University of Modena and Reggio Emilia

DEPARTMENT OF PHYSICS, INFORMATICS AND MATHEMATICS

Ph. D. degree in Mathematics

XXXIII Cycle

In convenzione con l'Università degli Studi di Ferrara e l'Università degli Studi di Parma

PhD Thesis

Hyperparameters setting in Stochastic Optimisation Methods

Thesis Advisor:
Professor Luca Zanni

Reserch supervisor:
Professor Valeria Ruggiero

Candidate:
Giorgia Franchini

Coordinatore del corso di dottorato: Professor Cristian Giardinà

Academic Year 2019–2020

To Angela and Romeo.

Contents

Abstract

In the context of deep learning, the computational more expensive phase is the full training of the learning algorithm. Indeed the design of a new learning algorithm requires an extensive numerical investigation with the execution of a significant number of experimental trials. A fundamental aspect in designing a suitable learning algorithm is the selection of the hyperparameters (parameters that are not trained during the learning process), in a static or adaptive way. The aim of this thesis is to investigate the hyperparameters selection strategies on which standard machine learning algorithms are designed. In particular, we are interested in the different techniques to select the parameters related to the stochastic gradient methods used for training the machine learning methodologies. The main purposes that motivate this study are the improvement of the accuracy (or other metrics suitable for evaluating the inspected methodology) and the acceleration of the convergence rate of the iterative optimisation schemes. To achieve these purposes, the analysis has mainly focused on the choice of the fundamental hyperparameters in the stochastic gradient methods: the steplength, the mini-batch size and the potential adoption of variance reduction techniques. In a first approach we considered separately the choice of steplength and mini-batch size; then, we presented a technique that combines the two choices. About the steplength selection, we propose to tailor for the stochastic gradient iteration the steplength selection adopted in the full-gradient method known as Limited Memory Steepest Descent method. This strategy, based on the Ritz-like values of a suitable matrix, enables to give a local estimate of the inverse of the local Lipschitz parameter. Regarding the mini-batch size the idea is to increasing dynamically, in an adaptive manner (based on suitable validation tests), this size. The experiments show that this training strategy is more efficient (in terms of time and costs) compared with the approaches available in literature. We combine the two parameter choices (steplength and mini-batch size) in an adaptive scheme without introducing line search techniques, while the possible increase of the size of the sub-sample used to compute the stochastic gradient enables to control the variance of this direc-

tion. In the second part of the thesis, we introduce an Automatic Machine Learning (AutoML) technique to set these hyperparameters. In particular, we propose a low-cost strategy to predict the accuracy of the learning algorithm, based only on its initial behavior. The initial and final accuracies observed during this beforehand process are stored in a database and used as train set of a Support Vector Machines learning algorithm. The aim is to predict the accuracy of a learning methodology, given its accuracy on the initial iterations of its learning process. In other word, by a probabilistic exploration of the hyperparameter space, we are able to find the setting providing optimal accuracies at a quite low cost. An extensive numerical experimentation was carried out involving convex and non-convex functions (in particular Convolutional Neural Networks). For the numerical experiments several datasets well known in literature have been used, for different problems such as: classification, segmentation, regression. Finally, a computational study is carried out to extend the proposed approaches to other methods, such as: Momentum, AdaM, SVRG. In conclusion, the contribution of the thesis consists in providing useful ideas about an effective and inexpensive selection of the hyperparameters in the class of the stochastic gradient methods.

Chapter 1

Introduction

Machine Learning (ML) and Deep Neural Networks (DNN) are pervasive in the domain of surveillance, health-care, autonomous machinery, and vehicles. Examples can be found in EU-founded H2020 projects on automotive and smart-cities, such as CLASS (https://class-project.eu/partners) and PRYSTINE (https://www.ecsel.eu/projects/prystine), which are relying on ML and DNN technologies for solving several tasks, such as object-detection for obstacle avoidance and emergency-braking, vehicle-to-vehicle tracking, monitoring and data-fusion. However, the process of designing an efficient and accurate ML methodology for a specific task is power and time-consuming and traditionally requiring direct human intervention. Moreover, the same ML methodology may lead to substantially different accuracies, latencies, and energy efficiencies depending on the hardware target used for the deployment [1], or just changing the data-types used.
In view of these preliminary considerations, this thesis aims to analyse the most expensive phase in terms of power and time of the above cited methodologies: the training phase. Whether you are in a convex (ML) or non-convex (DNN) context, it is certainly crucial to set the hyperparameters connected to the optimiser correctly. We define hyperparameters of a learning method as those parameters that are not trained during the learning process but they are set a-priori as input data. In this work, whenever we talk about hyperparameters connected to the optimiser we refer to: mini-batch size, steplength and type of optimiser.
In the initial part of this work the problem is formalised and the state of the art of stochastic methods is briefly recalled. We report also convergence results in the case of convex and non-convex functions, for fixed and diminishing steplength. This is followed by a description of some DNN methodologies and related challenges, in particular that of the automatic setting of hyperparameters.

Starting with the simplest network element, the perceptron, the concept of a multi-layer network is formalised and the most common layers in imaging are presented: convolutional layer and pooling layer. Particular attention is given to backpropagation, being the operation where the main tool used is stochastic optimisation.
In the literature there are different philosophies to approach the problem of setting hyperparameters. There are static rules, i.e. rules that do not depend on the training phase, and dynamic rules, which only operate under certain conditions connected to the course of the training phase. Another approach to tackle this problem, is focus on automated machine learning (AutoML) and Neural Architecture Search (NAS). AutoML and NAS try to tune, exploiting machine-learning techniques, the ANN topologies itself mainly focusing on accuracy improvement.
This thesis focuses on the choice of hyperparameters connected to the stochastic optimiser used: steplength, mini-batch size and optimiser type (with or without variance reduction). First of all, the mini-batch size is taken into account independently of the other hyperparameters. Assuming that one is often forced with very large and redundant datasets, the idea is to dynamically increase the size of the considered sample. Like the ANN themselves, the inspiration here is biological: curiosity to improve the accuracy. In this case, the test to decide whether to increase the sample size is a dynamic one and is used also as stopping rule.
Furthermore, a crucial remark is that the selection of a suitable steplength can be strictly related to the mini-batch size. Starting from an idea used in the deterministic field to set steplength [20], this is tailored for the stochastic model. In addition to an adaptive rule to set the steplength, the basic iteration provides a test that determines whether the sample size has to be increased.
Finally, all three hyperparameters mentioned are taken into account. In this case, the philosophy is that of AutoML. In particular, we will use a classical ML methodology to set the hyperparameters of a Deep Learning methodology. The numerical experimentation shows promising results, which will be the subject of future investigation.
This thesis, in conjunction with the introduction and conclusions, is essentially structured in four sections. In the first section we present a survey on the state of the art. In the second section we describe a technique for dynamic adjustment of the mini-batch size, using the validation set. In the third section, we propose a technique of steplength selection, combined with the adaptive increase of sample size. In the fourth section, we detail the idea of using the Support Vector Regression for the search of optimal hyperparameters of a learning method. All the sections are equipped, besides

the explanation of the algorithms, with an exhaustive number of numerical experiments, aimed to evaluate the effectiveness of the proposed approaches. At the end, in the final section, in addition to the conclusions, the current directions of research that we are pursuing in order to expand and complete the work carried out, are illustrated.

Chapter 2

State of the art

2.1 Statement of the problem

Let consider an unconstrained problem with the following form:

$$\min_{x \in \mathbb{R}^d} F(x) \equiv \mathbb{E}[f(x, \xi)], \tag{2.1}$$

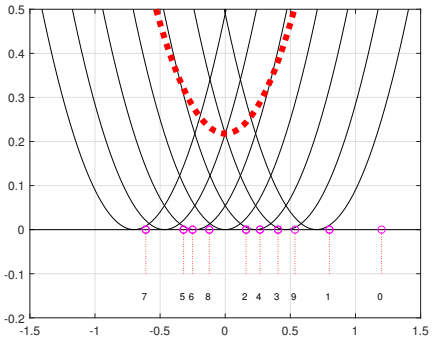
where ξ is a random vector, $f : \mathbb{R}^d \times \xi \to \mathbb{R}$ and the expectation $\mathbb{E}$ is respect to ξ in the probability space $(\Xi, \mathcal{F}, \mathcal{P})$. It is assumed the function f is either known analytically or it is assigned by a black box oracle with a predetermined accuracy. The problem (2.1) includes the cases, often arising in engineering applications, where $F(x)$ has an integral representation with respect to a function of ξ (see [56]); furthermore, machine learning applications lead to optimisation problems of the form (2.1), where F is the expected risk of misclassification over all the possible input-output pairs. In this case $\xi = (u, v)$ is the set of all input-output pairs, the probability distribution P is the true relationship between inputs and outputs and $f = f(\ell(x, u), v)$ is a cost function measuring the distance between a prediction function $\ell(x, u)$ for the u input and the true output v.
In practice, since the probability distribution of ξ is unknown, having available only a sample of data (maybe also a large sample of data) but not their distribution, the solution of a problem involving an estimate of the target function $F(x)$ is searched for. The approximation of this quantity that is most commonly used is Sample Average Approximation, defined as

$$\min_{x \in \mathbb{R}^d} F_n(x) \equiv F_n(x, \xi^{(n)}), \tag{2.2}$$

where the objective function is the Empirical Risk

$$F_n(x, \xi^{(n)}) = \frac{1}{n} \sum_{i=1}^{n} f(x, \xi_i^{(n)}) = \frac{1}{n} \sum_{i=1}^{n} f_i(x), \tag{2.3}$$

based on a random sample $\xi^{(n)} = \{\xi_1^{(n)}, ..., \xi_n^{(n)}\}$ of size n of the variable ξ. In the machine learning context, each $f_i(x) \equiv f(x, \xi_i^{(n)})$ denotes the loss function related to the instance $\xi_i^{(n)} = (u_i^{(n)}, v_i^{(n)})$ of the training set. In the big data framework, since n can be a very large number, it is prohibitively expensive to compute all the terms of the objective function $F_n(x, \xi^{(n)})$, its gradient or its Hessian matrix. On the other hand, often the whole dataset may be too large to be completely stored in memory; in other cases it may not be convenient, compared to the computation time, to use all the available sample, also in view of the redundancy of data.
Finally, in online learning contexts, where the dataset is not available from the beginning in its completeness but is acquired during the learning process, it is impossible to work with $F_n(x, \xi^{(n)})$. In all these cases, the minimization problem is faced by exploiting stochastic approximations of the gradient that lead to the use of Stochastic Gradient (SG) methods.
Another motivation in support of SG methods could be found in a simple example from [6], where the objective function is the sum of some convex univariate quadratic functions f_i with minima distributed in $[-1, 1]$, so that the minimisation of F_n is $x_n^* = 0$.



When $x^{(0)} \gg 1$, SG will, with certainty, move to the left.
Even if a subsequent iterate lies slightly to the left of the minimum of the rightmost quadratic, it is likely (but not certain) that SG will continue moving to the left. However, as iterates approach x_n^*, the algorithm enters a *region of confusion.* The progress will slow significantly.
Starting from the assumption that the function to be minimised is presented as the sum of elements, which are independent of each other, a well known idea in literature is to consider only some of the elements of the dataset at each iteration of an optimisation method. In the following, without claiming to be exhaustive, we recall the most significant methods for solving the problem (2.2)-(2.3).

2.2 Stochastic gradient methods

A known strategy to address the problem (2.1) or its approximation (2.2)-(2.3) is the Stochastic Gradient (SG) method and its variants, which only require the gradient of one or a few terms of $F_n(x)$ per iteration. In this way the computational cost of the single iterate is low compared to the version in which the whole dataset is managed at each iteration.

As can be seen in Algorithm 1, the basic iteration of the SG method can be

Algorithm 1 Stochastic Gradient (SG) Method

Choose an initial iterate $x^{(0)}$.
for $k = 1, 2, \ldots$ **do**
  Generate a realisation of the random variable ξ_k.
  Compute a stochastic vector $g(x^{(k)}, \xi^{(n_k)})$.
  Choose a learning rate $\alpha_k > 0$.
  Set the new iterate as $x^{(k+1)} \leftarrow x^{(k)} - \alpha_k g(x^{(k)}, \xi^{(n_k)})$.
end for

written as

$$x^{(k+1)} = x^{(k)} - \alpha_k g(x^{(k)}, \xi^{(n_k)}), \tag{2.4}$$

where $\xi^{(n_k)}$ denotes a set of n_k realisations of the random variable ξ, randomly chosen from the sample data $\xi^{(n)}$, $g(x^{(k)}, \xi^{(n_k)})$ is the stochastic gradient vector at the current iterate $x^{(k)}$ and α_k is a positive steplength, known also as learning rate. The main strategies for the choices of $\xi^{(n_k)}$ give rise to the standard SG method, when $n_k = 1$ for all k, and its mini-batch version, for $n_k > 1$. In particular, given a randomly chosen subset $S_k \subset \{1, ..., n\}$ of $|S_k| = n_k$ indices, $n_k \geq 1$, and a sub-sample of the training set $\xi^{(n_k)} = \{\xi_i^{(n_k)}\}_{i \in S_k}$, the stochastic gradient is defined as

$$g_k^{(n_k)} \equiv g(x^{(k)}, \xi^{(n_k)}) = \frac{1}{n_k} \sum_{i \in S_k} \nabla f_i(x^{(k)}). \tag{2.5}$$

2.2.1 Convergence results

For the sake of completeness and to justify some of the strategies undertaken, we report the main results of the convergence of SG (for details, see the survey [9]. In a first part we report assumptions and some fundamental lemmas, then we recall the convergence theorems and the related proofs. We observe that the convergence results hold both for function $F(x)$ and its approximation $F_n(x)$. The only difference is that in the first case the sample

is selected in according to a probability distribution, in the second case it is selected uniformly from a discrete training set.

The convergence theorems are basically four: for strongly convex functions with constant stelength, for strongly convex functions with decreasing stelength, for non-convex functions with constant stelength and for non-convex functions with decreasing stelength.

Assumption 1 (Lipschitz-continuous gradient of the pbjective function)**.** The objective function $F : \mathbb{R}^d \to \mathbb{R}$, is continuously differentiable and the gradient function of F, namely $\nabla F : \mathbb{R}^d \to \mathbb{R}^d$, is Lipschitz continuous with Lipschitz constant $L > 0$, i.e.:

$$\| \nabla F(x) - \nabla F(\bar{x} \|_2 \leq L \| x - \bar{x} \|_2 \text{ for all } x, \bar{x} \in \mathbb{R}^d.$$

From the assumption we can obtain the first lemma.

Lemma 1. Under Assumption 1, the iterates of SG satisfy the following inequality for all $k \in \mathbb{N}$:

$$\begin{aligned} &\mathbb{E}[F(x^{(k+1)})] - F(x^{(k)}) \\ &\leq -\alpha_k \nabla F(x^{(k)})^T \mathbb{E}[g(x^{(k)}, \xi^{(k)})] + \tfrac{1}{2}\alpha_k^2 L \mathbb{E}[\| g(x^{(k)}, \xi^{(k)}) \|_2^2]. \end{aligned} \tag{2.6}$$

Proof. By Assumption 1, the iterates generated by SG satisfy

$$\begin{aligned} F(x^{(k+1)}) - F(x^{(k)}) &\leq \nabla F(x^{(k)})^T (x^{(k+1)} - x^{(k)}) + \frac{1}{2} L \| x^{(k+1)} - x^{(k)} \|_2^2 \\ &= -\alpha_k \nabla F(x^{(k)})^T g(x^{(k)}, \xi^{(k)}) + \frac{1}{2}\alpha_k^2 L \| g(x^{(k)}, \xi^{(k)}) \|_2^2 . \end{aligned}$$

Taking expectations in these inequalities with respect to the distribution of $\xi^{(k)}$, and noting that $x^{(k+1)}$ but not $x^{(k)}$ depends on $\xi^{(k)}$, we obtain the desired bound. □

Therefore we introduce some assumptions on the first and second moments of the stochastic process.

Assumption 2 (Bounds for the first and second moments)**.** The objective function and SG method satisfy the following conditions.

1. The sequence of iterates $\{x^{(k)}\}$ is contained in an open set over which F is bounded below by a scalar F_{inf}.

2. There exist scalars $\mu_G \geq \mu > 0$ such that, for all $k \in \mathbb{N}$,

$$\nabla F(x^{(k)})^T \mathbb{E}[g(x^{(k)}, \xi^{(k)})] \geq \mu \| \nabla F(x^{(k)}) \|_2^2 \text{ and} \tag{2.7}$$

$$\| \mathbb{E}_{\xi^{(k)}}[g(x^{(k)}, \xi^{(k)})] \|_2 \leq \mu_G \| \nabla F(x^{(k)}) \|_2 . \tag{2.8}$$

3. There exist scalars $M \geq 0$ and $M_V \geq 0$ such that, for all $k \in \mathbb{N}$,

$$\mathbb{V}[g(x^{(k)}, \xi^{(k)})] \leq M + M_V \parallel \nabla F(x^{(k)}) \parallel_2^2 . \tag{2.9}$$

Consequently, Assumption 2, combined with the variance definition, requires that the second moment of $g(x^{(k)}, \xi^{(k)})$ satisfies

$$\mathbb{E}[\parallel g(x^{(k)}, \xi^{(k)}) \parallel_2^2] \leq M + M_G \parallel \nabla F(x^{(k)}) \parallel_2^2 \tag{2.10}$$

with $M_G := M_V + \mu_G^2 \geq \mu^2 > 0$.
We are now ready to state the following lemma.

Lemma 2. Under Assumptions 1 and 2, the iterates of SG satisfy the following inequalities for all $k \in \mathbb{N}$:

$$\mathbb{E}[F(x^{(k+1)})] - F(x^{(k)}) \tag{2.11}$$

$$\leq -\mu\alpha_k \parallel \nabla F(x^{(k+1)}) \parallel_2^2 + \frac{1}{2}\alpha_k^2 L \mathbb{E}[\parallel g(x^{(k)}, \xi^{(k)}) \parallel_2^2] \tag{2.12}$$

$$\leq -(\mu - \frac{1}{2}\alpha_k L M_G)\alpha_k \parallel \nabla F(x^{(k)}) \parallel_2^2 + \frac{1}{2}\alpha_k^2 L M. \tag{2.13}$$

Proof. By Lemma 1 and (2.7), it follows that

$$\begin{aligned}
&\mathbb{E}[F(x^{(k+1)})] - F(x^{(k)}) \\
&\leq -\alpha_k \nabla F(x^{(k)})^T \mathbb{E}[g(x^{(k)}, \xi^{(k)})] + \frac{1}{2}\alpha_k^2 L \mathbb{E}[\parallel g(x^{(k)}, \xi^{(k)}) \parallel_2^2] \\
&\leq -\mu\alpha_k \parallel \nabla F(x^{(k)}) \parallel_2^2 + \frac{1}{2}\alpha_k^2 L \mathbb{E}[\parallel g(x^{(k)}, \xi^{(k)}) \parallel_2^2]
\end{aligned}$$

which is the first part.
By (2.10) it follows that:

$$\begin{aligned}
&\mathbb{E}[F(x^{(k+1)})] - F(x^{(k)}) \\
&\leq -\mu\alpha_k \parallel \nabla F(x^{(k)}) \parallel_2^2 + \frac{1}{2}\alpha_k^2 L (M + M_G \parallel \nabla F(x^{(k)}) \parallel_2^2) \\
&= \parallel \nabla F(x^{(k)}) \parallel_2^2 \alpha_k(-\mu + \frac{1}{2}\alpha_k L M_G) + \frac{1}{2}\alpha_k^2 L M
\end{aligned}$$

which is the second part. □

Another assumption on F could be the strong convexity:

Assumption 3 (Strong convexity)**.** The objective function $F : \mathbb{R}^d \to \mathbb{R}$ is strongly convex, i. e., there exists a constant $c > 0$ such that, for all $\bar{x}, x \in \mathbb{R}^d$

$$F(\bar{x}) \geq F(x) + \nabla F(x)^T(\bar{x} - x) + \frac{1}{2}c \parallel \bar{x} - x \parallel_2^2 . \tag{2.14}$$

Under this assumption,F has a unique minimum, denoted as $x_* \in \mathbb{R}^d$ with $F_* := F(x_*) = F_{inf}$.

A useful result is that, under Assumption 3, one can bound the optimality gap at a given point in terms of the squared l_2-norm of the gradient of the objective at this point:

$$2c(F(x) - F_*) \leq \| \nabla F(x) \|_2^2 \text{ for all } x \in \mathbb{R}^d. \tag{2.15}$$

Hence, we can state the first convergence theorem, for strongly convex objective function with fixed steplength.

