chikkerur, D. Liu, M. Wattenberg, A. M. Hrafnkelsson, T. Boulos, and J. Kubica. Ad click prediction: a view from the trenches. *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013.

[146] G. Menghani. Efficient deep learning: A survey on making deep learning models smaller, faster, and better. *ACM Computing Surveys*, 55(12):1–37, 2023.

[147] M. A. Mercioni and S. Holban. P-swish: Activation function with learnable parameters based on swish activation function in deep learning. In *2020 International Symposium on Electronics and Telecommunications (ISETC)*, pages 1–4. IEEE, 2020.

[148] K. Mishchenko, A. Khaled, and P. Richtárik. Random reshuffling: Simple analysis with vast improvements. *Advances in Neural Information Processing Systems*, 33:17309–17320, 2020.

[149] K. Mishchenko, A. Khaled, and P. Richtarik. Random reshuffling: Simple analysis with vast improvements. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 17309–17320. Curran Associates, Inc., 2020.

[150] K. Mishchenko, A. Khaled, and P. Richtárik. Proximal and federated random reshuffling. In *International Conference on Machine Learning*, pages 15718–15749. PMLR, 2022.

[151] A. Mokhtari, M. Gurbuzbalaban, and A. Ribeiro. Surpassing gradient descent provably: A cyclic incremental method with linear convergence rate. *SIAM Journal on Optimization*, 28(2):1420–1447, 2018.

[152] J. J. Moré and S. M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM Journal on Optimization*, 20(1):172–191, 2009.

[153] M. Mutschler and A. Zell. Parabolic approximation line search for dnns. *Advances in Neural Information Processing Systems*, 33:5405–5416, 2020.

[154] M. Mutschler and A. Zell. Empirically explaining sgd from a line search perspective. In *Artificial Neural Networks and Machine Learning–ICANN 2021: 30th International Conference on Artificial Neural Networks, Bratislava, Slovakia, September 14–17, 2021, Proceedings, Part II 30*, pages 459–471. Springer, 2021.

[155] M. V. Narkhede, P. P. Bartakke, and M. S. Sutaone. A review on weight initialization strategies for neural networks. *Artificial intelligence review*, 55(1):291–322, 2022.

[156] R. M. Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 1996.

[157] A. Nedić and D. Bertsekas. Convergence rate of incremental subgradient algorithms. In *Stochastic optimization: algorithms and applications*, pages 223–264. Springer, 2001.

[158] A. Nedic and D. P. Bertsekas. Incremental subgradient methods for nondifferentiable optimization. *SIAM Journal on Optimization*, 12(1):109–138, 2001.

[159] L. M. Nguyen, Q. Tran-Dinh, D. T. Phan, P. H. Nguyen, and M. Van Dijk. A unified convergence analysis for shuffling-type gradient methods. *The Journal of Machine Learning Research*, 22(1):9397–9440, 2021.

[160] J. Nocedal and S. J. Wright. *Numerical optimization*. Springer, New York, 1999.

[161] J. Nocedal and S. J. Wright. Numerical optimization. In *Numerical Optimization*. Springer, New York, NY, USA, second edition, 2006.

[162] S. Pal, E. Ebrahimi, A. Zulfiqar, Y. Fu, V. Zhang, S. Migacz, D. Nellans, and P. Gupta. Optimizing multi-gpu parallelization strategies for deep learning training. *Ieee Micro*, 39(5):91–101, 2019.

[163] L. Palagi. Global optimization issues in deep network regression: an overview. *Journal of Global Optimization*, 73(2):239–277, 2019.

[164] L. Palagi and R. Seccia. On the convergence of a block-coordinate incremental gradient method. *Soft Computing*, 25(19):12615–12626, 2021.

[165] L. Palagi and R. Seccia. On the convergence of a block-coordinate incremental gradient method. *Soft Computing*, pages 1–12, 2021.

[166] L. Papa, P. Russo, I. Amerini, and L. Zhou. A survey on efficient vision transformers: algorithms, techniques, and performance benchmarking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[167] P. Pauli, A. Koch, J. Berberich, P. Kohler, and F. Allgöwer. Training robust neural networks using lipschitz bounds. *IEEE Control Systems Letters*, 6:121–126, 2021.