Theorem 3 (Strongly Convex Objective, Fixed Steplength)**.** *Under Assumptions 1, 2, and 3, suppose that SG method is run with a fixed steplength $\alpha_k = \bar{\alpha}$ for all $k \in \mathbb{N}$, satisfying*

$$0 < \bar{\alpha} \leq \frac{\mu}{LM_G}. \tag{2.16}$$

Then, the expected optimality gap satisfies the following inequality for all $k \in \mathbb{N}$:

$$\begin{aligned} &\mathbb{E}[F(x^{(k)}) - F_*] \leq \\ &\tfrac{\bar{\alpha}LM}{2c\mu} + (1 - \bar{\alpha}c\mu)^{k-1} \left(F(x^{(1)}) - F_* - \tfrac{\bar{\alpha}LM}{2c\mu} \right) \xrightarrow{k\to\infty} \tfrac{\bar{\alpha}LM}{2c\mu}. \end{aligned} \tag{2.17}$$

Proof. Using Lemma 2 with $\alpha_k = \bar{\alpha}$,(2.15) and (2.16), we have for all $k \in \mathbb{N}$ that

$$\begin{aligned} \mathbb{E}[F(x^{(k+1)})] - F(x^{(k)}) &\leq -(\mu - \frac{1}{2}\bar{\alpha}LM_G)\bar{\alpha} \| \nabla F(x^{(k)}) \|_2^2 + \frac{1}{2}\bar{\alpha}^2 LM \\ &\leq -\frac{1}{2}\bar{\alpha}\mu \| \nabla F(x^{(k)}) \|_2^2 + \frac{1}{2}\bar{\alpha}^2 LM \\ &\leq -\bar{\alpha}c\mu(F(x^{(k)}) - F_*) + \frac{1}{2}\bar{\alpha}^2 LM. \end{aligned}$$

Subtracting F_* from both sides, taking total expectations, this yields

$$\mathbb{E}[F(x^{(k+1)}) - F_*] \leq \mathbb{E}[F(x^{(k)}) - F_*](-\bar{\alpha}c\mu + 1) + \frac{1}{2}\bar{\alpha}^2 LM$$

Subtracting $\frac{\bar{\alpha}LM}{2c\mu}$ from both sides, one obtains

$$\begin{aligned} &\mathbb{E}[F(x^{(k+1)}) - F_*] - \frac{\bar{\alpha}LM}{2c\mu} \\ &\leq (1 - \bar{\alpha}c\mu)\mathbb{E}[F(x^{(k)}) - F_*] + \frac{\bar{\alpha}^2 LM}{2} - \frac{\bar{\alpha}LM}{2c\mu} \\ &\leq (1 - \bar{\alpha}c\mu) \left(\mathbb{E}[F(x^{(k)}) - F_*] - \frac{\bar{\alpha}LM}{2c\mu} \right). \end{aligned}$$

Observe that last inequality is a contraction since, by (2.16) and (2.10),

$$0 < \bar{\alpha} c\mu \leq \frac{c\mu^2}{LM_G} \leq \frac{c\mu^2}{L\mu^2} = \frac{c}{L} \leq 1. \tag{2.18}$$

The result thus follows by applying the contraction repeatedly through iteration $k \in \mathbb{N}$. □

We now move towards the convergence of the method with decreasing steplength, always in the case of a strongly convex function.

Assumption 4. Suppose that SG method is run with a steplength sequence such that, for all $k \in \mathbb{N}$,

$$\sum_{k=1}^{\infty} \alpha_k = \infty, \quad \sum_{k=1}^{\infty} \alpha_k^2 < \infty. \tag{2.19}$$

Theorem 4 (Strongly Convex Objective, Diminishing Steplengths)**.** *Under Assumptions 1, 2, and 3, suppose that SG method is run with a steplength sequence such that, for all $k \in \mathbb{N}$,*

$$\alpha_k = \frac{\beta}{\gamma + k} \text{ for some } \beta > \frac{1}{c\mu} \text{ and } \gamma > 0 \text{ such that } \alpha_1 \leq \frac{\mu}{LM_G}. \tag{2.20}$$

Then, for all $k \in \mathbb{N}$, the expected optimality gap satisfies

$$\mathbb{E}[F(x^{(k)}) - F_*] \leq \frac{\nu}{\gamma + k}, \tag{2.21}$$

where

$$\nu := \max\left\{ \frac{\beta^2 LM}{2(\beta c\mu - 1)} \, , \, (\gamma + 1)(F(x^{(1)}) - F_*) \right\}. \tag{2.22}$$

Proof. By (2.20), the inequality $\alpha_k LM_G \leq \alpha_1 LM_G \leq \mu$ holds for all $k \in \mathbb{N}$. Hence, along with (2.15), Lemma 2 and (2.15), one has for all $k \in \mathbb{N}$ that

$$\begin{aligned}
&\mathbb{E}[F(x^{(k+1)})] - F(x^{(k)}) \\
&\leq -(\mu - \frac{1}{2}\alpha_k LM_G)\alpha_k \parallel \nabla F(x^{(k)}) \parallel_2^2 + \frac{1}{2}\alpha_k^2 LM \\
&\leq -\frac{1}{2}\alpha_k \mu \parallel \nabla F(x^{(k)}) \parallel_2^2 + \frac{1}{2}\alpha_k^2 LM \\
&\leq -\alpha_k c\mu (F(x^{(k)}) - F(x_*)) + \frac{1}{2}\alpha_k^2 LM.
\end{aligned}$$

Subtracting F_* from both sides, taking total expectations, and rearranging, this yields

$$\mathbb{E}[F(x^{(k+1)}) - F_*] \leq (1 - \alpha_k c\mu)\mathbb{E}[F(x^{(k)}) - F_*] + \frac{1}{2}\alpha_k^2 LM. \tag{2.23}$$

We now prove (2.21) by induction.
First, the definition of ν ensures that the inequality (2.21) holds for $k = 1$. Then, assuming (2.21) holds for some $k \geq 1$, it follows from (2.23) that

$$\begin{aligned}
\mathbb{E}[F(x^{(k+1)}) - F_*] &\leq \left(1 - \frac{\beta c\mu}{\hat{k}}\right)\frac{\nu}{\hat{k}} + \frac{\beta^2 LM}{2\hat{k}^2} \quad (\text{ where } \hat{k} := \gamma + k) \\
&= \left(\frac{\hat{k} - \beta c\mu}{\hat{k}^2}\right)\nu + \frac{\beta^2 LM}{2\hat{k}^2} \\
&= \left(\frac{\hat{k} - 1}{\hat{k}^2}\right)\nu - \left(\frac{\beta c\nu - 1}{\hat{k}^2}\right)\nu + \frac{\beta^2 LM}{2\hat{k}^2} \leq \frac{\nu}{\hat{k} + 1},
\end{aligned}$$

where the last inequality follows because $\hat{k}^2 \geq (\hat{k}+1)(\hat{k}-1)$ and $-\left(\frac{\beta c\nu - 1}{\hat{k}^2}\right)\nu + \frac{\beta^2 LM}{2\hat{k}^2}$ is nonpositive by the definition of ν. □

From hereafter we present similar results for non-convex functions.

Theorem 5 (Non convex Objective, Fixed Steplength)**.** *Under Assumptions 1 and 2, suppose that SG method is run with a fixed steplength $\alpha_k = \bar{\alpha}$ for all $k \in \mathbb{N}$, satisfying*

$$0 < \bar{\alpha} < \frac{\mu}{LM_G}. \tag{2.24}$$

Then, the expected sum-of-squares and average-squared gradients of F corresponding to the SG iterates satisfy the following inequalities for all $K \in \mathbb{N}$:

$$\mathbb{E}\left[\sum_{k=1}^{K} \| \nabla F(x^{(k)}) \|_2^2\right] \leq \frac{K\bar{\alpha}LM}{\mu} + \frac{2(F(x^{(1)}) - F_{inf})}{\mu\bar{\alpha}} \tag{2.25}$$

$$\mathbb{E}\left[\frac{1}{K}\sum_{k=1}^{K} \| \nabla F(x^{(k)}) \|_2^2\right] \leq \frac{\bar{\alpha}LM}{\mu} + \frac{2(F(x^{(1)}) - F_{inf})}{K\mu\bar{\alpha}} \xrightarrow{K\to\infty} \frac{\bar{\alpha}LM}{\mu}. \tag{2.26}$$

Proof. Taking the total expectation of (2.11), in view of (2.24), we obtain

$$\begin{aligned}
&\mathbb{E}[F(x^{(k+1)})] - \mathbb{E}[F(x^{(k)})] \\
&\leq -(\mu - \frac{1}{2}\bar{\alpha}LM_G)\bar{\alpha}\mathbb{E}[\| \nabla F(x^{(k)}) \|_2^2] + \frac{1}{2}\bar{\alpha}^2 LM \\
&\leq -\frac{1}{2}\mu\bar{\alpha}\mathbb{E}[\| \nabla F(x^{(k)}) \|_2^2] + \frac{1}{2}\bar{\alpha}^2 LM.
\end{aligned}$$

Summing both sides of this inequality for $k \in \{1, \cdots, K\}$ and recalling Assumption 2 gives

$$\begin{aligned} F_{inf} - F(x^{(1)}) &\leq \mathbb{E}[F(x^{(K+1)})] - F(x^{(1)}) \\ &\leq -\frac{1}{2}\mu\bar{\alpha}\sum_{k=1}^{K}\mathbb{E}[\| \nabla F(x^{(k)}) \|_2^2] + \frac{1}{2}K\bar{\alpha}^2 LM. \end{aligned}$$

Rearranging yields (2.25), and dividing further by K yields (2.26). □

Again in the non-convex context we now consider the case of decreasing steplengths.

Theorem 6 (Non convex Objective, Diminishing Steplengths)**.** *Under Assumptions 1 and 2, suppose that SG method is run with a steplength sequence satisfying* (2.19)*. Then*

$$\liminf_{k\to\infty} \mathbb{E}[\| \nabla F(x^{(k)}) \|_2^2] = 0. \tag{2.27}$$

The proof of this theorem follows based on the results given in Theorem (7) below.

Theorem 7 (Non convex Objective, Diminishing Steplengths)**.** *Under Assumptions 1 and 2, suppose that SG method is run with a steplength sequence satisfying* (2.19)*. Then, with* $A_k := \sum_{k=1}^{K} \alpha_k$*,*

$$\mathbb{E}\left[\sum_{k=1}^{K} \alpha_k \| \nabla F(x^{(k)}) \|_2^2\right] < \infty \tag{2.28}$$

$$\mathbb{E}\left[\frac{1}{A_k}\sum_{k=1}^{K} \alpha_k \| \nabla F(x^{(k)}) \|_2^2\right] \xrightarrow{K\to\infty} 0. \tag{2.29}$$

Proof. The second condition in (2.19) ensures that $\{\alpha_k\} \to 0$, meaning that, without loss of generality, we may assume that $\alpha_k LM_G \leq \mu$ for all $k \in \mathbb{N}$. Then, taking the total expectation of (2.11),

$$\begin{aligned} &\mathbb{E}[F(x^{(k+1)})] - \mathbb{E}[F(x^{(k)})] \\ &\leq -(\mu - \frac{1}{2}\alpha_k LM_G)\alpha_k \mathbb{E}[\| \nabla F(x^{(k)}) \|_2^2] + \frac{1}{2}\alpha_h^2 LM \\ &\leq -\frac{1}{2}\mu\alpha_k \mathbb{E}[\| \nabla F(x^{(k)}) \|_2^2] + \frac{1}{2}\alpha_k^2 LM. \end{aligned}$$

Summing both sides of this inequality for $k \in \{1, \cdots, K\}$ gives

$$\begin{aligned} F_{inf} - \mathbb{E}[F(x^{(1)})] &\leq \mathbb{E}[F(x^{(K+1)})] - \mathbb{E}[F(x^{(1)})] \\ &\leq -\frac{1}{2}\mu \sum_{k=1}^{K} \alpha_k \mathbb{E}[\| \nabla F(x^{(k)}) \|_2^2] + \frac{1}{2} LM \sum_{k=1}^{K} \alpha_k^2. \end{aligned}$$

Dividing by $\frac{\mu}{2}$ and rearranging the terms, we obtain

$$\sum_{k=1}^{K} \alpha_k \mathbb{E}[\| \nabla F(x^{(k)}) \|_2^2] \leq \frac{2(\mathbb{E}[F(x^{(1)})] - F_{inf})}{\mu} + \frac{LM}{\mu} \sum_{k=1}^{K} \alpha_k^2.$$

The second condition in (2.19) implies that the right-hand side of this inequality converges to a finite limit when K increases, proving (2.28). Then, (2.29) follows since the first condition in (2.19) ensures that $A_K \to \infty$ when $K \to \infty$. □

Corollary 8. Suppose the conditions of Theorem 7 hold for any $K \in \mathbb{N}$. Then $k(K) \in \{1, \cdots, K\}$ represent a random index chosen with probabilities proportional to $\{\alpha_k\}_{k=1}^{K}$.
Then $\| \nabla F(x^{(k(K))}) \|_2 \xrightarrow{K \to \infty} 0$ in probability.

Proof. Using Markov's inequality:

$$\mathbb{P}(X \geq a) \leq \frac{\mathbb{E}[X]}{a} \text{ for } X \text{ random variable and } a \geq 0$$

and (2.28), for any $\epsilon > 0$, we can write

$$\begin{aligned} \mathbb{P}\{\| \nabla F(x^{(k)}) \|_2 \geq \epsilon\} &= \mathbb{P}\{\| \nabla F(x^{(k)}) \|_2^2 \geq \epsilon^2\} \\ &\leq \epsilon^{-2} \mathbb{E}[\mathbb{E}_k[\| \nabla F(x^{(k)}) \|_2^2]] \xrightarrow{K \to \infty} 0, \end{aligned}$$

which is the definition of convergence in probability. □

Now we have that SG is arguably one of the most popular algorithms in machine learning. Unfortunately, SG suffers from slow convergence, which is due to the fact that the variance of the stochastic gradient as an estimator of the gradient does not naturally diminish. For this reason, SG is typically used with a decreasing steplength rule, which ensures that the variance converges to zero. However, this has an adverse effect on the convergence rate. For instance, SG has a sublinear rate even if the function to be minimised is strongly convex.

2.2.2 Variance reduced methods

In literature gradient aggregation methods have a long history. The basic idea of this class of methods is to achieve a lower variance by reusing and/or revising previously computed informations. For example, Bertsekas et al. [6] have proposed incremental gradient methods, that can be view as instance of a basic SG method. Therefore other methods named Stochastic Variance Reduced Gradient (SVRG) and SAGA are able to achieve a linear rate of convergence on strongly convex problems. On the other hand these methods achieve this rate of convergence with an increase in the computation or in the storage.
SVRG method operates in cycles. The scheme is described in Algorithm 2; at the beginning of each cycle, the full gradient $\nabla F_n(x^{(k)})$ is available; then, starting from $x^{(k)}$, a set of m inner iterations $\tilde{x}^{(j)}$ are performed by a stochastic gradient, corrected by means of a perceived bias:

$$\tilde{g}_j \leftarrow \nabla f_{i_j}(\tilde{x}^{(j)}) - (\nabla f_{i_j}(x^{(k)}) - \nabla F_n(x^{(k)}))$$

with i_j randomly chosen in $\{1, ..., n\}$.
In this scheme $\tilde{g}_j$ is an unbiased estimator of the gradient but with a reduced variance. In options (b) and (c) when $\rho = \frac{1}{1-2\alpha L}(\frac{1}{mc\alpha} + 2L\alpha) < 1$, the rate

Algorithm 2 SVRG Method

Select an initial iterate $x^{(1)} \in \mathbb{R}^d$, a steplength $\alpha > 0$ and $m \in \mathbb{N}$.
for $k = 1, 2, \cdots$ **do**
  Compute the batch gradient $\nabla F_n(x^{(k)})$
  Initialize $\tilde{x}^{(1)} = x^{(k)}$.
  for $j = 1, 2, \cdots, m$ **do**
    Choose i_j uniformly from $\{1, ..., n\}$
    Compute $\tilde{g}_j \leftarrow \nabla f_{i_j}(\tilde{x}^{(j)}) - (\nabla f_{i_j}(x^{(k)}) - \nabla F_n(x^{(k)}))$.
    Set $\tilde{x}^{(j+1)} \leftarrow \tilde{x}^{(j)} - \alpha \tilde{g}_j$.
  end for
  Option (a): Set $x^{(k+1)} \leftarrow \tilde{x}^{(m+1)}$.
  Option (b): Set $x^{(k+1)} \leftarrow \frac{1}{m}\sum_{j=1}^{m} \tilde{x}^{(j+1)}$.
  Option (c): Set $x^{(k+1)} \leftarrow \tilde{x}^{(j+1)}$, where j is chosen uniformly from $\{1, ..., m\}$.
end for

of convergence is linear

$$\mathbb{E}[F_n(x^{(k+1)} - F_n(x_*)] \leq \rho \mathbb{E}[F_n(x^{(k)} - F_n(x_*)]$$

The outer iteration of SVRG method is very time consuming; in fact it requires $n+2m$ gradient evaluations. Numerical experiments show that, in the beginning phase, SG is more efficient than SVRG.
A different approach is used in SAGA method. At the first iteration, the full gradient is computed, storing any component of the sum; then, at each iteration, a stochastic gradient is computed as the average of the last gradient components evaluated at previous iterates; the method will have stored $\nabla f_i(x^{[i]})$, $i = 1, ..., n$, where $x^{[i]}$ is the latest iterate at which ∇f_i was evaluated. The basic idea of the method is that $\mathbb{E}[g_k] = \nabla F_n(x^{(k)})$ is an unbiased estimator of the gradient but with a reduced variance. In order to start the method, one could perform one epoch of simple SG steps or one can assimilate iterates one by one. Beyond the initial iteration, the computational

Algorithm 3 SAGA Method

Select an initial iterate $x^{(1)} \in \mathbb{R}^d$, a steplength $\alpha > 0$.
for $i = 1, 2, \cdots, n$ **do**
  Compute $\nabla f_i(x^{(1)})$
  Store $\nabla f_i(x^{[i]}) \leftarrow \nabla f_i(x^{(1)})$
end for
for $k = 1, 2, \cdots$ **do**
  Choose j uniformly in $\{1, ..., n\}$
  Compute $\nabla f_j(x^{(k)})$.
  Set $g_k \leftarrow \nabla f_j(x^{(k)}) - \nabla f_j(x^{[j]}) + \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(x^{[i)]})$.
  Store $\nabla f_j(x^{[x]}) \leftarrow \nabla f_j^{(x_k)}$
  Set $x^{(k+1)} \leftarrow x^{(k)} - \alpha g_k$
end for

cost is as in SG method. The drawback is that we need to store n gradient vectors. In addition, the memory traffic can be very expensive.
For special functions (for example logistic and least squares regression) as $f_i(x^{(k)}) = \hat{f}(a_i^T x^{(k)})$, then $\nabla f_i(x^{(k)}) = \hat{f}'(a_i^T x^{(k)}) a_i$. Consequently, it is sufficient the additional storing of the scalars $\hat{f}'(a_i^T x^{(k)})$ since the examples are already available.
When $\alpha = \frac{1}{2(cn+L)}$, the rate of convergence is linear

$$\mathbb{E}[\|x^{(k)} - x_*\|^2] \leq \left(1 - \frac{c}{2(cn+L)}\right)^k (\|x^{(1)} - x^{(*)}\|^2 + \frac{nD}{cn+L})$$

with $D = F_n(x^{(1)}) - \nabla F_n(x_*)^T(x^{(1)} - x_*) - F_n(x_*)$.
For very large n, gradient aggregation methods are comparable to batch algorithms and therefore cannot beat SG.

Another way to overcome the variance-reduced problem is to consider a new class of methods that was developed over the last years; these kinds of methods include a momentum term [40] or they are bases on adaptive estimates of lower-order moments [36].
As we can see in Algorithm 4, the vector m_k used in the basic iteration takes into account all the gradients seen in the past, weighted with a term β that makes them decay exponentially (being β a scalar always smaller than 1) (see [40]).
Similarly in Algorithm 5, known as AdaM, Adaptive Moment estimation, in literature [36], in addition to a term related to the first moment of the stochastic process of the gradient evaluation, it is considered a term that takes into account the average of the component-wise square of the gradients. Also in this case weighted averages are considered, with exponential decay of the gradients seen in the past. Recent convergence results on variants of AdaM are obtained in [15].

Algorithm 4 Momentum

Choose α, $\beta \in [0, 1)$, $x^{(0)}$;
initialize $m_0 \leftarrow 0$, $k \leftarrow 0$
for $k = 0, 1, \cdots$ **do**
  $k \leftarrow k + 1$
  $g_k \leftarrow \nabla f_{i_k}(x^{(k-1)})$
  $m_k \leftarrow \beta \cdot m_{k-1} + g_k$
  $x^{(k)} \leftarrow x^{(k-1)} - \alpha \cdot m_k$
end for

Algorithm 5 Adam

Choose α, $\hat{\epsilon}$, β_1 and $\beta_2 \in [0, 1)$, $x^{(0)}$;
initialize $m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $k \leftarrow 0$
for $k = 0, 1, \cdots$ **do**
  $k \leftarrow k + 1$
  $g_k \leftarrow \nabla f_{i_k}(x^{(k-1)})$
  $m_k \leftarrow \beta_1 \cdot m_{k-1} + (1 - \beta_1) \cdot g_k$
  $v_k \leftarrow \beta_2 \cdot v_{k-1} + (1 - \beta_2) \cdot g_k^2$
  $\alpha_k = \alpha \frac{\sqrt{1-\beta_2^k}}{(1-\beta_1^k)}$
  $x^{(k)} \leftarrow x^{(k-1)} - \alpha_k \cdot m_k / (\sqrt{v_k} + \hat{\epsilon})$
end for

2.3 Artificial Neural Networks

In addition to the classic Machine Learning context, stochastic gradient methods are also used for Artificial Neural Network (ANN) training or, more generally, in the context of Deep Learning. Deep Learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction.
ANNs are biologically inspired algorithms designed to simulate the way in which the human brain processes information. The networks collect their knowledge by detecting the patterns and relationships in data, and learn (or are trained) through experience, not from programming. An ANN is formed by hundreds of single units, artificial neurons or processing elements (PE), connected with coefficients (weights), which constitute the neural structure and are organised in layers. Each PE has weighted inputs, activation function, and one output. The behaviour of a neural network is determined not only by the activation functions of its neurons, by the loss function, and by the architecture itself but also by the number of information that it processes simultaneously. The weights are the adjustable parameters and, in that sense, an ANN is a parameterised system. The weighed sum of the inputs constitutes the activation signal of the neuron. The activation signal is passed through activation function to produce a single output of the neuron. Activation function introduces non-linearity to the network. During training, the inter-unit connections (the weights) are optimised until the error in predictions is minimised; once the network is trained and tested, it can be given new input information to predict the output.

Examples of ANN and of activation functions

A first example of ANN was the Linear Threshold Unit (LTU) or perceptron. Assigned an input vector $\xi \in R^d$, indicating with $x \in R^d$ the weight vector, a Linear Threshold Unit (LTU) model is defined as follows:

$$LTU(\xi) = \begin{cases} 1, & \text{if } \sum_{i=1}^{d} x_i \xi_i \geq \theta \\ -1, & \text{if } \sum_{i=1}^{d} x_i \xi_i < \theta \end{cases}$$

where θ is a threshold that specifies the amount of stress to be accumulated in the neuron before it fires a positive signal (see Figure 2.1).
Mathematically, a LTU is able to draw an hyperplane in $\mathbb{R}^d$.

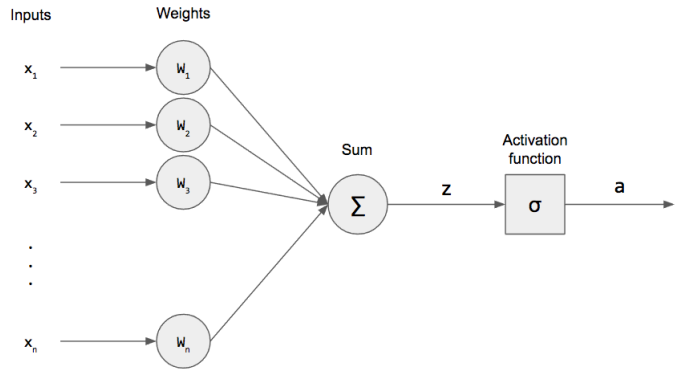


Figure 2.1: Scheme of a formal neuron

Indeed, since $y \in \{-1, 1\}$ is the output of the neuron, we can define:

$$y(\xi) = g\left(\sum_{i=1}^{d} x_i \xi_i - \theta\right) = g(x^T \xi - \theta)$$

where g is called *activation function* and can be defined as the *sign* function $LTU(\xi)$. In this case the ANN is an hyperplane of $\mathbb{R}^d$.
In more general cases, different types of activation functions can be introduces. We report some examples.

1 Sigmoid or Logistic Activation Function $g(t) = \frac{1}{1+e^{-t}}$;

- it exists between (0 to 1); therefore, it is especially used for models where we have to predict the probability as an output;
- it is differentiable;
- it is monotonic but function's derivative is not.

2 Tanh or hyperbolic tangent Activation Function $g(t) = \tanh(t) = \frac{e^t - e^{-t}}{e^t + e^{-t}}$;

- the range of the tanh function is from (-1 to 1);
- tanh is also sigmoidal (s-shaped);
- the advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph;
- it is differentiable;

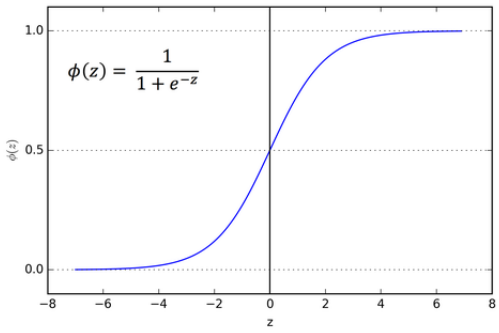


Figure 2.2: Sigmoid function

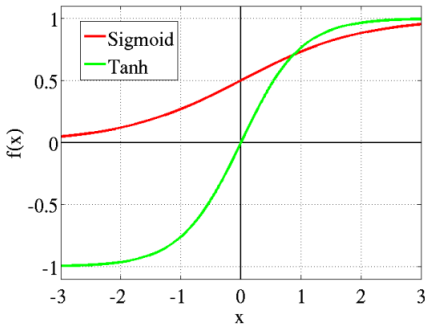


Figure 2.3: tanh function

- it is a monotonic function while its derivative is not monotonic.

3 ReLU (Rectified Linear Unit) Activation Function $g(t) = \max(0, t)$;

- range: [0 to infinity);
- the function and its derivative both are monotonic;
- a drawback is that all the negative values become zero immediately; this decreases the ability of the model to fit or train from the data properly.

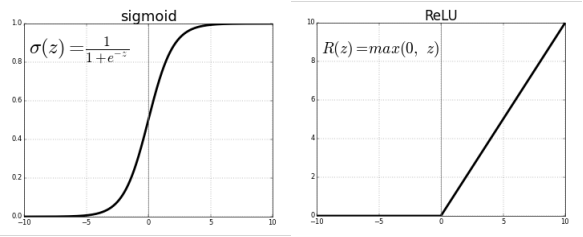


Figure 2.4: ReLU vs Logistic

4 Leaky ReLU $g(t) = \begin{cases} ax, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases}$;

- the leak helps to increase the range of the ReLU function, usually, the value of a is 0.01;
- the range of the Leaky ReLU is $(-\infty, \infty)$;
- when a is different from 0.01, then it is called Randomised ReLU;
- both Leaky and Randomised ReLU functions are monotonic, their derivatives also are monotonic.

2.3.1 Definition of a general multilayer neural network

It is well known that a classifier based on the formal neuron has limited possibilities of application; indeed, there are simple classification problems in which the sample sets are not linearly separable. Some of these limitations can be overcome, in principle, by introducing a map $\phi : \mathbb{R}^d \to \mathbb{R}^d$, transforming the input data in a subset of a space, known as features space. With a

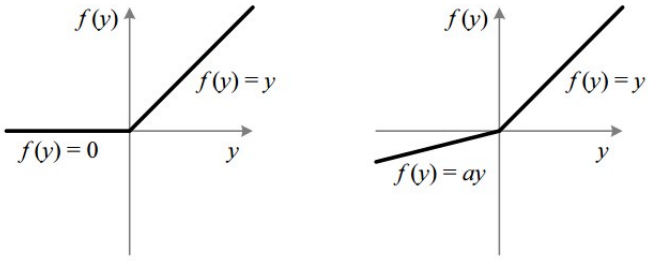


Figure 2.5: ReLU vs Leaky ReLU

suitable choice of this features space, the problem is reformulate as to find linearly separable sets in the transformed space:

$$y(\xi) = g\left(\sum_{i=1}^{d} x_i \phi_i(\xi) - \theta\right).$$

This possibility is still subject to considerable limitations, at least as long as there are limits on the number or complexity of the component ϕ_i of the functions ϕ and the only parameters that can be determined with adaptive criteria are the weights x_i. To overcome the limitations of the perceptron, it is necessary to use more complex structures, making the structure of the functions ϕ_i dependent on the training process. Networks of this type allow, in principle, under appropriate hypotheses on the activation functions of neurons, to approximate within a prefixed accuracy any continuous function on a compact set and, therefore, to solve problems of classification of sets which are not linearly separable. In this regard, there are several results in the literature. In particular, the Universal Approximation Theorem is often referred to. This theorem is presented in [41] in a version with assigned depth. In this work the authors show a universal approximation theorem for width-bounded ReLU networks: width-$(n + 4)$ ReLU networks, where n is the input dimension, are universal approximators. Moreover, except for a measure zero set, all functions cannot be approximated by width-n ReLU networks, which exhibits a phase transition.
The architecture of a multilayer neural network can be described by defining:

- a set of d input nodes associated to the d inputs of the network;
- a set of neurons organised in $L \geq 2$ different layers divided in $L-1$ hidden layers, each consisting of neurons whose outputs contribute to the neuronal inputs of the next layer, and an output layer, constituted from $K \geq 1$ neurons whose outputs constitute the outputs of the network;

- a set of oriented and weighted arcs representing the internal connections between the neurons and the connections with the input nodes; it is assumed that there are no connections between the neurons of the same layer, nor connections in feedback between the outputs of the neurons of a layer and the inputs of the neurons of the previous layers.

The following notation is used:

- $l = 1, \ldots, L$ are the indices associated with the different layers;
- each oriented arc entering the neuron j of the layer l and leaving the neuron i of the layer $l-1$ has a weight consisting of a real number $x_{ji}^{(l)}$ which represents the entity of the synaptic connection;
- each formal neuron is supposed to be characterised by an activation function $g_j^l : \mathbb{R} \to \mathbb{R}$ which operates on a weighted combination of the inputs and a threshold value $x_{j0}^{(l)}$.

Indicating with $a_j^{(l)}$ the weighted sum of the inputs and the threshold and with $z_j^{(l)}$ the output of the neuron, for the neuron j of layer 1 we can write:

$$a_j^{(1)} = \sum_{i=1}^{d} x_{ji}^{(1)} \xi_i - w_{j0}^{(1)}, \quad z_j^{(1)} = g_j^{(1)}(a_j^{(1)})$$

and, for neuron j of layer $l > 1$,

$$a_j^{(l)} = \sum_{i=1}^{N^{(l-1)}} x_{ji}^{(l)} z_i^{(l-1)} - x_{j0}^{(l)}, \quad z_j^{(l)} = g_j^{(l)}(a_j^{(l)})$$

being $N^{(l)}$ the number of neurons of layer l. Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer (see Figure 2.7). For performing backpropagation, stochastic gradient methods are used. Indeed backpropagation is the technique to compute the gradient of one component of the objective function (2.3).
The following is a minimal example of a gradient calculation for a fully connected network.

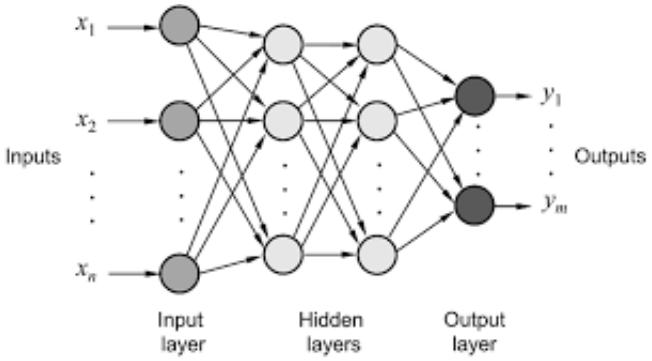


Figure 2.6: Scheme of multilayer ANN

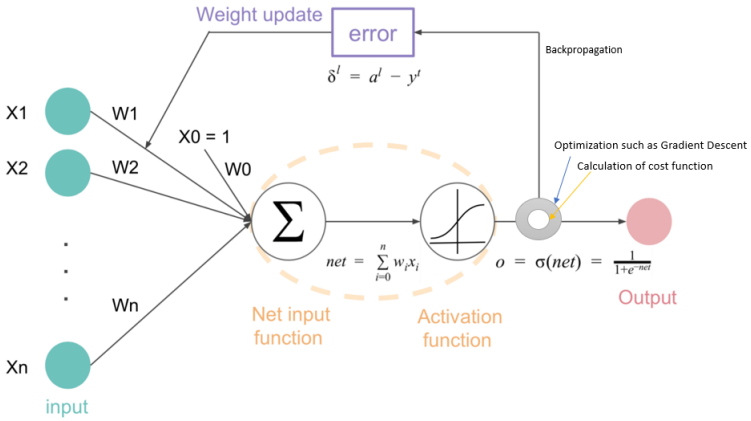
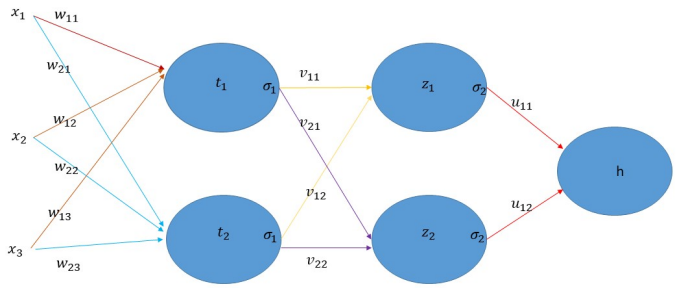


Figure 2.7: Backpropagation scheme



For any sample $\xi = (a, b)$, we assume that $h(a; x)$ is a prediction function of the neural network about the example a with given parameters x and $\ell(h(a; x), b)$ is the loss function, measuring the distance between the computed prediction $h(a; x = (w, v, u))$ and the true value b; then the expression of $\ell(h(a; x = (w, v, u)), b)$ can be written as

$$\begin{aligned} h(a; x = (w, v, u)) &= u_{11}\sigma_2(z_1) + u_{12}\sigma_2(z_2) = \\ &= u_{11}\sigma_2(v_{11}\sigma_1(t_1) + v_{12}\sigma_1(t_2)) + u_{12}\sigma_2(v_{21}\sigma_1(t_1) + v_{22}\sigma_1(t_2)) = \\ &= u_{11}\sigma_2(v_{11}\sigma_1(w_{11}a_1 + w_{12}a_2 + w_{13}a_3) + v_{12}\sigma_1(w_{21}a_1 + w_{22}a_2 + w_{23}a_3)) + \\ &\quad + u_{12}\sigma_2(v_{21}\sigma_1(w_{11}a_1 + w_{12}a_2 + w_{13}a_3) + v_{22}\sigma_1(w_{21}a_1 + w_{22}a_2 + w_{23}a_3)) \end{aligned}$$

We compute the gradient:

$$\begin{aligned} \nabla_{(w\ v\ u)}\ell(h(a; x = (w, v, u)), b) &= \ell'(h(a; x = (w, v, u)), b)\nabla_{(w\ v\ u)}h(a; x = (w, v, u)) \\ \nabla_u h(a; x = (w, v, u)) &= \begin{pmatrix}\sigma_2(z_1) & \sigma_2(z_2)^T\end{pmatrix} \\ \text{row v(1,:)}\quad \nabla_{v_{1,:}} h(a; x = (w, v, u)) &= u_{11}\sigma_2'(z_1)\begin{pmatrix}\sigma_1(t_1) & \sigma_1(t_2)\end{pmatrix}^T \\ \text{row v(2,:)}\quad \nabla_{v_{2,:}} h(a; x = (w, v, u)) &= u_{12}\sigma_2'(z_2)\begin{pmatrix}\sigma_1(t_1) & \sigma_1(t_2)\end{pmatrix}^T \\ \text{row w(1,:)}\quad \nabla_{w_{1,:}} h(a; x = (w, v, u)) &= (u_{11}\sigma_2'(z_1)v_{11} + u_{12}\sigma_2'(z_2)v_{21})\sigma_1'(t_1)\begin{pmatrix}a_1 & a_2 & a_3\end{pmatrix}^T \\ \text{row w(2,:)}\quad \nabla_{w_{2,:}} h(a; x = (w, v, u)) &= (u_{11}\sigma_2'(z_1)v_{12} + u_{12}\sigma_2'(z_2)v_{22})\sigma_1'(t_2)\begin{pmatrix}a_1 & a_2 & a_3\end{pmatrix}^T \end{aligned}$$

Deep Learning and Convolutional Neural Networks (CNN)

Deep Learning and Convolutional Neural Networks have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics.

Deep convolutional nets have brought about breakthroughs in processing images, video, speech and audio, whereas recurrent nets have shone light on sequential data such as text and speech [37].

A CNN is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.
The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image. CNN need not be limited to only one Convolutional Layer; the first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc; with added layers, the architecture adapts to the High-Level features as well, giving us a network which has the wholesome understanding of images in the dataset, similar to how we would. There are two types of results to the operation: one in which the convolved feature is reduced in dimensionality as compared to the input, and the other in which the dimensionality is either increased or remains the same. This is done by applying Valid Padding in case of the former, or Same Padding in the case of the latter.
Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature; this is to decrease the computational power required to process the data through dimensionality reduction; furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model. There are two types of Pooling: *Max Pooling* and *Average Pooling*. In both cases we consider as input (kernel) a rectangular region and as output a single value. *Max Pooling* returns the maximum value from the portion of the image covered by the kernel. On the other hand, *Average Pooling* returns the average of all the values from the portion of the image covered by the kernel.
Max Pooling also performs as a Noise Suppressant. It discards the noisy activations altogether and also performs denoising along with dimensionality reduction. On the other hand, Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism. Hence, we can say that Max Pooling performs a lot better than Average Pooling. Adding a Fully-Connected layer is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer. The Fully-Connected layer is learning a possibly non-linear function in that space. Now that we have converted the input image into a suitable form by the Multi-Level Perceptron, we shall flatten the image into a column vector. The flattened output is fed to a feed-forward neural network and backpropagation applied to every iteration of training. Over a series of epochs, the model is able to distinguish between dominating and certain low-

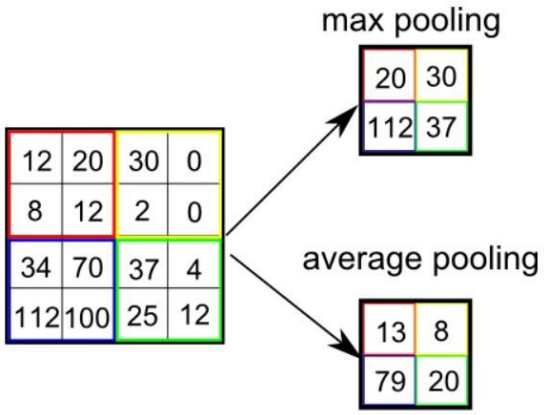


Figure 2.8: Different pooling kinds

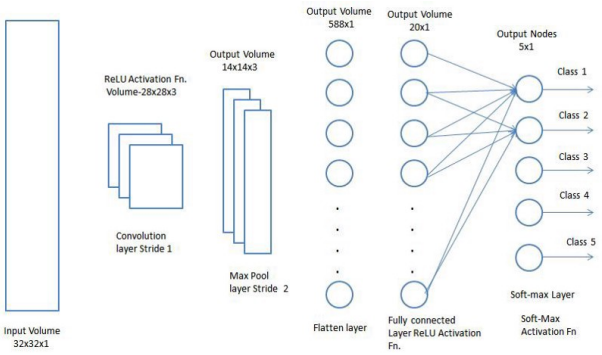


Figure 2.9: Fully connected layer

level features in images and classify them using the Softmax Classification technique. The softmax function, also known as softargmax or normalised exponential function is a function that:

- takes as input a vector of K real numbers, and normalises it into a probability distribution consisting of K probabilities;
- prior to applying softmax, some vector components could be negative, or greater than one; and might not sum to 1;
- after applying softmax, each component will be in the interval $(0, 1)$ and the components will add up to 1;
- they can be interpreted as probabilities;

- the larger input components will correspond to larger probabilities;
- softmax is often used in neural networks, to map the non-normalised output of a network to a probability distribution over predicted output classes.