[168] B. T. Polyak. Introduction to optimization. optimization software. *Inc., Publications Division, New York*, 1, 1987.

[169] S. J. Prince. *Understanding deep learning*. MIT press, 2023.

[170] J. Qian, Y. Wu, B. Zhuang, S. Wang, and J. Xiao. Understanding gradient clipping in incremental gradient methods. In *International Conference on Artificial Intelligence and Statistics*, pages 1504–1512. PMLR, 2021.

[171] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[172] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[173] B. Recht and C. Ré. Parallel stochastic gradient algorithms for large-scale matrix completion. *Mathematical Programming Computation*, 5(2):201–226, 2013.

[174] D. Richards and M. Rabbat. Learning with gradient descent and weakly convex losses. In *International Conference on Artificial Intelligence and Statistics*, pages 1990–1998. PMLR, 2021.

[175] P. Richtárik and M. Takáč. Distributed coordinate descent method for learning with big data. *Journal of Machine Learning Research*, 17(75):1–25, 2016.

[176] H. Robbins and S. Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[177] H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

[178] B. Rokh, A. Azarpeyvand, and A. Khanteymoori. A comprehensive survey on model quantization for deep neural networks in image classification. *ACM Transactions on Intelligent Systems and Technology*, 14(6):1–50, 2023.

[179] O. Ronneberger, P. Fischer, and T. Brox. 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*, pages 234–241. Springer, 2015.

[180] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[181] A. Sadiev, G. Malinovsky, E. Gorbunov, I. Sokolov, A. Khaled, K. Burlachenko, and P. Richtárik. Federated optimization algorithms with random reshuffling and gradient compression. *arXiv preprint arXiv:2206.07021*, 2022.

[182] I. Safran and O. Shamir. How good is sgd with random shuffling? In *Conference on Learning Theory*, pages 3250–3284. PMLR, 2020.

[183] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[184] M. Schmidt, N. Le Roux, and F. Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112, 2017.

[185] R. Seccia. *Block Coordinate Incremental Methods for training Deep Neural Networks*. PhD thesis, Sapienza, University of Rome, 2020.

[186] R. Seccia, C. Coppola, G. Liuzzi, and L. Palagi. Convergence of ease-controlled random reshuffling gradient algorithms under lipschitz smoothness. *arXiv e-prints*, pages arXiv–2212, 2022.

[187] B. Shah and H. Bhavsar. Time complexity in deep learning models. *Procedia Computer Science*, 215:202–210, 2022.

[188] S. Shalev-Shwartz and S. Ben-David. Understanding machine learning: From theory to algorithms. *Cambridge University Press*, 2014.

[189] O. Shamir. Without-replacement sampling for stochastic gradient methods. *Advances in neural information processing systems*, 29, 2016.

[190] P. Sharma, J. Li, and G. Joshi. On improved distributed random reshuffling over networks. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 13211–13215. IEEE, 2024.

[191] N. Shazeer and M. Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pages 4596–4604. PMLR, 2018.

[192] S. Sheikholeslami, M. Meister, T. Wang, A. H. Payberah, V. Vlassov, and J. Dowling. Autoablation: Automated parallel ablation studies for deep learning. In *Proceedings of the 1st Workshop on Machine Learning and Systems*, pages 55–61, 2021.

[193] L. Shen, Y. Sun, Z. Yu, L. Ding, X. Tian, and D. Tao. On efficient training of large-scale deep learning models: A literature review. *arXiv preprint arXiv:2304.03589*, 2023.

[194] M. V. Solodov. Incremental gradient algorithms with stepsizes bounded away from zero. *Computational Optimization and Applications*, 11:23–35, 1998.

[195] L. Su and V. K. Lau. Hierarchical federated learning for hybrid data partitioning across multitype sensors. *IEEE Internet of Things Journal*, 8(13):10922–10939, 2021.