The standard (unit) softmax function $\sigma : \mathbb{R}^k \to \mathbb{R}^k$ is defined by the formula

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K$$

In particular: we apply the standard exponential function to each element z_i of the input vector $\mathbf{z}$ and normalise these values by dividing by the sum of all these exponentials; this normalisation ensures that the sum of the components of the output vector $\sigma(\mathbf{z})$ is 1.

2.4 Mini-batch size

Masters and Luschi in [42] say that modern Deep Neural Network (DNN) training is typically based on mini-batch stochastic gradient optimisation. While the use of large mini-batches increases the available computational parallelism, small batch training has been shown to provide improved generalisation performance and allows a significantly smaller memory footprint, which might also be exploited to improve machine throughput. In this way the authors stress how important it is in the learning process to set up a good mini-batch size that allows the method to achieve high accuracy while reducing the time spent on the learning phase as much as possible. In the context of classic Machine Learning several authors suggest a constant and linear growth of the mini-batch to allow the process to reach the entire dataset. Some authors, for example Schmidt in [26], suggest a hybrid approach to the problem. The hybrid approach consists in using some initial iterates with a stochastic gradient method, to exploit the initial fast decreasing ability of these methods, and then move to deterministic iterates to exploit the stability of the method and the ability to go further down to the identified minimum. Obviously, these techniques cannot be applied to every context, but only to off-line learning contexts and with datasets of limited size. In the field of Deep Learning it is often suggested the use of a mini-batch size in powers of 2 such as $32, 64, 128$ to facilitate the use of internal memories of accelerators such as Graphics Processing Unit (GPU) and Field Programmable Gate Array (FPGA). The sample size selection strategies are in some cases not adaptive but simply guided by heuristics or hardware constraints, while in

other adaptive contexts they are guided by the learning process itself [45].
As in any learning methodology from examples also in Machine Learning in general, and Deep Learning in particular, it is important to adopt techniques to avoid overfitting. This can be obtained by adding to the error function a Tikhonov–like penalty term with the effect of restricting the set within which the parameters are chosen:

$$\min_{x\in\mathbb{R}^d} \sum_{i=1}^{n} f_i(x) + \lambda \parallel x \parallel^2 \quad \lambda > 0. \tag{2.30}$$

The "optimal" value of λ can be determined using a cross–validation technique. In particular, for different values of λ, various networks are trained solving(2.30), and the parameter λ is chosen as the one which minimises the error on a validation set.
Starting from these considerations in this thesis we will present an idea of dynamic increase of the sample size of adaptive type. As in all methodologies involving learning from examples, also in Deep Learning, it is fundamental to keep under control the phenomenon of overfitting, i.e. the excessive adherence to data with consequent loss of the ability to generalise. An effective method to avoid overfitting is the Early Stopping approach, which interrupts the learning process, as the name suggests. The proposed methodology starts from the intuition of the Early stopping [47] technique and takes advantage of the validation set. As an alternative to the regularisation technique, a heuristic strategy sometimes used is to prematurely stop the minimisation (early stopping) of the error function. In particular, the early stopping technique is based on the idea to periodically evaluate, during the minimisation process, the error that the network commits on an auxiliary validation set. In general, in the first iterations the error on the validation set decreases with the objective function, while it can increase if the error occurs on the training set it becomes "sufficiently small". The training process ends when the error on the validation set starts to increase, because this might correspond to the point in which the network begins to overfit the training set to the detriment of generalisation capacity. Several numerical examples have been made to validate the effectiveness of this new idea to increase the accuracy of the method without increasing the computational time.
Turning now to consider the steplength we can find different approaches in literature. In many versions of the stochastic gradient method the steplength is usually left fixed during the learning process; in AdaM [36] instead we have an initial decreasing and then it asymptotically tends (with exponential speed) to a constant value. Considering the convergence results that come from the theory [9], the steplength should be chosen as a value of a diminishing sequence, i.e., $\alpha_k = \mathcal{O}(\frac{1}{k})$, but in practice this choice would lead to

a too quick reduction. The decreasing of the steplength would cause an interruption of the learning process, going to have this value close to machine precision in a few iterates. For this reason in practice we can find a similar idea, but with a much slower speed called learning rate annealing. Several works in literature show how from a theoretical point of view, both in Deep Learning contexts and with convex functions, modifying steplengths during the learning process can bring benefits. For example in [39] the authors claim that SG with a large initial learning rate is widely used for training modern neural net architectures. Although a small initial learning rate allows for faster training and better test performance initially, the large learning rate achieves better generalisation soon after the learning rate is annealed. Other authors claim, however, that these benefits can also be achieved in a convex context, bringing effective numerical examples [46]. The basic idea of learnig rate annealing is to decrease the steplength after some time from the beginning of the learning process, in an automatic and non-adaptive way. This technique is widely used in the most recent ANN for segmentation and other tasks [7]. In contrast to these methods, in the recent panorama of research, techniques have been introduced in which the variation of the steplength is adaptive. From the convergence analysis, it is clear that it is essential to correctly estimate the Lipschitz constant involved in the choice of steplength. Regarding the convergence results of the standard SG method (2.4) and its variant with fixed size sub-sample n_k, in [9] a very thorough analysis is provided. The results provided in [9] are valid in the case of the solution of both problems (2.1) and (2.2). Under the fundamental assumption that the gradient of the objective function is L-Lipschitz continuous and some additional conditions on the first and second moment of the stochastic gradient, when positive steplength α_k is limited from above by a constant α_{max}, the expected optimality gap for strongly convex objective functions, or the expected sum of gradients for non-convex objective functions, converge asymptotically to values proportional to α_{max}. In practice, if the steplength is small enough and $k \to \infty$, the method generates iterations near the optimal or stationary value.

Everything said is valid from a theoretical point of view. In practice the Lipschitz constant and the quantities that limit the first and second moment of the stochastic process are generally unknown. In addition to not being known, there are no documented techniques in the literature to approximate them effectively and inexpensively. For these reasons there are no recommended choices for the steplength depending on the single problem we are facing. By choosing an inappropriate steplength we may be faced with two different scenarios. A choice of too small value can lead to a very slow learning process, while a choice of too high value can cause the method to diverge.

At this point the idea is to be able to incorporate second order information without the computational cost of a real Hessian calculation. There are many different proposals in the literature to overcome this problem without using second-order methods or introducing line search techniques.
In particular, we refer to [51, 57], where the updating rule of the steplength is borrowed from the Barzilai's rules, well known in the deterministic context. In case of strongly convex objective functions, in order to obtain a linear convergence in expectation to zero for the optimality gap [57] or to a solution for the sequence of the iterates [51], the updating rules are inserted in variance reducing schemes as SVRG [34] or SAGA [17]. These methods require to periodically compute the full gradient or to store the last computed term of each gradient in the sum (2.3). As mentioned before, these techniques cannot always be used. They are suitable only if we are in an off-line learning context and with a dataset small enough to be managed entirely.
Another way to obtain the linear convergence for strongly convex objective functions consists in increasing n_k at a geometric rate [11] (see also [25]). Despite this very strong condition, from the practical point of view, a procedure based on the so-called *norm test*, enables to control the sample size n_k so that

$$\mathbb{E}[\|g_k^{(n_k)} - \nabla F(x^{(k)})\|^2] \leq \zeta \|\nabla F(x^{(k)})\|^2$$

for some $\zeta > 0$ [28]. In the practical implementation, the left side of the last inequality can be approximated with the sample variance and the gradient $\nabla F(x^{(k)})$ on the right side with a sample gradient [11, 9]. Similar techniques are developed in [13], relaxing the norm test by the use of a line search technique based on the true value of the objective function.
In several works it has been shown how steplength and mini-batch size are strongly correlated [55]. For this reason it makes sense to consider together the steplength and the mini-batch size in a mutually dependent way. Recently, Bollagragrada et al. suggest in [8] to increase the mini-batch size on the basis of an *internal product test*, combined with an *orthogonality test*. These conditions ensure that the negatives of stochastic gradients based on -samples of adequate size are descent directions for the method. From the numerical experiments we can see that the mechanism which increases the sample size n_k with these two tests is slower than the norm test induced. On the other hand, the linear speed of convergence is maintained for objective functions that fulfill the Polyak-Lojasiewicz (P-L) condition. These results strongly depend on the knowledge of the Lipschitz L parameter or its appropriate (local) estimation. Consequently, motivated by the numerical experiences and theoretical results shown in [21], in this thesis we propose to adapt the selection rule for the steplength adopted in the Limited Memory

Steepest Descent (LMSD) method [20] to give a local estimate of the inverse L in the SG framework, combining this strategy with the technique to increase the size of the detailed sub-sample in [8] to control in an adaptive way the variance of stochastic directions.
Notwithstanding the success of Machine Learning, algorithms of this type can still be difficult to design effectively and their performance usually depends very much on the choice of several criteria (mini-batch size, steplength, optimiser, ANN layer structure, etc.), called hyperparameters. Hyperparameters are considered all those parameters of the method, numerical or non-numerical, which are not trained by the method itself, but which are decided at the beginning and can be modified in an adaptive way. In practice, finding a set of optimal parameters can make a significant difference. A good combination of these hyperparameters can lead to performance comparable to the state of the art, while a poorly studied and hasty setting can lead to bad results. When we perform the search for these excellent hyperparameters we must have a measure to evaluate the performance of the algorithm. In general this evaluation can be done in two ways: considering the accuracy of the method (in the case of classification) or considering the loss function decrease. Although in theory this process is a simple maximisation, in practice it is very difficult due to the considerable time taken to make each test converge, so it is very expensive from a computational point of view. Currently employed architectures in the case of ANN or ML methods more in general have mostly been developed manually by human experts, which is a time-consuming and error-prone process. Because of this, there is growing interest in automated Neural Architecture Search methods (NAS) or Automated Machine Learning (AutoML). It immediately became obvious that the methods of automatic setting of hyperparameters are particularly promising, in fact in recent years there have been many academic publications in this field [18]. The more generic AutoML [19] gives way to NAS when we talk specifically about ANN architecture. In literature this can often be found as optimisation of hyperparameters or meta-learning [29]. The applications are, of course, many: from images classification [59] to the semantic segmentation [32].
On the basis of the observations just made in this thesis, a technique with a double efficacy will be presented. The method presented can predict, at low cost, the accuracy of a method when its hyperparameters vary. This prediction can then be included in a methodology for the research of excellent parameters with Reinforcement Learning (RL) techniques [52]. The idea is to use a Support Vector Machine for Regression (SVR), trained to predict the accuracy of an ANN given its hyperparameters and performance in early epochs. The algorithm we propose may be of particular interest for a fairly

rapid quality assessment of a new learning algorithm. In particular, it may facilitate the optimisation of hyperparameters in an interesting way.

Chapter 3

Dynamic Mini-batch size

The fundamental idea behind Artificial Neural Networks (ANNs) is to learn through experience, by replacing classical programming techniques. This is because in the beginning the inspiration was biological and therefore they were born to emulate the behaviour of the brain. Therefore the aim of an ANN is to acquire knowledge through experience or, more properly, training.

There are several types of ANNs, but we can identify a basic structure for all of them. Artificial neurons or Processing Elements (PE) are organised in layers and form the basic unit of the network; they are usually present in large quantities, from a few hundred to several thousand. The PEs are connected to each other by coefficients (weights). As we can see in Figure 3.1 the parallelism with the biological setting is evident: different information, with their weights, enter the body cell, then they are processed with the addition of a bias. At this point an activation function is applied and the output is created in this way. This process is the modelling of the behaviour of the axon, suitably simplified. The dendrids enter in the axon, are joined and processed to give rise to the output information of the axon itself. Before an

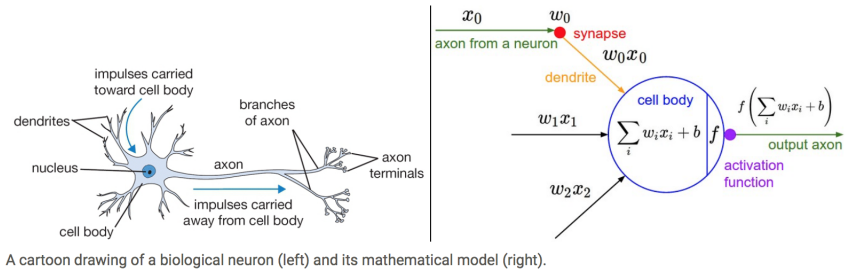


Figure 3.1: Neuron and mathematical model

ANN can be used effectively, several hyperparameters must be defined. Some of them are quite intuitive: number and type of layers, activation functions, loss function, but the number of information processed simultaneously is also very important. The neural network is a parameterised system where, the weights, mentioned before, are the parameters. The weighted sum of the inputs constitutes the activation of the neuron. The activation signal is passed through the activation function to produce a single output of the neuron. The activation function introduces non-linearity into the network. We can represent the merging of all basic operations as a non-linear and, in general, non-convex function, depending on a set of variables, that is the weights of the ANN. At this point the optimisation occurs. What we want to achieve is a minimisation of errors in predictions, and this is reached by modifying the connections between the basic units; this is done through backpropagation or gradient backpropagation. This process is repeated over and over again giving rise to the training of the ANN itself. Once trained, the network is able, given a specific input, to predict the output. The applications of ANNs are innumerable, we can summarise them in model classification or recognition, prediction or modelling.

3.1 ANN history and biological inspiration

The first theory of the neural network model dates back several decades ago; in particular McCulloch and Pitts proposed a first model in 1943. The first proposal was to consider the neuron as a binary device, i.e., simply a logical unit that, given a series of inputs, calculates a logical function. McCulloch and Pitts' neuron can therefore only be found in one of two possible states $\{0, 1\}$. Therefore the inputs that enter the neuron all have the same value. Inside the neuron a threshold operation takes place: if the input impulses exceed a certain threshold the neuron is activated. McCulloch and Pitts were able to demonstrate that a network of neurons of this type could calculate any finite logical expression. This result, in its simplicity, is very strong. For the first time it has been shown that a large number of very simple elements can perform even complex tasks [50]. In this last statement lies the reasons why the ANNs did not develop immediately. There are two main reasons that caused the development of neural networks to stall in the 1940s, the same reasons that have made them progress considerably in recent decades. In the past there were no large amounts of data and powerful hardware to perform large amounts of particularly simple calculations. Recently, however, the collection of data from various types of devices (mobile phones, cameras installed in homes and cities, home automation) has significantly increased

the amount of information available. On the other hand, the use of parallel architectures and of inexpensive GPU units for the execution of particularly simple calculations (typically sums and products) has made available an enormous amount of computing power compared to that of individual CPUs. However, the problem of building an intelligent "neural machine" is still a long way from being solved: in order to make an ANN capable of complex tasks, it is necessary to find a self-organisation mechanism which allows the network to build itself in relation to the task to be executed. An hypothesis proposed by Donald Hebb (1949) is particularly elegant. It has had a great influence on the development of neural network models and has been confirmed, at least partially, by neuro-physiological research. The idea is simple and elegant: synapses do not all have the same intensity, but modify themselves in order to favour the repetition of firing patterns which have more frequently occurred in the past.

3.2 Machine Learning and large scale problems

The promise of Artificial Intelligence (AI) has been a topic of both public and private interest for decades. As mentioned earlier, although the theoretical foundations of AI have developed a long time ago, from a practical point of view only in the last period there have been improvements. Alongside AI, many other methodologies on a statistical basis have developed as in the rapidly evolving and expanding field of Machine Learning (ML). ML in intelligent systems has become an indispensable part of modern society. One of the pillars of machine learning is mathematical optimisation. The aim of such optimisation is to find the best parameters for a system designed to make decisions on never-before-seen data. That is, based on currently available data, these parameters are chosen to be optimal for a given learning problem (and a given loss or cost function) [9]. A loss function or cost function, in this context, is a function that maps the values of variables to a real number that represents a "cost" associated with the event.

Returning to the problem to minimise the objective function, tackling it with the stochastic gradient method corresponding to Algorithm 1 combined with the mini-batch estimate of the gradient (2.5), the main target is to increase accuracy. To increase the accuracy, therefore the effectiveness, of the method, many techniques have been proposed in the literature: we limit ourselves to those involving the mini-batch size. What is proposed is to decrease the steplength or increase the size of the mini-batch. As pointed out by the authors in [55], decreasing the steplength or increasing the size

of the mini-batch has leads to the same learning curves, i.e. the learning rate is comparable. However, what is difficult to find in the literature is an adaptive adjustment using the validation set. In particular, as already pointed out, when we face a problem with any ML technique, we do not know exactly the law of probability that governs the data but, as the only means to guide learning, we have the Data Set. The Data Set is sometimes divided into Training and Validation sets. The Validation set, usually much less numerous than the Training set, contains a set of data that are not used to optimise the parameters, but rather to better set the hyperparameters. In this case the hyperparameter we take into consideration is the size of the mininatch.

3.3 The idea: curiosity to improve accuracy

Consider the iteration

$$x_{k+1} \leftarrow x_k - \bar{\alpha} g(x_k, \xi_k) \tag{3.1}$$

where the stochastic directions are computed for some $\tau > 1$ as

$$g(x_k, \xi_k) := \frac{1}{n_k} \sum_{i \in \mathcal{S}_k} \nabla f_i(x_k; \xi_{k,i}) \text{ with } n_k := |\mathcal{S}_k| = \lceil \tau^{k-1} \rceil. \tag{3.2}$$

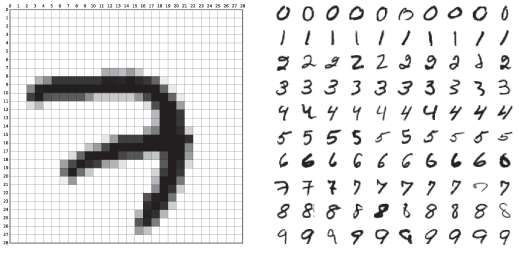
What we are considering is therefore a Stochastic Gradient method in which the steplength is considered constant while the sample size increases [22]. This dimension can increase with a geometric law or in another way that depends on k. As mentioned above, the intuition behind neural networks is biological. For this reason, the proposed new approach can also be justified in a similar context. The analogy in this case is between the number of information managed by animals, or humans, to learn and the size of the mini-batch. Although it is only a few years that man inhabits the earth, it has radically changed its appearance. This extraordinary progress has been made possible by our cognitive structure, particularly by our ability to construct models of external events linked by cause and effect. This characteristic is mainly made possible by our high degree of curiosity. If we analyse the way we understand the world of living beings, animals and especially man, we realise that the cognitive process is always the same. We tend to move from an initial set that contains little information to a higher degree of knowledge, which allows us to process much more information simultaneously. Moving towards a more specific field of Neuroscience, what we observe is that in cognitive processes the most common strategy is to select random actions or to have

bias that lead us to select a behaviour because it is recurrent. Sometimes this behaviour can lead to novelty and uncertainty: and it is here that the human being can acquire new knowledge. Actions guided by randomness lead the subject who performs them to interface with new scenarios and therefore lead to the possibility of learning new things. But, these new actions also have a limitation: they do not guarantee that the subject will learn new concepts. The mere fact that one is approaching a new scenario does not guarantee that it will contain useful information to generalise new knowledge, which can then be spent in other areas. For this reason heuristics based on novelty (in our case a large initial sample) can guide efficient learning in small and limited spaces but in large Data Sets, where the information processed simultaneously is much less than the overall information, this strategy may be ineffective. The same can be said in an on-line learning context, where information is acquired during the learning process. This motivates the introduction of additional strategies, aimed at incremental learning which in our case is the dynamic increase of the sample [27]. In the same way, we find the concept of novelty search and its application in the Evolutionary neural network to create complex and growing structures interesting [38].

3.4 Numerical experiment

The *database* MNIST (Modified National Institute of Standards and Technology database) is a collection of handwritten figures commonly used to test various classification methodologies on images. The *database* is already

Figure 3.2: MNIST dataset.



(a) MNIST sample belonging to the digit '7'. (b) 100 samples from the MNIST training set.

provided divided into $60,000$ images representing the training set and $10,000$ images representing the testing set. Images representing the testing set, even if labelled, should not be used at any stage of training, not even to set the hyperparameters. They must be considered unknown examples, useful only to verify the effectiveness of the method. The images of the training set are further divided, to create the validation set, which contains $5,000$ images.

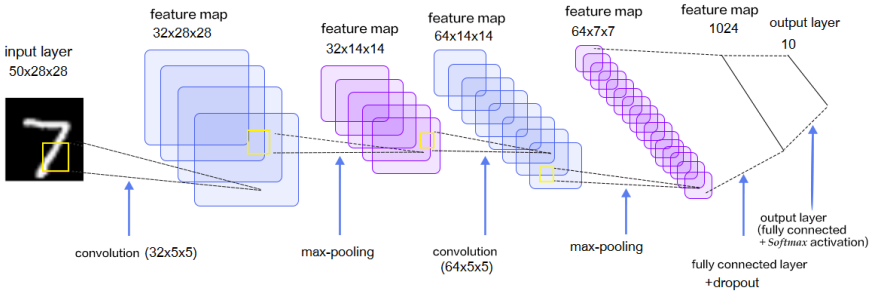


Figure 3.3: CNN with 10 outputs

This will be used to set the stop policy and dynamically increase the size of the mini-batch. So for the actual training will remain $55,000$ images. The images are grayscale (0-255), will then be normalised and moved into the range (0-1) and centered in a 28×28 pixel box.

3.4.1 The problem

The basic idea is to build, through a Convolutional Neural Network (CNN), a multiple classifier and a binary classifier. The multiple classifier will discriminate between the ten digits present in the MNIST database, while the binary classifier will distinguish the figure eight from all the others. In this way we will have to deal with a balanced dataset (in the case of the multiple classification), but also an unbalanced dataset in the case of the binary classification, a task notoriously more difficult in literature.

3.4.2 Convolutional Neural Network

Starting from Figure 3.3 we can briefly describe the structure of the network. The network has an input layer of 784 pixels, that is the linearisation of a 28×28 image of the MNIST dataset. This image is processed through 5 hidden layers, up to the output layer which can be composed of two neurons, in the case of binary classification, or ten neurons, in the case of multiple classification. Obviously, by modifying the size of the input and/or output layer, the network can be adapted to process any other dataset. In the first convolutive+max-pool layer of the network, the image passes through 16 or 32 filters each with a size of 5×5, so we have an output tensor with a size of $16 \times 28 \times 28$ or $32 \times 28 \times 28$. At this point a max-pool operation is applied to this tensor to reduce the image size. We use the Rectified Linear Unit (ReLU) as activation function. At this point we have other convolution and max-pool

layers similar to the previous ones but dimensionally different. In this case the filters applied are 64 always of the size 5×5 and the max-pool applied is the same. We therefore have a resulting tensor of the size $64 \times 7 \times 7$ which is linearised and connected to one layer of 1024 or 2048 neurons through a fully connected one. To reduce the overfitting phenomenon we apply a dropout, i.e. with a given probability we eliminate one or more network connections, and use different probabilities. At this point we connect to the last layer, i.e. the output layer through the Softmax regression function or multinomial logistic regression, which leads us to identify the label among one of the ten possible ones, identifying which of them is assigned the maximum probability.
For the training phase, therefore to adjust the weights with gradient back-propagation, we used the AdaM optimiser. We use this optimiser with the default parameters in Tensorflow.

Listing 3.1: Default parameters

```
tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.0, nesterov=False,
    name='SGD', **kwargs
)
tf.keras.optimizers.Adam(
    learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07,
    amsgrad=False,
    name='Adam', **kwargs
)
```

We propose a new technique, based on an adaptive increase of the mini-bach size, as explained in the new section and we evaluate this approach on different network architectures; in particular we will have networks with a different number of parameters; indeed this number can be modified by the choice of the layer size itself, but also by the dropout coefficient. In addition to the fact that there are two versions of the problem, in multiple and binary classification.
The maximum number of iterates of a network is prefixed equal to 20000. Within the method, however, a stop criterion called "early stopping" is included, which takes into account the actual improvements that the network is making [37]. These improvements are measured, on the validation set, i.e., the set of examples that the network considers only at this stage to evaluate the results. These checks are performed every 100 iterations and, if we don't have any improvements for 15 subsequent times, the training phase stops, even if the maximum number of iterations has not been reached. At this point, the method considers the best obtained network, based on the check

made on the validation set and verifies its accuracy on the testing set to get the final result. In addition to accuracy, the number of errors made and the iterations to compute the best network is also stored.

3.4.3 The new idea in practice

The basic idea that is developed is to increase the size of the mini-batch. Normally, "early stopping" works by evaluating the performance every 100 iterations and checking if the accuracy has improved. At this point, if the accuracy is not improved, a counter is increased; if the accuracy is not increased for 15 subsequent times, then the output from the cycle is forced. The new idea is not to wait for 15 subsequent iterations, but to operate after 10 non-improvement checks. Specifically, after these 10 unsuccessful checks the size of the mini-batch is increased according to the law $n_k \equiv 2n_k$. At this point the training phase is resumed with the increased mini-batch and, if the method reaches 15 consecutive checks without improvement, the exit from the training phase is forced. As usual, the best result obtained on the validation set is verified on the testing set.

3.4.4 Results

Since the preliminary experiments, on 1.6 GHz Intel Core i5 processor, we noticed the effectiveness of the new idea; in particular, errors dropped on average by 30%, but this effectiveness is associated with a doubling of the computational time. The assessment that can be made is that in many applications, however, the important thing is the inference time, not so much the training time, and this is not changed in any way. From a theoretical point of view, the sample can increase until it reaches the whole dataset, but observing the examples in practice this never happens, nor do we approach a dimension of the mini-batch close to the whole space of the examples. What we observe in practice is that the maximum size reached by the mini-batch is 400/800 images in the case of multiple classification and 200/400 in the case of binary classification. Table 3.1 shows the results of a CNN with the same setting of the parameters both in the STATIC case and in the DYNAMIC case; we are in the $32 \times 64 \times 1024 \times 10$ case with dropout equal to 0.5; all the accuracy results have been calculated with respect to the testing set. The only difference between the two simulations is the output criterion explained in detail above. So the STATIC version starts with an initial sample of 50, which it maintains throughout the learning process and the classic "early stopping" is applied with patience equal to 15 (expect 15 controls where there is no improvement before forcing the exit from the learning cycle). While in

the DYNAMIC version the initial sample is always 50, but after 10 controls without an improvement, it is doubled; also in this case after 15 controls without improvement the exit from the cycle is forced.

DYNAMIC	STATIC
99.27	99.34
99.24	99.27
99.35	99.01
99.4	99.12
99.42	99.1

DYNAMIC	STATIC
73	76
66	73
65	99
60	88
58	90

Table 3.1: Experiment 1: accuracy and number of errors

In view of the stochasticity of the model, sample on which the ANN will be trained is randomly chosen and it can be different at any iteration. For this reason 5 simulations have been repeated for each model with exactly the same hyperparameters. At this point, having different results, it makes sense to consider the average and the resulting variance in order to analyse the results more clearly. For the STATIC version, the average accuracy is 99.15 while the average number of errors is 85. In the DYNAMIC case, the average accuracy is 99.36 while the average number of errors is 64, with a decrease greater than 20 units. To understand also how robust the results are beyond the average, one must consider the variance of the method. In the STATIC case, the standard deviation of the method accuracy is 0.1341 while in the DYNAMIC case it is 0.0789. Therefore, in addition to having better results, the new method is also more robust, demonstrating greater stability. Even if we consider the number of errors of the two approaches, we can make similar evaluations; indeed, in the DYNAMIC case the standard deviation is 5.8566 while in the STATIC case it is 10.6630. Nevertheless, we point out that the execution time of the DYNAMIC version is doubled with respect to the STATIC case.
In view of these considerations, it could be assumed that, with more time available to increase the accuracy of the method, it is sufficient to increase the size of the initial sample. But by simply doing this initial increase we do not obtain a higher accuracy of the method. In fact, from an experiment conducted by setting the size of the mini-batch equal to 400 from the beginning, we notice that the times are much higher than the DYNAMIC version, but the accuracy is not so good; in fact the average accuracy stops at 99.2 and the number of errors is 80. These results are slightly better than the STATIC version but not comparable to those obtained with the DYNAMIC version.

DYNAMIC	STATIC-50	STATIC-400
99.88	99.71	99.72
99.77	99.55	99.75
99.81	99.77	99.8
99.87	99.69	99.82
99.81	99.79	99.77

Table 3.2: Accuracy

DYNAMIC	STATIC-50	STATIC-400
12	29	28
23	45	25
9	23	20
13	31	18
19	21	23

Table 3.3: Number of errors

For completeness, in an additional experiment, a comparison between two static versions with different sample sizes and the dynamic version is carried out. The network architecture in this case is $32 \times 64 \times 1024 \times 2$, so we are in the binary case, with a dropout of 0.5. Therefore Tables 3.2 and 3.3 show the results of the cases STATIC-50 (static version with 50 mini-batch size), STATIC-400 (static version with 400 mini-batch size) and DYNAMIC. As in the previous experiment, to appreciate the improvement, we consider the average values of both accuracy and number of errors. In the STATIC-50 case, the average accuracy is 99.7, the average number of errors 30 and the average execution time 96 seconds. In the STATIC-400 case, the average accuracy is 99.77, the average number of errors 23 and the average run time 185 seconds. In the DYNAMIC case, the average accuracy is 99.81, the average number of errors 17 and the average execution time 106 seconds. So, in the DYNAMIC version the time is slightly higher than in the STATIC-50 version but with an average error improvement of more than 30%. In conclusion the average improvement on the number of errors is about equal to 30%.

To evaluate the robustness of the different methods we also consider the standard deviation. In particular, the standard deviation of accuracy in

DYNAMIC	STATIC-50	STATIC-400
140	113	149
80	60	236
86	106	159
145	72	170
77	127	213

Table 3.4: Time in seconds

the DYNAMIC case is 0.046, in the STATIC-50 case is 0.0944 and in the STATIC-400 case 0.0396. These numerical results enables to suppose that the new method is more robust than the STATIC-50 case but less robust than the STATIC-400 case. This is reasonable because with an increase in the size of the mini-batch the stochastic process is more stable. We have the same result for error numbers: in fact the standard deviation for the DYNAMIC case is 4.6043, 9.4446 for the STATIC-50 case and 3.9623 for the STATIC-400 case.
To underline that the key tool of the accuracy improvement is precisely the dynamic increase in the size of the mini-batch, we consider this measurement after each increase of the sample size. In the DYNAMIC case, the final precision is 99.88 and it develops in the following way: accuracy of 99.68 with a mini-batch size of 50, accuracy of 99.82 with a mini-batch size of 100, accuracy of 99.88 with a mini-batch size of 200.
Consider a further experiment where the architecture of the ANN is modified.
We are again in the case of a CNN for binary classification, the structure of the network is $16 \times 64 \times 2048 \times 2$, with dropout equal to 0.1. As previously, we compare the values obtained for STATIC-50, STATIC-400 and DYNAMIC. We observe that the obtained results are similar to the previous ones: in the STATIC-50 case the average accuracy is 99.62 for an average number of errors of 38 and an average time of 158. In the STATIC-400 case the average accuracy is 99.72 for an average error number of 28 and an average time of 174. In the DYNAMIC case the average accuracy is 99.75 for an average number of errors of 20 and an average time of 199. The average improvements is about equal to 30% for DYNAMIC case. In the STATIC-400 case the improvement does not justify the increased time for the training phase, so in this case the DYNAMIC network is preferred to this one.

DYNAMIC	STATIC-50	STATIC-400
99.78	99.75	99.81
99.72	99.64	99.79
99.78	99.62	99.8
99.76	99.46	99.68
99.7	99.62	99.54

Table 3.5: Accuracy

DYNAMIC	STATIC-50	STATIC-400
22	25	19
28	36	21
22	38	20
24	54	32
30	38	46

Table 3.6: Number of errors

DYNAMIC	STATIC-50	STATIC-400
222	163	170
157	137	188
236	181	218
174	130	154
208	180	140

Table 3.7: Time

Chapter 4

Steplength selection

Let us now consider a strategy for the steplength selection in the stochastic gradient methods. We consider the basic SG iteration:

$$x^{(k+1)} = x^{(k)} - \alpha_k g(x^{(k)}, \xi^{(n_k)}) \tag{4.1}$$

As mentioned at the Section 2.2.1, we require that the stochastic gradient is a descent direction sufficiently often.

4.1 Theoretical results on SG method equipped with inner product and orthogonality tests

Assuming that S_k is pick at random from $\{1, ..., n\}$ with $|S_k| = n_k$ and $g_k^{(n_k)} \equiv g(x^{(k)}, \xi^{(n_k)})$ is an unbiased estimate of $\nabla F(x^{(k)})$, i.e., $\mathbb{E}[g_k^{(n_k)}] = \nabla F(x^{(k)})$,

$$\mathbb{E}[{g_k^{(n_k)}}^T \nabla F(x^{(k)})] = \|\nabla F(x^{(k)})\|^2, \tag{4.2}$$

for all $k \geq 0$.
In order for the stochastic gradient to be an appropriate estimation of the full gradient, the idea proposed in [8] is to control the variance of the term on the left of (4.2). In particular, the following condition can be imposed on the sample size n_k of $\xi^{(n_k)}$:

$$\mathbb{E}[({g_k^{(n_k)}}^T \nabla F(x^{(k)}) - \|\nabla F(x^{(k)})\|^2)^2] \leq \theta^2 \|\nabla F(x^{(k)})\|^4, \tag{4.3}$$

for some $\theta > 0$. In addition, this internal product test can be combined with the following orthogonality test, ensuring that the component of $g_k^{(n_k)}$

orthogonal to $\nabla F(x^{(k)})$ is bounded, i.e.,

$$\mathbb{E}[\|g_k^{(n_k)} - \frac{{g_k^{(n_k)}}^T \nabla F(x^{(k)})}{\|\nabla F(x^{(k)})\|^2} \nabla F(x^{(k)})\|^2] \leq \nu^2 \|\nabla F(x^{(k)})\|^2, \tag{4.4}$$

for some $\nu > 0$. The combination of the two tests (4.3) and (4.4) is also known as *augmented inner product test.*
Starting from the results in [8], the following assumptions are considered.

A. ∇F is L-Lipschitz continuous;

B. the Polyak-Lojasiewicz (P-L) condition holds

$$\|\nabla F(x)\|^2 \geq 2c(F(x) - F_*), \quad \forall x \in \mathbb{R}^d, \tag{4.5}$$

where c is a positive constant and $F_* = \inf_{x \in \mathbb{R}^d} F(x)$;

C. $\alpha_k \in (\alpha_{min}, \alpha_{max}]$, and $\alpha_{max} \leq \frac{1}{(1+\theta^2+\nu^2)L}$, for given positive constants θ, ν in (4.3) and (4.4).

We remind that the P-L condition holds when F is c-strongly convex, but not only in this case; indeed it can be satisfied for functions that are not convex (see [35]).
We also note that assumptions A and B do not guarantee the existence of a stationary point for F; however, under the two assumptions, any stationary point for F is a global minimum.
In addition, in view of the recruitment C, the iteration (2.4) can be equipped with a variable steplength, as long as it belongs to the $(\alpha_{min}, \alpha_{max}]$ interval, where α_{max} is proportional to the inverse of L.
Following the arguments of [8], the following theorems can be stated.

Theorem 9. *Suppose the assumptions A and B hold. Let $\{x^{(k)}\}$ be the sequence generated by (2.4), where the size n_k of any sub-sample is chosen so that the conditions (4.3) and (4.4) are satisfied and α_k satisfies the assumption C. Then, we have that*

$$\mathbb{E}[F(x^{(k)}) - F_*] \leq \rho^k (F(x^{(0)}) - F_*), \tag{4.6}$$

where $\rho = 1 - c\ \alpha_{min}$. In particular, for a constant steplength $\alpha_k \equiv \alpha_{max} = \frac{1}{(1+\theta^2+\nu^2)L}$, for all $k \geq 0$, we have $\rho = 1 - \frac{c}{(1+\theta^2+\nu^2)L}$.

Proof. From (4.4) we have

$$\begin{aligned}
&\mathbb{E}[\|g_k^{(n_k)} - \frac{{g_k^{(n_k)}}^T \nabla F(x^{(k)})}{\|\nabla F(x^{(k)})\|^2} \nabla F(x^{(k)})\|^2] = \\
&= \mathbb{E}[\|g_k^{(n_k)}\|^2] - \frac{\mathbb{E}[({g_k^{(n_k)}}^T \nabla F(x^{(k)}))^2]}{\|F(x^{(k)})\|^2} \\
&\leq \nu^2 \|\nabla F(x^{(k)})\|^2
\end{aligned}$$

Therefore we obtain the following inequality:

$$\mathbb{E}[\|g_k^{(n_k)}\|^2 \leq \nu^2 \|\nabla F(x^{(k)})\|^2 + \frac{\mathbb{E}[{g_k^{(n_k)}}^T \nabla F(x^{(k)})^2]}{\|\nabla F(x^{(k)})\|^2}. \tag{4.7}$$

From (4.3) we have

$$\mathbb{E}[({g_k^{(n_k)}}^T \nabla F(x^{(k)}))^2] \leq \|\nabla F(x^{(k)})\|^4 + \theta^2 \|\nabla F(x^{(k)})\|^4 = (1+\theta^2)\|\nabla F(x^{(k)})\|^4$$

Substituting this last inequality in (4.7), we obtain

$$\mathbb{E}[\|g_k^{(n_k)}\|^2] \leq (1 + \theta^2 + \nu^2)\|\nabla F(x^{(k)})\|^2. \tag{4.8}$$

Using the assumption A and $\mathbb{E}[({g_k^{(n_k)}}^T \nabla F(x^{(k)}))] = \|\nabla F(x^{(k)})\|^2$ we have

$$\begin{aligned}
&\mathbb{E}[F(x^{(k+1)})] \leq \mathbb{E}[F(x^{(k)})] - \mathbb{E}[\alpha_k {g_k^{(n_k)}}^T \nabla F(x^{(k)})] + \mathbb{E}[\frac{L\alpha_k^2}{2}\|g_k^{(n_k)}\|^2] \\
&= \mathbb{E}[F(x^{(k)})] - \alpha_k \|\nabla F(x^{(k)})\|^2 + \frac{L\alpha_k^2}{2}\mathbb{E}\|g_k^{(n_k)}\|^2 \\
&\leq \mathbb{E}[F(x^{(k)})] - \alpha_k \|\nabla F(x^{(k)})\|^2 + \frac{\alpha_k^2 L}{2}(1+\theta^2+\nu^2)\|\nabla F(x^{(k)})\|^2 \\
&= \mathbb{E}[F(x^{(k)})] - \alpha_k(1 - \frac{\alpha_k L}{2}(1+\theta^2+\nu^2))\|\nabla F(x^{(k)})\|^2,
\end{aligned}$$

where for $\alpha_k \leq \alpha_{max} \leq \frac{1}{L(1+\theta^2+\nu^2)}$ we obtain

$$\mathbb{E}[F(x^{(k+1)})] \leq \mathbb{E}[F(x^{(k)})] - \frac{\alpha_k}{2}\|\nabla F(x^{(k)})\|^2. \tag{4.9}$$

From the assumption B and subtracting $F(x^*)$ from both sides, we obtain

$$\begin{aligned}
\mathbb{E}[F(x^{(k+1)}) - F(x^*)] &\leq \mathbb{E}[F(x^{(k)}) - F(x^*)] - \frac{\alpha_k}{2} 2c\mathbb{E}[F(x^{(k)}) - F(x^*)] \\
&= (1 - \alpha_k c)\mathbb{E}[F(x^{(k)}) - F(x^*)] \\
&\leq (1 - \alpha_{min} c)\mathbb{E}[F(x^{(k)}) - F(x^*)] = \rho \mathbb{E}[F(x^{(k)}) - F(x^*)].
\end{aligned}$$

If $\alpha_k = \alpha_{max} = \frac{1}{L(1+\theta^2+\nu^2)}$, $\rho = 1 - \frac{c}{L(1+\theta^2+\nu^2)}$. □

Using inequality $\alpha_k \leq \alpha_{max} \leq \frac{1}{(1+\theta^2+\nu^2)L}$ and the condition (P-L) instead of the strong convexity, we get a proof as from Theorem 3.2 of [8].
There is also a version of the same theorem in which the F function is convex but does not meet the condition (P-L). Here we have to take into account $\alpha_k \leq \alpha_{max}$ and the additional close limit on α_{max} and at this point the proof derived from Theorem 3.3 of [8].

Theorem 10. *Suppose the assumption A holds. Let $\{x^{(k)}\}$ be the sequence generated by (2.4), where the size n_k of any sub-sample is chosen so that the conditions (4.3) and (4.4) are satisfied and α_k satisfies the assumption C, with $\alpha_{max} < \frac{1}{(1+\theta^2+\nu^2)L}$. Assume that $X^* = argmin_x F(x) \neq \emptyset$. Then, we have that*

$$\min_{0\leq k\leq K} \mathbb{E}[F(x^{(k)}) - F_*] \leq \frac{1}{2\alpha_{min}\gamma K}\|x^{(0)} - x^*\|^2, \tag{4.10}$$

where $x^* \in X^*$ and $\gamma = 1 - \alpha_{max}L(1 + \theta^2 + \nu^2)$.