[196] R.-Y. Sun. Optimization for deep learning: An overview. *Journal of the Operations Research Society of China*, 8(2):249–294, 2020.

[197] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

[198] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton. On the importance of initialization and momentum in deep learning. In *ICML*, 2013.

[199] C.-W. Tsai, C.-F. Lai, H.-C. Chao, and A. V. Vasilakos. Big data analytics: a survey. *Journal of Big data*, 2:1–32, 2015.

[200] N. D. Vanli, M. Gurbuzbalaban, and A. Ozdaglar. Global convergence rate of proximal incremental aggregated gradient methods. *SIAM Journal on Optimization*, 28(2):1282–1300, 2018.

[201] V. Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

[202] V. N. Vapnik. An overview of statistical learning theory. *IEEE transactions on neural networks*, 10(5):988–999, 1999.

[203] V. N. Vapnik, V. Vapnik, et al. Statistical learning theory. 1998.

[204] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[205] S. Vaswani, A. Mishkin, I. Laradji, M. Schmidt, G. Gidel, and S. Lacoste-Julien. Painless stochastic gradient: Interpolation, line-search, and convergence rates. *Advances in neural information processing systems*, 32, 2019.

[206] A. Virmaux and K. Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. *Advances in Neural Information Processing Systems*, 31, 2018.

[207] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018(1):7068349, 2018.

[208] R. Ward, X. Wu, and L. Bottou. Adagrad stepsizes: Sharp convergence over nonconvex landscapes. *Journal of Machine Learning Research*, 21(219):1–30, 2020.

[209] Y. Xie, X. Wu, and R. Ward. Linear convergence of adaptive stochastic gradient descent. In *International conference on artificial intelligence and statistics*, pages 1475–1485. PMLR, 2020.

[210] J. Xu, D. J. Hsu, and A. Maleki. Benefits of over-parameterization with em. *Advances in Neural Information Processing Systems*, 31, 2018.

[211] Z. Xu, Y. Chen, K. Vishniakov, Y. Yin, Z. Shen, T. Darrell, L. Liu, and Z. Liu. Weight selection for model initialization. In *The Twelfth International Conference on Learning Representations*, 2023.

[212] J. Yang, X. Li, I. Fatkhullin, and N. He. Two sides of one coin: the limits of untuned sgd and the power of adaptive methods. *Advances in Neural Information Processing Systems*, 36, 2024.

[213] B. Ying, K. Yuan, S. Vlaski, and A. H. Sayed. On the performance of random reshuffling in stochastic learning. In *2017 Information Theory and Applications Workshop (ITA)*, pages 1–5. IEEE, 2017.

[214] T. Yu and H. Zhu. Hyper-parameter optimization: A review of algorithms and applications. *arXiv preprint arXiv:2003.05689*, 2020.

[215] P. Yue, C. Fang, and Z. Lin. On the lower bound of minimizing polyak-łojasiewicz functions. In *The Thirty Sixth Annual Conference on Learning Theory*, pages 2948–2968. PMLR, 2023.

[216] S. Zagoruyko and N. Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[217] M. D. Zeiler. Adadelta: An adaptive learning rate method. *ArXiv*, abs/1212.5701, 2012.

[218] Y. Zhang, C. Chen, N. Shi, R. Sun, and Z.-Q. Luo. Adam can converge without any modification on update rules. *Advances in neural information processing systems*, 35:28386–28399, 2022.

[219] Y. Zhang and Z.-H. Huang. A nonmonotone smoothing-type algorithm for solving a system of equalities and inequalities. *Journal of Computational and Applied Mathematics*, 233(9):2312–2321, 2010.

[220] Y. Zhou, B. Karimi, J. Yu, Z. Xu, and P. Li. Towards better generalization of adaptive gradient methods. *Advances in Neural Information Processing Systems*, 33:810–821, 2020.

[221] B. Zhuang, J. Liu, Z. Pan, H. He, Y. Weng, and C. Shen. A survey on efficient training of transformers. In E. Elkind, editor, *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 6823–6831. International Joint Conferences on Artificial Intelligence Organization, 8 2023. Survey Track.