Proof. We consider

$$\begin{aligned}
&\mathbb{E}[\|x^{(k+1)} - x^*\|^2] = \\
&\mathbb{E}[\|x^{(k+1)} - x^{(k)}\|^2] + \|x^{(k)} - x^*\|^2 + 2\mathbb{E}[(x^{(k+1)} - x^{(k)})^T(x^{(k)} - x^{(*)})] \\
&= \alpha_k^2\mathbb{E}[\|g_k^{(n_k)}\|^2] + \|x^{(k)} - x^{(*)}\|^2 - 2\alpha_k\mathbb{E}[g_k^{(n_k)T}(x^{(k)} - x^*)].
\end{aligned}$$

Using (4.8) and the convexity of F, we obtain

$$\mathbb{E}[\|x^{(k+1)} - x^*\|^2] \leq \alpha_k^2(1 - \theta^2 + \nu^2)\|\nabla F(x^{(k)})\|^2 + \|x^{(k)} - x^{(*)}\|^2 - 2\alpha_k(F(x^{(k)}) - F(x^*)).$$

Since for any function with Lipschitz continuous gradient we have

$$\|\nabla F(x)\|^2 \leq 2L(F(x) - F^*),$$

we can obtain

$$\begin{aligned}
\mathbb{E}[\|x^{(k+1)} - x^*\|^2] &\leq \|x^{(k)} - x^*\|^2 - 2\alpha_k(1 - \alpha_k(1 + \theta^2 + \nu^2)L)(F(x^{(k)}) - F(x^*)) \\
&\leq \|x^{(k)} - x^*\|^2 - 2\alpha_k(1 - \alpha_{max}(1 + \theta^2 + \nu^2)L)(F(x^{(k)}) - F(x^*)) \\
&= \|x^{(k)} - x^*\|^2 - 2\alpha_k\gamma(F(x^{(k)}) - F(x^*)),
\end{aligned}$$

where $\gamma = 1 - \alpha_{max}(1 + \theta^2 + \nu^2)$.
Now, we can write the inequality:

$$\begin{aligned}
\mathbb{E}[F(x^{(k)})] - F(x^*) &\leq \frac{1}{2\alpha_k\gamma}(\mathbb{E}[\|x^{(k+1)} - x^*\|^2] - \mathbb{E}[\|x^{(k)} - x^*\|^2]) \\
&\leq \frac{1}{2\alpha_{min}\gamma}(\mathbb{E}[\|x^{(k+1)} - x^*\|^2] - \mathbb{E}[\|x^{(k)} - x^*\|^2])
\end{aligned}$$

and summing both sides of this inequality from $k = 0, ..., K$, we obtain:

$$\begin{aligned}\sum_{0\leq k\leq K} \mathbb{E}[F(x^{(k)})] - F(x^*) &\leq \frac{1}{K}\sum_{k=0}^{K-1}(\mathbb{E}[F(x^{(k)})] - F(x^*)) \\ &\leq \frac{1}{2\alpha_{min}\gamma K}(\mathbb{E}[\|x^{(0)} - x^*\|^2] - \mathbb{E}[\|x^{(k)} - x^*\|^2]) \\ &\leq \frac{1}{2\alpha_{min}\gamma K}\mathbb{E}[\|x^{(0)} - x^*\|^2]\end{aligned}$$

□

In case the F is not convex we have a last theorem about convergence. Taking into account that $\alpha_k > \alpha_{min}$ and following Theorem 3.4 in [8] we can say that $\nabla F(x^{(k)})$ converges to zero in expectation, with a sub-linear convergence rate.

Theorem 11. *Suppose the assumption A holds and F is bounded below from F_*. Let $\{x^{(k)}\}$ be the sequence generated by (2.4), where the size n_k of any sub-sample is chosen so that the conditions (4.3) and (4.4) are satisfied and α_k satisfies the assumption C. Assume that $X^* = argmin_x F(x) \neq \emptyset$. Then, we have that*

$$\lim_{k\to\infty} \mathbb{E}[\|\nabla F(x^{(k)})\|^2] = 0. \tag{4.11}$$

Furthermore, for any $K > 0$, we have

$$\min_{0\leq k\leq K-1} \mathbb{E}[\|\nabla F(x^{(k)})\|^2] \leq \frac{1}{2\alpha_{min}K}(F(x^{(0)}) - F_*). \tag{4.12}$$

Proof. From the inequality (4.9) we have:

$$\begin{aligned}\mathbb{E}[\|\nabla F(x^{(k)})\|^2] &\leq \frac{2}{\alpha_k}\mathbb{E}[F(x^{(k)} - F(x^{(k+1)})] \\ &\leq \frac{2}{\alpha_{min}}\mathbb{E}[F(x^{(k)} - F(x^{(k+1)})].\end{aligned}$$

Summing both sides of this inequality from $k = 0 \ldots k-1$ since F is bounded below by F_*, we get

$$\sum_{i=0}^{k-1} \mathbb{E}[\|\nabla F(x^{(k)})\|^2] \leq \frac{2}{\alpha_{min}}\mathbb{E}[F(x^{(0)} - F(x^{(k)})] \leq \frac{2}{\alpha_{min}}\mathbb{E}[F(x^{(0)} - F_*].$$

Taking limits, we obtain

$$\lim_{K\to\infty} \sum_{i=0}^{K-1} \mathbb{E}[\|\nabla F(x^{(K)})\|^2] \leq \infty$$

which implies (4.11). We can also conclude that

$$\min_{0\leq k\leq K-1} \mathbb{E}[\|\nabla F(x^{(k)})\|^2] \leq \frac{1}{K} \sum_{i=0}^{k-1} \mathbb{E}[\|\nabla F(x^{(k)})\|^2] \leq \frac{2}{\alpha_{min} K}[F(x^{(0)}) - F_*].$$

□

The basic idea from which we start is that an estimate, even a local estimate, of the Lipschitz parameter can be useful to evaluate the best steplength. Bollapragada et al. proposes to do this through a backtracking line search procedure.
Our proposal, instead, is to estimate the steplength using the stochastic version of the LMSD calculation of Ritz-like values, taking advantage of the hypothesis that α_k belongs to an appropriate limited range.

4.2 Steplength selection via Ritz and harmonic Ritz values

In the state of the art there are several methods that, through the use of gradients computed at the previous steps, try to obtain second order information, without the computational heaviness that would result from the calculation of the Hessian matrix. In particular, a very effective method is the one proposed in [20]. The idea is to start from this method, which is very effective in the deterministic field, and adapt it appropriately to the stochastic context.

4.2.1 The deterministic framework

Let us first focus on the case of a convex quadratic programming problem. Given a very small m integer, generally no bigger than 7, the idea of [20] is to generate the next m steplengths using the m gradients computed at the last iterations. After a group of m iterations, called sweep, a symmetrical tridiagonal matrix is calculated using the past gradients collected inside the sweep. The eigenvalues of this matrix are approximations of the eigenvalues of the Hessian matrix, so the reciprocals of these eigenvalues are

used as steplengths to perform the next m steps. Obviously, a crucial point is to build the tridiagonal matrix in an economic way. In [20] the following strategy is proposed: suppose that the iterate $x^{(j)}$ and m steplengths $\alpha_{j+k}, k = 0, \ldots, m-1$, are available for performing a new sweep and store the gradients and the steplengths used within the sweep in the following way:

$$G_j = [g_j,\ g_{j+1}, \ldots,\ g_{j+m-1}], \tag{4.13}$$

$$J_j = \begin{pmatrix} \frac{1}{\alpha_j} & & \\ -\frac{1}{\alpha_j} & \ddots & \\ & \ddots & \frac{1}{\alpha_{j+m-1}} \\ & & -\frac{1}{\alpha_{j+m-1}} \end{pmatrix}. \tag{4.14}$$

From the $d \times m$ matrix G_j, an upper triangular $m \times m$ matrix R_j such that $G_j^T G_j = R_j^T R_j$ can be obtained, for example by means of the Cholesky factorisation of $G_j^T G_j$; assuming that G_j is a full-column rank matrix, the matrix R_j is non-singular if G_j is full rank. By using G_j, J_j and R_j define the matrix

$$T_j = R_j^{-T} G_j^T [G_j \quad g_{j+m}] J_j R_j^{-1} = [R_j \quad r_j] J_j R_j^{-1} \tag{4.15}$$

where r_j is the solution of the linear system $R_j^T r_j = G_j^T g_{j+m}$. T_j is the symmetrical tridiagonal matrix of the Lanczos process applied to the Hessian matrix of the objective function, whose initial vector is $g_j/\|g_j\|$. Its eigenvalues are called Ritz values and they approximate the eigenvalues of the Hessian matrix.
In a general non-quadratic case, T_j is upper Hessenberg and a tridiagonal symmetric matrix $\overline{T}_j$ can be obtained as:

$$\overline{T}_j = \text{tril}(T_j) + \text{tril}(T_j, -1)', \tag{4.16}$$

where we use the Matlab notation. The limited memory steplength rule consists in using the reciprocals of the eigenvalues $\lambda_i, \quad i = 1, \ldots, m$ of $\bar{T}_j$ as steplengths for the next m steps:

$$\alpha_{j+m-1+i} = \frac{1}{\lambda_i}, \quad i = 1, \ldots, m. \tag{4.17}$$

Following the terminology used in the quadratic case, we call Ritz-like values the eigenvalues of $\overline{T}_j$.
In [20] another similar strategy is proposed for the calculation of steplengths

for the next steps. In the strictly convex quadratic case, this idea consists in obtaining the steplengths as eigenvalues of the matrix $P_J^{-1}T_j$, where

$$
\begin{aligned}
P_j &= R_j^{-T} J_j^T \begin{pmatrix} R_j & r_j \\ 0 & \rho_j \end{pmatrix}^T \begin{pmatrix} R_j & r_j \\ 0 & \rho_j \end{pmatrix} J_j R_j^{-1} = \\
&= \begin{pmatrix} T_j^T & t_j \end{pmatrix} \begin{pmatrix} T_j \\ t_j^T \end{pmatrix},
\end{aligned} \tag{4.18}
$$

$\rho_j = \sqrt{g_{j+m}^T g_{j+m} - r_j^T r_j}$ and t_j is the solution of the linear system $R_j^T t_j = J_j^T \begin{pmatrix} 0 \\ \rho_j \end{pmatrix}$. The reciprocals of the eigenvalues of $P_J^{-1}T_j$ are called harmonic Ritz. Replacing T_j in (4.18) by the non-singular tridiagonal matrix $\overline{T}_j$, a pentadiagonal matrix $\overline{P}_j$ is obtained. For a non-strictly convex or non-quadratic objective function, the matrices $\overline{T}_j$ and $\overline{T}^{-1}\overline{P}_j$ can have non-positive eigenvalues. This arises when at the current iterate the curvature is negative. There are several works in the literature that take these occurrences into account and propose strategies to obtain a good approximation of Hessian eigenvalues. For example, in [20, 49] the authors suggest, simply, to discard negative values. If too many values are discarded in this way, any provisional values can be adopted for the next steplength. Another strategy, aimed at managing non-positive curvature, is to adopt a local cubic model, which is reduced to a standard quadratic model when only positive eigenvalues are computed [14].

4.2.2 Stochastic framework

As in the deterministic context, also in the stochastic one the computation of the Hessian matrix is very expensive so it makes sense to adopt Limited Memory Steepest Descent (LMSD) type strategies to calculate the best steplengths, also taking into account second order information. The main difference with respect to the deterministic case, lies in the construction of the G_j matrix, where we must replace the full gradients computed at the most recent m iterations ($m \geq 1$) with the stochastic gradients related to the iterate $x^{(j+i)}$, obtained using different data samples $\{\xi^{(n_{j+i})}\}$, $i = 0, .., m-1$:

$$
\overline{G}_j = [g_j^{(n_j)}, g_{j+1}^{(n_{j+1})}, ..., g_{j+m-1}^{(n_{j+m-1})}]. \tag{4.19}
$$

Following the procedure developed in the deterministic case combined with the approximation (4.16), the matrices $\overline{T}_j$ and $\overline{P}_j$ can be computed, by replacing G_j in (4.15) with $\overline{G}_j$.
When the collected stochastic gradients are suitable approximations of the

full gradients, i.e., they are in expectation suitable descent directions at the current iterate with a reduced variance, it is quite likely that from the reciprocals of the eigenvalues of the matrices $\overline{T}_j$ and $\overline{T}_j^{-1}\overline{P}_j$, that are the Ritz-like and harmonic Ritz-like values, useful approximations of the inverse of the local Lipschitz constant of ∇F can be obtained for the new sweep of iterations. For simplicity, we refer in the following to the Ritz-like values, but, similarly, the same considerations hold for harmonic Ritz-like values. Even in the stochastic case we have problems similar to those highlighted in the deterministic case. In particular we may have negative eigenvalues. Also in this case the strategy is to discard them, discarding also the gradients used for their calculation. Consequently, less than m values $\lambda_i,\ i = 1, \ldots,\ m_R \leq m$ may be available. Furthermore, in order to avoid line search techniques, it is convenient to consider only the values λ_i belonging to a prefixed range $[\frac{1}{\alpha_{max}}, \frac{1}{\alpha_{min}})$, where $\alpha_{max} > \alpha_{min} > 0$. In particular, we redefine

$$\lambda_i \leftarrow \max\left(\frac{1}{\alpha_{max}}, \min\left(\lambda_i, \frac{1}{\alpha_{min}}\right)\right), \quad i = 1, ..., m_R, \tag{4.20}$$

and we eliminate the values $\lambda_i = 1/\alpha_{min}$, reducing again m_R and discarding all the stochastic gradients giving rise to these values. The same procedure applies to the harmonic Ritz-like values.
If $m_R = 0$, a tentative steplength $\overline{\alpha} \in (\alpha_{min}, \alpha_{max}]$ can be adopted for a sweep of length 1. This reference value is also used at the first iterate. If $m_R > 0$, the steplengths in the next sweep are defined as

$$\alpha_{j+m+i} = \frac{1}{\lambda_i}, \quad i = 1, ..., m_R. \tag{4.21}$$

A similar procedure involving the harmonic Ritz-like values enables us to define alternatively the steplengths in the next sweep as

$$\alpha_{j+m+i} = \frac{1}{\overline{\lambda}_i}, \quad i = 1, ..., m_H. \tag{4.22}$$

We remark that, in view of Theorem 3.3 in [14], the positive harmonic Ritz-like values are greater or equal than the corresponding Ritz-like values; as a consequence, the rule (4.22) generates shorter steplengths with respect to the ones defined by (4.21). The alternate use of different rules to generate long and short steplengths in the full gradient methods has been deeply investigated (see, for example, [16, 58, 24]), showing a large increase in their practical performance. Also in the stochastic framework, we can explore an alternate use of the Ritz-like and harmonic Ritz-like values. We will propose two different approaches. In a first case the proposal is to alternate the

Ritz-like values to the Ritz-like harmonic values at each sweep (alternative Ritz-like version or A-R version).
The second variant is more complex because it introduces the possibility of modifying the mini-batch size, and linking the choice of one or the other type to this possible increase. The crucial point is to understand when the gradients we are using to calculate the eigenvalues are reliable or not. We will follow the idea proposed in [8] which suggests to increase the size of the sub-sample n_k in case a certain condition is not verified. More precisely, the internal test condition (4.3) can be set to the sample size n_k. Since the term on the left of (4.3) is delimited from above by the true variance of the individual gradient, the (4.3) condition is valid when the following *test of the internal product of the exact variance* is satisfied:

$$\frac{\mathbb{E}[(\nabla f_i(x^{(k)})^T \nabla F(x^{(k)}) - \|\nabla F(x^{(k)})\|^2)^2]}{n_k} \leq \theta^2 \|\nabla F(x^{(k)})\|^4. \tag{4.23}$$

In order to implement condition (4.23), the variance can be approximate with the sample variance

$$\mathrm{Var}_{i \in S_k}(\nabla f_i(x^{(k)})^T \nabla F(x^{(k)}))$$

and the gradient $\nabla F(x^{(k)})$ on the right side with a sample gradient, so that the *approximate inner product test* can be written as follows:

$$\frac{(\sum_{i \in S_k}(\nabla f_i(x^{(k)})^T g_k^{(n_k)} - \|g_k^{(n_k)}\|^2)^2}{n_k(n_k-1)} \leq \theta^2 \|g_k^{(n_k)}\|^4. \tag{4.24}$$

When this condition is not satisfied by the current sample size, the sample size is increased so that (4.24) is satisfied. With regard to the orthogonality test, a sufficient condition for (4.4) is the following *exact variance orthogonality test*:

$$\frac{\mathbb{E}[\|\nabla f_i(x^{(k)}) - \frac{\nabla f_i(x^{(k)})^T \nabla F(x^{(k)})}{\|\nabla F(x^{(k)})\|^2} \nabla F(x^{(k)})\|^2]}{n_k} \leq \nu^2 \|\nabla F(x^{(k)})\|^2. \tag{4.25}$$

As for the previous test (4.23), a practical variant, named *approximate variance orthogonality test*, based on the sample approximation can be formulated as follows:

$$\frac{\sum_{i \in S_k} \|\nabla f_i(x^{(k)}) - \frac{\nabla f_i(x^{(k)})^T g_k^{(n_k)}}{\|g_k^{(n_k)}\|^2} g_k^{(n_k)}\|^2}{n_k(n_k-1)} \leq \nu^2 \|g_k^{(n_k)}\|^2. \tag{4.26}$$

In order to choose a new sample size n_k when the conditions (4.24) and (4.26) are not satisfied, we can compute

$$
\begin{aligned}
Z_1 &= \frac{\mathrm{Var}_{i\in S_k}(\nabla f_i(x^{(k)})^T g_k^{(n_k)}}{\theta^2 \|g_k^{(n_k)}\|^4}, \\
Z_2 &= \frac{\mathrm{Var}_{i\in S_k}(\nabla f_i(x^{(k)}) - \frac{\nabla f_i(x^{(k)})^T g_k^{(n_k)}}{\|g_k^{(n_k)}\|^2} g_k^{(n_k)})}{\nu^2 \|g_k^{(n_k)}\|^2}
\end{aligned} \tag{4.27}
$$

and we set $n_k = \min(\lceil \max(Z_1, Z_2) \rceil, n)$.
Furthermore, when an increase of the sample size occurs, since the previously stored gradients are relative to sub-samples of lower size, they will be discarded. This strategy, called Adaptive Alternation of Ritz-like values (AA-R), leads to shorter steplengths in this transition phase.
Summarising, with the purpose of checking the goodness of the $g_k^{(n_k)}$ estimate of the gradient $\nabla F(x^{(k)})$, the approximate version (4.24) - (4.26) of the internal test is adopted to increase the size of the sub-sample at the current iteration. We have followed this strategy because numerical experience has shown that the growth of the sample size is generally not too fast. This makes it suitable for use in practice. Other approaches can also be found in the literature, as for example the norm test in [11, 13, 28], or the rule based on the matrix Bernstein inequality [53, Theorem. 6.1.1, Corollary 6.2.1]. (see [13, 4]).

4.3 Numerical experiments

In order to evaluate the effectiveness of the proposed steplength rule for SG methods, we consider the optimisation problems arising in training binary and multi-labels classifiers for the following well known datasets:

- the *MNIST* dataset of handwritten digits, categorised in 10 classes (downloadable from `http://yann.lecun.com/exdb mnist`);
- the web dataset *w8a* downloadable from `https://www.csie.ntu.edu.tw/ ~cjlin/libsvmtools/datasets/binary.html`, containing 49749 examples, partitioned in 44774 samples for training and 4975 for testing; each example is described by 300 binary features.

In order to carry out the experiments we faced two categories of functions: convex functions and non-convex functions. In the case of convex problems,

we realised a binary classifier able to distinguish between even and odd digits with the dataset *MNIST* and able to distinguish between the two classes already present in the dataset *w8a*. In the non-convex case, the function to optimise is the one derived from a Convolutional Neural Network of multiple classification, trained on the dataset *MNIST*. We have added a regularisation term for both cases to avoid overfitting.

4.3.1 Convex problems

We built linear classifiers corresponding to three different convex loss functions. Thus the minimisation problem has the form

$$\min_{x\in\mathbb{R}^d} F_n(x) + \frac{\delta}{2}\|x\|_2^2, \tag{4.28}$$

where $\delta > 0$ is the regularisation parameter. By denoting as $a_i \in \mathbb{R}^d$ and $b_i \in \{1, -1\}$ the feature vector and the class label of the i-th sample, respectively, the loss function $F_n(x)$ assumes one of the following form:

- logistic regression (LR) loss:

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} \log\left[1 + e^{-b_i a_i^T x}\right];$$

- square loss (SL):

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} (1 - b_i a_i^T x)^2;$$

- smooth hinge loss (SH):

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} \begin{cases} \frac{1}{2} - b_i a_i^T x, & \text{if } b_i a_i^T x \le 0 \\ \frac{1}{2}(1 - b_i a_i^T x)^2, & \text{if } 0 < b_i a_i^T x < 1 \\ 0, & \text{if } b_i a_i^T x \ge 1. \end{cases}$$

We compare the effectiveness of the following schemes:

- SG with a fixed mini-batch size in the version with fixed steplength, denoted by **SG mini**;

- methods using Ritz-like values to adaptively select a suitable steplength; in particular, we consider:

– Alternate Ritz-like values in the scheme denoted by **A-R**, which toggles the use of the Ritz-like values to the one of the harmonic Ritz-like values at each sweep;

– Adaptive Alternation of Ritz-like values in the scheme denoted by **AA-R**; in this method, when at the iteration k the size of the sample increases, we discard the available Ritz-like values and we exploit the current stored stochastic gradients to determine a set of harmonic Ritz-like values.

For both methods, an adaptive strategy is used for increasing the mini-batch size, as detailed in Section 4.2.2. In all the numerical simulations, we set $\theta = 0.7$ in (4.24) and $\nu = 7$ in (4.26).

Numerical Results

In all the numerical experiments, carried out in Matlab® on 1.6 GHz Intel Core i5 processor, we use the following setting:

- the regularisation parameter δ is equal to 10^{-8};
- in **SG mini** the size of the mini-batch is set as $|S_k| = |S| = 50$ for all $k \geq 0$;
- in **A-R** and **AA-R** methods, the size of the initial mini-batch is $n_0 = 3$; furthermore the maximum length of the sweep is set as $m = 3$;
- each method is stopped after 10 epochs, i.e., after a time interval equivalent to 10 evaluations of a full gradient of F_n or 10 visits of the whole dataset; in this way we compare the behaviour of the methods in a time equivalent to 10 iterations of a full gradient method applied to $F_n(x)$.

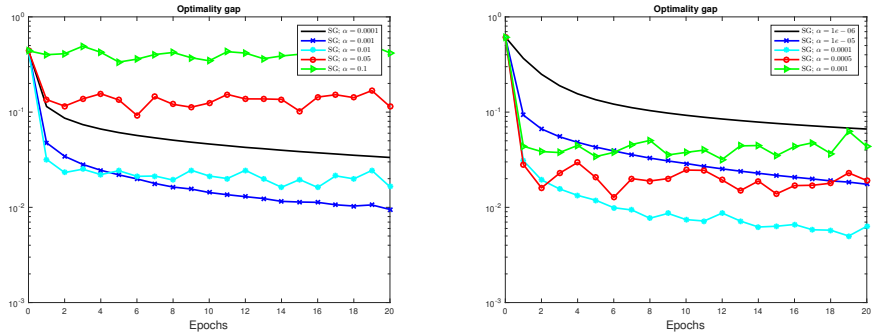
In the following tables and figures, we report the results obtained by the considered methods using the three loss functions (logistic regression, square and smooth hinge loss functions) on *MNIST* and *w8a*. Because of the stochasticity of the methods considered, for each experiment we perform 10 simulations, leaving the pseudo-random number generator free. In this way, considering the averages of the results, we will have more reliable values.
In particular, for any numerical test, we report the following results:

- the average value of the optimality gap $F_n(\overline{x}) - F_*$, where $\overline{x}$ is the iterate obtained at the end of the 10 epochs and F_* is an estimate of the optimal objective value; this value is obtained by a full gradient method with a huge number of iterations;

Table 4.1: Values of the best-tuned steplength α_{OPT} in 20 epochs for the standard SG method in the case of the two datasets and the three loss functions.

	MNIST			*w8a*		
$F_n(x)$	LR	SL	SH	LR	SL	SH
α_{OPT}	10^{-3}	10^{-4}	10^{-3}	10^{-1}	10^{-3}	$5\ 10^{-2}$

Figure 4.1: Behaviour of standard SG in 20 epochs on the *MNIST* dataset, with logistic regression (on the left panel) and square loss (in the right panel).



- the related average accuracy $A(\overline{x})$ at the end of the 10 epochs with respect to the testing set, i.e., the percentage of well-classified examples.

First of all, we determine by a trial procedure the best steplength α_{OPT} for the standard SG method, i.e., the steplength corresponding to the best obtained results. Indeed, following [9], a suitable steplength for **SG mini** is $\alpha_{\text{SG mini}} = |S|\alpha_{OPT}$, where α_{OPT} is a fixed steplength for the standard SG method. We have tried five different steplengths for each combination of standard SG and dataset. In Table 4.1, we report the value of the steplength α_{OPT} corresponding to the best performance of standard SG in 20 epochs. Furthermore, in order to highlight the difficulties to define a suitable learning rate, in Figure 4.1 we show the trend of the optimality gap for five values of the steplengths in the case of *MNIST* dataset with logic regression and square loss functions. The instability of the standard SG method behavior with respect to the selection of the steplength motivates the expensive trial process that produces Table 4.1.
In the following, we report the numerical results of the comparison between **SG mini** and **A-R** and **AA-R** methods. In particular, in **A-R** and **AA-R** methods, different settings of the bounds α_{max} and α_{min} are used:

Table 4.2: Values of the setting providing the best results for **A-R** method.

	MNIST			*w8a*		
$F_n(x)$	LR	SL	SH	LR	SL	SH
$\overline{\alpha}$	10^{-2}	10^{-3}	10^{-2}	1	10^{-2}	$5\ 10^{-1}$
α_{min}	10^{-5}	10^{-7}	10^{-6}	10^{-3}	10^{-6}	$5\ 10^{-4}$
α_{max}	1	$5\ 10^{-2}$	$5\ 10^{-1}$	100	1	25

Table 4.3: Values of the setting providing the best results for **AA-R** method.

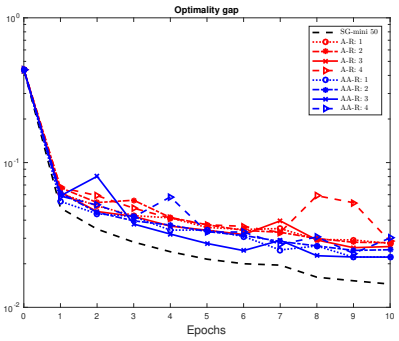
	MNIST			*w8a*		
$F_n(x)$	LR	SL	SH	LR	SL	SH
$\overline{\alpha}$	10^{-2}	10^{-3}	10^{-2}	1	10^{-2}	$5\ 10^{-1}$
α_{min}	10^{-5}	10^{-6}	10^{-5}	10^{-3}	10^{-6}	$5\ 10^{-4}$
α_{max}	1	10^{-1}	1	100	$5\ 10^{-1}$	50

1. $\alpha_{min} = \alpha_{OPT}\ 10^{-2}$, $\alpha_{max} = \alpha_{OPT}\ 500$;
2. $\alpha_{min} = \alpha_{OPT}\ 10^{-3}$ and $\alpha_{max} = \alpha_{OPT}\ 500$;
3. $\alpha_{min} = \alpha_{OPT}\ 10^{-2}$ and $\alpha_{max} = \alpha_{OPT}\ 1000$;
4. $\alpha_{min} = \alpha_{OPT}\ 10^{-3}$ and $\alpha_{max} = \alpha_{OPT}\ 1000$.

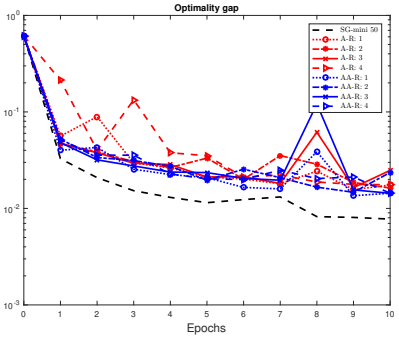
The tentative value of the steplength $\overline{\alpha}$ is set as $10\alpha_{OPT}$.
Figures 4.2 and 4.3 show the behaviour of the optimality gap with respect to the first 10 epochs for *MNIST* and *w8a*, respectively, in the case of the three loss functions. In particular the dashed black line refers to **SG mini** whereas the red and the blues lines are related to **A-R** and **AA-R** methods respectively in the above specified four settings. We observe that the results obtained with the **A-R** and **AA-R** methods are comparable with the ones obtained with the **SG mini** equipped with the best tuned steplength. Indeed, the adaptive steplength rules in **A-R** and **AA-R** methods seem to be slightly dependent on the values of α_{max} and α_{min}, making the choice of a suitable learning rate a less difficult task with respect to the selection of a good constant value in standard SG and **SG mini** methods.

Figures 4.2 and 4.3 show the behaviour of the optimality gap for **SG mini** (dashed black line), **A-R** (red lines) and **AA-R** (blue lines) methods. In the A-R and AA-R cases we plot the results obtained with four different settings:

Figure 4.2: Behaviour of the optimality gap in 10 epochs for **SG mini**, **A-R** and **AA-R** methods in the case of the *MNIST* dataset.



Square loss



Smooth hinge loss

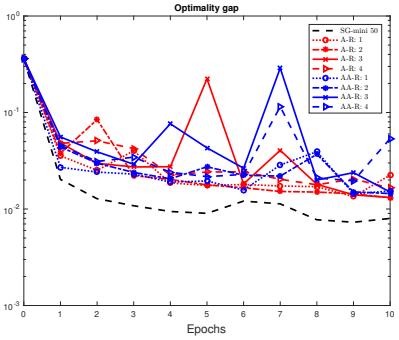
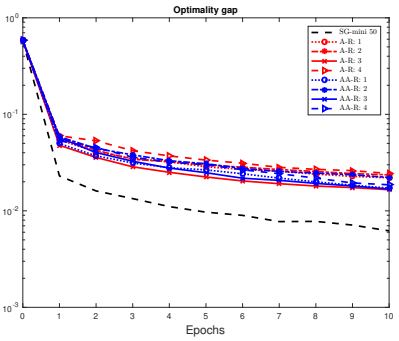
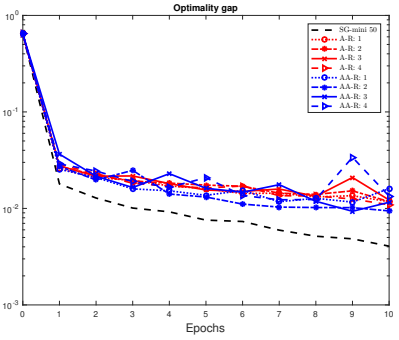


Figure 4.3: Behaviour of the optimality gap in 10 epochs for **SG mini**, **A-R** and **AA-R** methods in the case of the *w8a* dataset.

Logistic regression



Square loss



Smooth hinge Loss

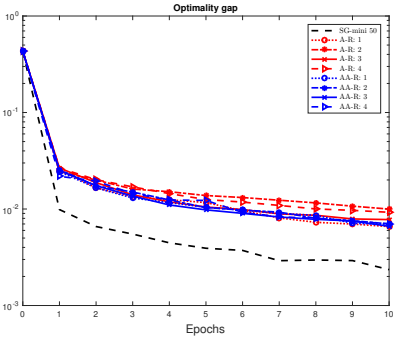


Table 4.4: Numerical results of the considered methods with $F_n(x)$ given by the logistic regression after 10 epochs.

	MNIST		*w8a*	
method	$F_n(\overline{x}) - F_*$	$A(\overline{x})$	$F_n(\overline{x}) - F_*$	$A(\overline{x})$
SG mini	0.0145	0.890	0.0062	0.901
A-R	0.0263	0.890	0.0165	0.903
AA-R	0.0222	0.893	0.0168	0.903

Table 4.5: Numerical results of the considered methods with $F_n(x)$ given by the square loss after 10 epochs.

	MNIST		*w8a*	
method	$F_n(\overline{x}) - F_*$	$A(\overline{x})$	$F_n(\overline{x}) - F_*$	$A(\overline{x})$
SG mini	0.0078	0.892	0.0041	0.890
A-R	0.0163	0.888	0.0109	0.888
AA-R	0.0144	0.890	0.0094	0.888

Table 4.6: Numerical results of the considered methods with $F_n(x)$ given by the smooth hinge loss after 10 epochs.

	MNIST		*w8a*	
method	$F_n(\overline{x}) - F_*$	$A(\overline{x})$	$F_n(\overline{x}) - F_*$	$A(\overline{x})$
SG mini	0.0079	0.897	0.0024	0.907
A-R	0.013	0.896	0.0067	0.904
AA-R	0.0149	0.896	0.0067	0.904

1. $\alpha_{min} = \alpha_{OPT}$ $1e-3$ and $\alpha_{max} = \alpha_{OPT}$ 5;
2. $\alpha_{min} = \alpha_{OPT}$ $1e-4$ and $\alpha_{max} = \alpha_{OPT}$ 5;
3. $\alpha_{min} = \alpha_{OPT}$ $1e-3$ and $\alpha_{max} = \alpha_{OPT}$ 10;
4. $\alpha_{min} = \alpha_{OPT}$ $1e-5$ and $\alpha_{max} = \alpha_{OPT}$ 50.

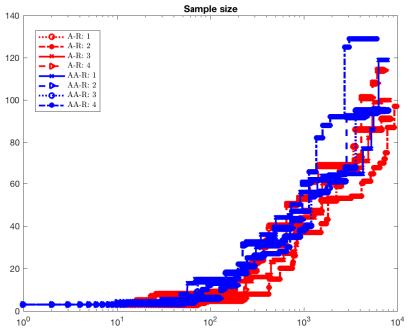
In Tables 4.2 and 4.3, we summarise the setting that provides the best results for **A-R** and **AA-R** methods. In Tables 4.4, 4.5 and 4.6, we show the final optimality gap (with respect to the train set) and accuracy (with respect to the test set) obtained at the end of 10 epochs for the logistic regression, square and smooth hinge loss functions, respectively, for the best setting. The final accuracy of the three methods differs at most to the third decimal digit. This observation can also be extended to the simulations obtained for A-R and AA-R methods with the other settings. In Figures 4.4 and 4.5 we show the increase of the sub-sample size in **A-R** and **AA-R** methods in the case of all the convex loss functions.
Starting with $n_0 = 3$, the size of current sub-sample is at least 120 in the case of *MNIST* dataset and 900 in the case of *w8a* dataset at the end of the 10 epochs, much smaller than the number of sample n of the train set.
Finally, in Figures 4.6 and 4.7, we compare the behaviour of **SG mini** and **A-R** and **AA-R** methods when the parameter α_{SGmini} is not the best-tuned value, as in the previous experiments.
In particular, **SG mini** method in Figure 4.6 is carried out with α_{OPT} replaced by $\alpha = 10^{-5}$ for logistic regression function and $\alpha = 10^{-5}$ for smooth hinge loss function, that is $\alpha_{SGmini} = \alpha|S|$. In Figure 4.7, **SG mini** is equipped with $\alpha = 1$ for logistic regression function and $\alpha = 10^{-5}$ for square loss function. **A-R** and **AA-R** methods are executed using the four previously specified settings, with α_{OPT} set as above.

Figures 4.6 and 4.7 highligth that a too small fixed steplength in SG mini produces a slow descent of the optimality gap; on the other hand, a steplength value larger than the best-tuned one can cause oscillating behavior of the optimality gap and, sometimes, it does not guarantee the convergence of **SG mini** method. As regard **A-R** and **AA-R** methods, these approaches appear less dependent on an optimal setting of the parameters and they enable us to obtain smaller optimality gap values after the same number of epochs exploited by **SG mini**.
Furthermore, we observe that in **A-R** method, the behaviour of the optimality gap is more stable than in **AA-R** method. Nevertheless, **AA-R** method can produce a smaller optimality gap at the end of 10 epochs. In conclusion,

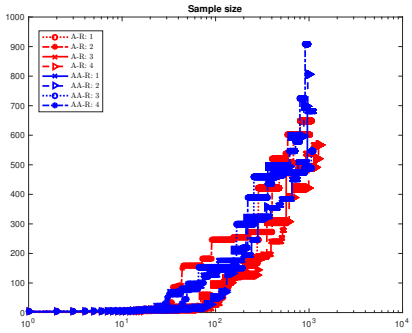
Figure 4.4: Mini-batch size in **A-R** and **AA-R** methods on the *MNIST* dataset with respect to the iterations.

Figure 4.5: Mini-batch size in **A-R** and **AA-R** methods on the *w8a* dataset with respect to the iterations.

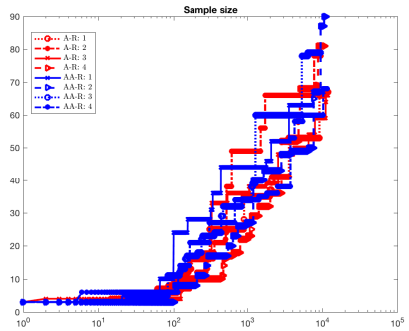
Logistic regression



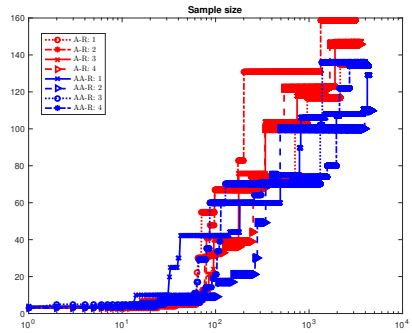
Logistic regression



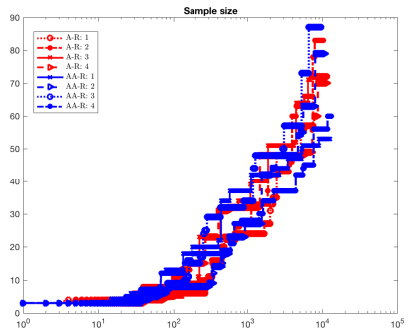
Square loss



Square loss



Smooth hinge loss



Smooth hinge loss

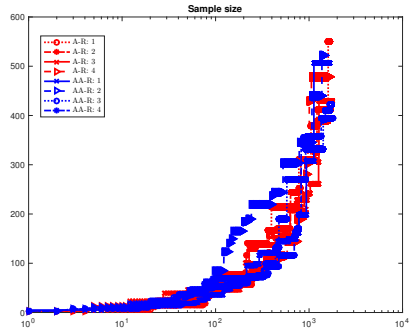
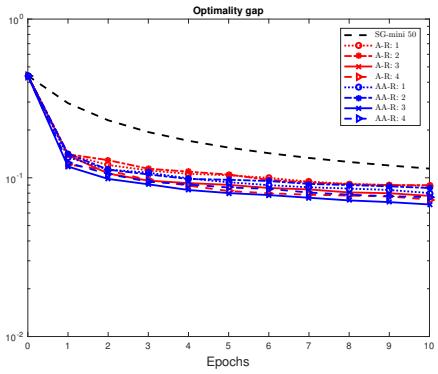


Figure 4.6: Comparison between **SG mini** with respect to **A-R** and **AA-R** in 10 epochs on the *MNIST* dataset.

(a) Logistic regression loss with smaller steplength than the best-tuned



(b) Smooth hinge loss with smaller steplength than the best-tuned

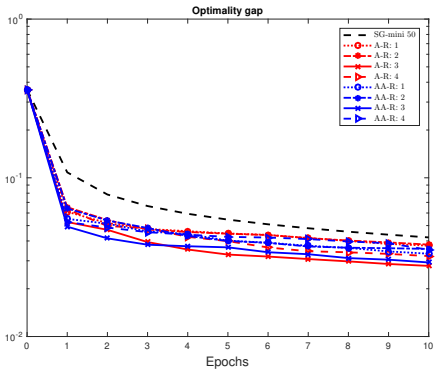
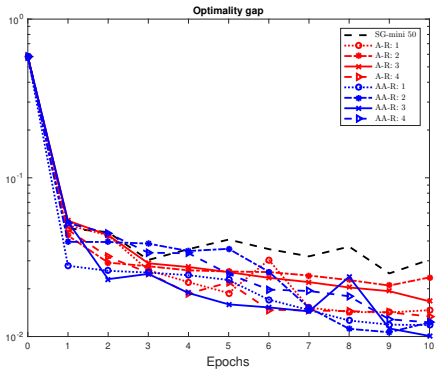
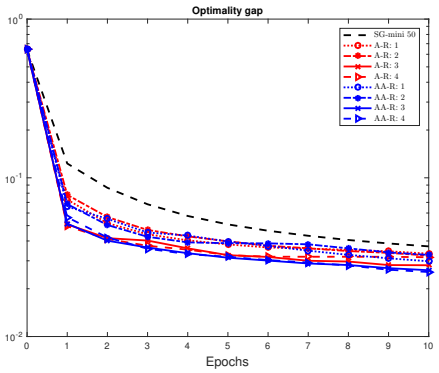


Figure 4.7: Comparison between **SG mini** with respect to **A-R** and **AA-R** in 10 epochs on the *w8a* dataset.

(a) Logistic regression loss with greater steplength than the best-tuned



(b) Square loss with smaller steplength than the best-tuned



the optimal steplength search process for the SG method is often computationally long and expensive, since it requires a trial-and-error approach. On the other hand, the methods equipped with the Ritz-like values and an adaptive rule appear to be weakly affected by the definition of its working interval.

4.3.2 Some further experiments and remarks on convex problems

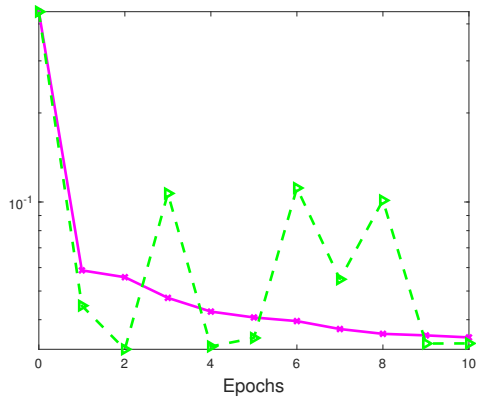
In the previous experiments, **A-R** and **AA-R** methods are equipped with the approximated version of the augmented inner test, based on sample statistics. For small samples, the conditions (4.24)-(4.26) may not be reliable enough in providing a sample size able to control the errors in the gradient estimates; indeed, in presence of noise, the norm of the current stochastic gradient $g_k^{(n_k)}$ can be greater than $\|\nabla F(x^{(k)}\|$, so that the conditions (4.24)-(4.26) are verified for many iterations before producing an increase in the sample size. To prevent this drawback, in [8], when the sample size does not change for at least r consecutive iterations, an average vector of the last r sample gradients is computed:

$$g_{avg} = \frac{1}{r} \sum_{j=k-r+1}^{k} g_j^{(n_j)}. \tag{4.29}$$

When $\|g_{avg}\| < \gamma \|g_k^{(n_k)}\|$, for a prefixed $\gamma \in (0, 1)$, the augmented inner product test is performed by replacing the current stochastic vector with the average vector g_{avg}; the possible consequence is an increase of the sample size. Typical values for r and γ are 10 and 0.38 respectively. For more details, also on this special setting, see [8]; here, this practical procedure is viewed as a *recovery strategy* to improve the stability of **SG** method equipped with a line search rule for providing a suitable steplength. On the other hand, after some epochs, the effectiveness of the method can degrade for faster increase of the sequence $\{n_k\}$, although the adoption of the recovery procedure makes smaller the total number of backtracking steps [54].

In order to highlight this remark, in Figure 4.8 we shows the results obtained for *MNIST* when the problem (4.28) with logistic regression function is addressed by **SG** method equipped with a simple line search. In particular, we report the optimality gap with respect to 10 epochs when the augmented inner product test is coupled with the recovery procedure (magenta line) and without this recovery procedure (green line). In the latter case, the final sample size is 48 with a large number of backtracking steps (2700), while in the

Figure 4.8: Behaviour of the optimality gap in 10 epochs for **SG** method equipped with a line search rule; magenta line is related to the version of the method combined with the recovery procedure while the green line is used for the version without this procedure. The parameters are chosen as in [8]. In the experiment, logistic regression is the loss function and *MNIST* is the dataset.

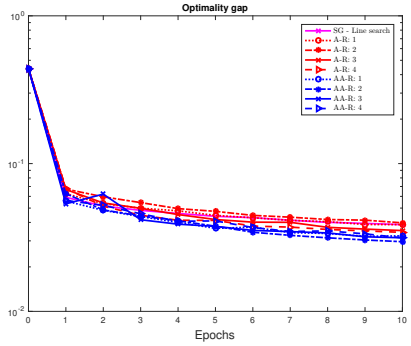


former one the sample size increases until 3300 with very few backtracking steps (110). As a consequence, the recovery procedure appears crucial for the control of the effectiveness of the line search and the sequence $\{n_k\}$.
The numerical results of the previous section show that **A-R** and **AA-R** methods are less dependent on the lack of reliability of the augmented inner product test for small values of n_k. Nevertheless, we can introduce the recovery procedure when the computation of Ritz-like values gives rise to $m_R = 0$ and the steplength at the next iteration is set to a tentative value $\overline{\alpha}$. More precisely, when this situation occurs, if the sample size has not changed in the last r iterations, the novel sample size is determined by using the approximated augmented inner product test with $g_k^{(n_k)}$ replaced by the average vector (4.29). In Figures 4.9 and 4.10, we show the behaviour of the optimality gap obtained by using the modified versions of **A-R** and **AA-R** methods and **SG** method equipped with the line search rule for *MNIST* and *w8a*, respectively, in the case of the three loss functions in the objective. The comparison with Figures 4.2 and 4.3 allows to observe that the recovery procedure improves the stability of **A-R** and **AA-R** methods with respect to the setting of α_{min} and α_{max}, preserving the effectiveness of the approach. Indeed the accuracy of the two versions at the end of 10 epochs differs at most to the third decimal digit. The final value of the sample size is at most 10 times the one obtained without the use of the recovery procedure. As already observed in the previous section, **AA-R** method allows to obtain

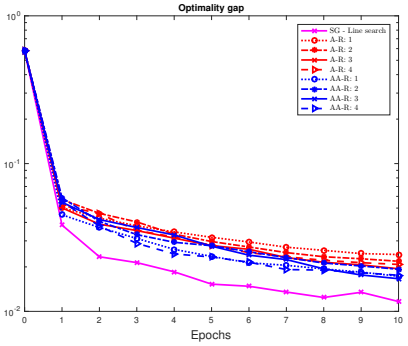
Figure 4.9: Behaviour of the optimality gap in 10 epochs for the versions of **A-R** and **AA-R** methods using with the recovery procedure and **SG** equipped with a line search rule in the case of the *MNIST* dataset.

Figure 4.10: Behaviour of the optimality gap in 10 epochs for the versions of **A-R** and **AA-R** methods using with the recovery procedure and **SG** equipped with a line search rule in the case of the *w8a* dataset.

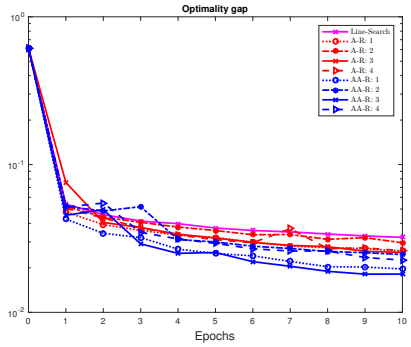
Logistic regression



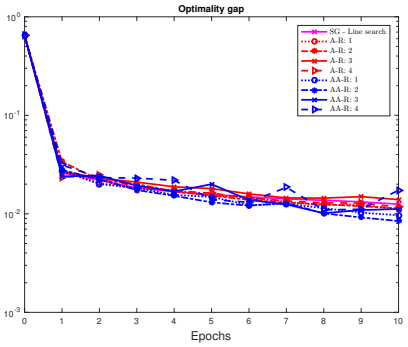
Logistic regression



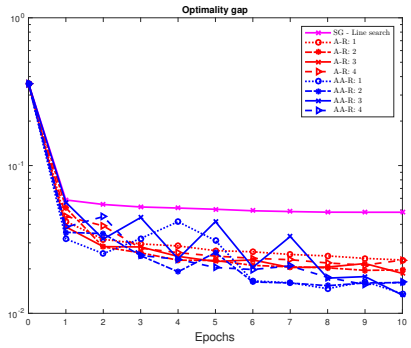
Square loss



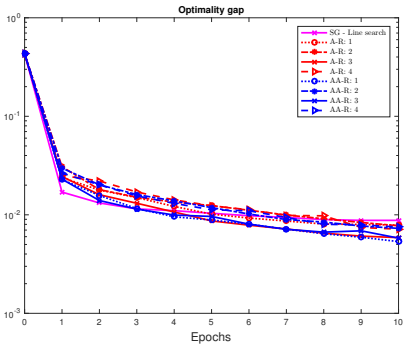
Square loss



Smooth hinge loss



Smooth hinge Loss



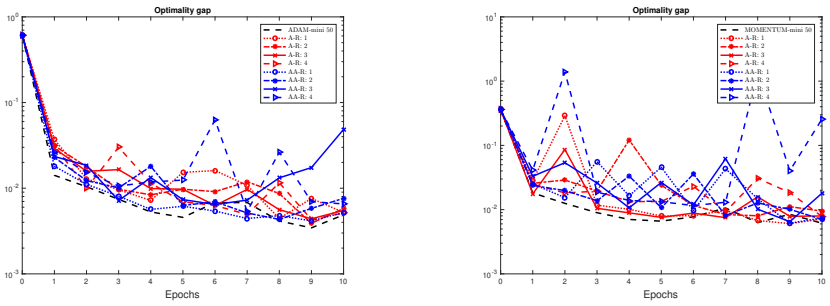
better results with respect to **A-R** in most experiments.
Furthermore, we observe that the performance of our approach appears generally better with respect to **SG** with a line search procedure. The comparison is carried out by considering only the number of scalar products performed in all methods, that is n scalar products for each epoch. Indeed, for the considered loss functions, the computational cost of the evaluation of the stochastic gradient $g_k^{(n_k)}$ is essentially given by the n_k scalar products $a_i^T x^{(k)}$, $i \in S_k$.
As regard **SG** method with a line search rule, we observe that, although the evaluation of an estimate of the objective function $\frac{1}{n_k}\sum_{i\in S_k} f_i(x^{(k)})$ at $x^{(k)}$ does not require additional scalar products and it is negligible, the computation of the same estimate at $x^{(k)} - \alpha g_k^{(n_k)}$ requires at least additional n_k scalar products. Thus, each iteration of **SG** with line search has a computational cost at least equal to two evaluations of the stochastic gradient on the same sample. Any backtracking step increases the count of total scalar products. This preliminary analysis appears to favor schemes that avoid a line search rule for the determination of the steplength, also in the case of a few epochs when the sample size remains low. This topic may be the subject of future investigations.

Momentum and AdaM

In this section we describe the numerical results obtained by inserting the proposed steplength selection rules combined with the adaptive increase of mini-batch size in two stochastic schemes, as Momentum and AdaM. In particular, referring the same problem of compute a linear classifier for MNIST dataset using the three convex loss functions LR, SL and SH, we compare the effectiveness of the following schemes: 1) SG, Momentum and AdaM [36] with a fixed mini-batch size and a fixed steplength; 2) **A-R** and **AA-R** schemes combined with Momentum or AdaM. These numerical experiments are carried out in Matlab® on 1.6 GHz Intel Core i5 processor. We decided to conduct the experiments on a maximum of 10 epochs.

The Figure 4.11 shows the behaviour of the objective function optimality gap, in the case of LR with AdaM method and SH with Momentum method. In particular the dashed black line refers to **ADAM-mini** and **MOMENTUM-mini** (with best-tuned steplength), whereas the red and the blue lines are related to **A-R** and **AA-R** versions of the two methods respectively. These results highlight that the results obtained with the **A-R** and **AA-R** methods are comparable with the ones obtained with the mini-batch standard versions equipped with the best-tuned steplength. Both **A-R** and **AA-R** methods have been tested on 4 different settings of $(\alpha_{min}, \alpha_{max}]$:

Figure 4.11: Optimality gap in 10 epochs for **A-R** and **AA-R** methods: on the left panel, comparison with **ADAM-mini** on the LR loss function and, on the right panel, comparison with **MOMENTUM-mini** on the SH loss function.



1. $(10^{-5}, 0.5]$,
2. $(10^{-6}, 0.5]$,
3. $(10^{-5}, 1]$,
4. $(10^{-6}, 1]$.

It is important to remark that the adaptive steplength rules in **A-R** and **AA-R** methods seem to be slightly dependent on the values of α_{max} and α_{min}, making the choice of a suitable learning rate a less difficult task with respect to the selection of a good constant value in standard methods. The final accuracies obtained by the methods on the testing set differ at most to the third decimal digit.

4.3.3 A non-convex problem: a Convolutional Neural Network

In the non-convex case we consider as loss function an Artificial Neural Network. In particular, dealing with image classification, we consider a Convolutional Neural Network (CNN). The network is composed of an input layer, two sequences of convolutional and max-pooling layers, a fully connected layer and an output layer. We make use of Rectified Linear Unit (ReLU) activations combined by a softmax function for the output layer and of a cross entropy as loss function (see Figure 4.12). We consider the optimisation problem arising in training a multi-class classifier for the *MNIST* data set.

Figure 4.12: Artificial Neural Network structure

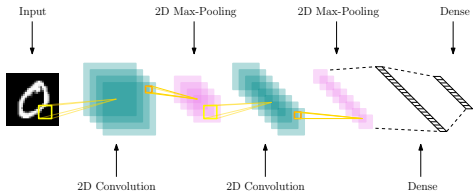
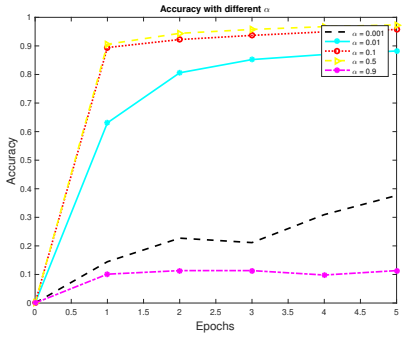


Figure 4.13: CNN Accuracy in the SG mini case



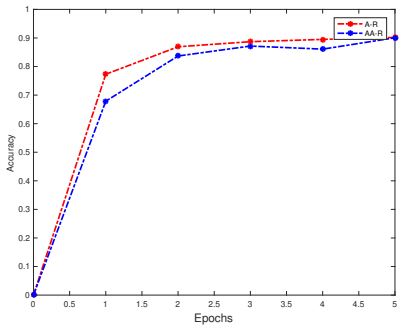
We compare the effectiveness of the same methods considered in the previous section, i.e., **SG mini**, **A-R** and **AA-R** methods. In all the numerical experiments we use the following setting:

- regularisation parameter $\delta = 10^{-4}$;
- the first convolutional layer is composed by 64 filters, each filter has 5×5 dimension; after we apply a max-pooling of size 2×2;
- the second convolutive layer is composed by 32 filters, each filter has 5×5 dimension; after we apply a max-pooling of size 2×2;
- in **SG mini**, the size of the mini-batch, is set as $|S| = 50$;
- in **A-R** and **AA-R** methods, the length of any sweep is at most $m = 3$; furthermore, $\theta = 0.7$ in (4.24) and $\nu = 7$ in (4.26) for all the numerical simulations;

The numerical experiments were carried out in Matlab® on Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz with 8 CPUs.

In the Figure 4.13 we can observe the different accuracies (with respect to the testing set) provided by the CNN trained with **SG mini** in 5 epochs; the fixed steplength is set to values between 0.001 and 0.9. As we can see, the

Figure 4.14: Accuracy obtained by training the CNN with **A-R** and **AA-R** methods.



method provides effective results for $\alpha = 0.5$; a similar accuracy is obtained for $\alpha = 0.1$. In cases with smaller steplengths, the accuracy in 5 epochs is unsatisfactory, while a higher steplength can lead to the divergence of the method. Hence, in a more marked way than the convex case, for non-convex problems, finding an effective steplength requires a very expensive trial procedure. In Figure 4.14 we report the results obtained by training the CNN with the **A-R** and **AA-R** methods. In particular we show the behaviour of the accuracy with respect to the testing set in the first 5 epochs with the following settings:

- for **A-R** method, $\alpha_{min} = 10^{-3}, \alpha_{max} = 1$, $n_0 = 10$;
- for **AA-R** method, $\alpha_{min} = 10^{-2}, \alpha_{max} = 1$, $n_0 = 3$.

The parameter $\overline{\alpha}$ is set as 0.1 in all cases. We observe that **A-R** appears more robust with respect to the amplitude of the interval where α_k can belong. Furthermore, we notice that the sub-sample size increases up to a maximum of 204 and 182 in **A-R** and **AA-R** methods respectively.

Momentum and AdaM

Also for the CNN we perform experiments with Momentum and AdaM optimiser.
The setting for Momentum is:

- $\beta = 0.9$;
- $|S| = 50$ sub-sample size.

The setting of **A-R** and **AA-R** versions of Momentum is:

Table 4.7: Numerical results of the considered methods with Momentum optimiser after 5 epochs.

α	**MOMENTUM-mini**	α_{max}	**A-R**	**AA-R**
0.01	0.8819	0.8	0.8783	0.9182
0.1	0.9573	0.9	0.8675	0.8829
0.5	0.9708	1	0.902	0.9269
0.9	0.0958	1.2	0.8733	0.8644

Table 4.8: Accuracies of the considered methods with AdaM optimiser after 5 epochs.

α	**ADAM-mini**	α_{max}	**A-R**	**AA-R**
0.001	0.9705	0.1	0.8768	0.9061
0.01	0.9492	0.3	0.9557	0.9333
0.1	0.1148	0.5	0.8537	0.8372

- for **A-R** method, $\alpha_{min} = 10^{-3}$, $n_0 = 10$;
- for **AA-R** method, $\alpha_{min} = 10^{-3}$, $n_0 = 10$.

The setting for AdaM is:

- $\beta_1 = 0.9$ and $\beta_2 = 0.999$;
- $\epsilon = 1e - 8$;
- $|S| = 50$ sub-sample size.

The setting of **A-R** and **AA-R** versions of AdaM is:

- for **A-R** method, $\alpha_{min} = 10^{-3}$, $n_0 = 10$;
- for **AA-R** method, $\alpha_{min} = 10^{-3}$, $n_0 = 10$.

The conclusions are similar to the previous cases. As we can see in Table 4.7 if in **MOMENTUM-mini** the steplength is too big, for example 0.9, the method diverges. The same remark can be repeated for AdaM; in Table 4.8 we can see a bad accuracy result for $\alpha = 0.1$.

Table 4.9: Sub-sample size of the considered methods with Momentum and AdaM optimiser after 5 epochs.

	MOMENTUM-mini			**ADAM-mini**	
α_{max}	**A-R**	**AA-R**	α_{max}	**A-R**	**AA-R**
0.8	327	306	0.1	182	204
0.9	214	294	0.3	263	244
1	204	155	0.5	226	363

Chapter 5

Meta-learning

The performance of the ANNs and more generally of the machine learning algorithms have shown great potential, especially in recent years. The fields in which these technologies are applied are many: from medicine to autonomous driving, from the study of time series in industrial processes to climate forecasts of rare events. Despite the vast state of the art on the subject, designing a new machine learning methodology, especially in Deep Learning field, remains a very difficult task. Great part of the difficulty lies in designing the structure of the method by setting hyperparameters such as: steplength, mini-batch size, type of optimiser, type of layer, number of neurons per layer, dropout rate. The different combination of these hyperparameters can make the difference between a mediocre performance and an accuracy comparable to the state of the art. The search for the optimal configuration is done by maximising some type of measurement that quantifies the performance of the network. In the case of classification problems this measurement can be the accuracy of the method, defined as the percentage of well classified cases by the methodology with respect to the total cases. This process of maximisation is made particularly difficult by the fact that, in general, this phase is particularly expensive from a computational point of view.
Starting from the observations just made, what we have developed is an offline method to predict the accuracy of a new machine learning algorithm. The proposed method allows to predict the behaviour of an ANN by avoiding excessive costs, since it bases the prediction on the first phase of the learning process, without having to lead the algorithm to convergence. To realise this prediction we based ourselves on two methodologies at the same time: Support Vector Machine for Regression (SVR) and curve fitting. These methodologies can be particularly useful to predict the performance of an ANN from the earliest epochs and to look for the configuration of optimal

parameters. The algorithm we propose may be of particular interest for a fairly fast quality assessment of a new learning algorithm. In particular, it can facilitate the optimisation of hyperparameters in an interesting way.

5.1 Methodology

The proposed method is particularly complex. Before describing and motivating the method, let's summarise the steps that form its structure. First of all we will explain how to create a dataset of examples for our purpose. In a second phase this dataset will be used to simultaneously train an SVR and a curve-fitting methodology; their union will be able to predict the final accuracy of a learning methodology starting from the performance provided in the early epochs. In the third phase it will be illustrated how this predictive tool will be used to optimise the hyperparameters of a network.
The first step to apply the method is to create a dataset of examples. The dataset must be as meaningful and complete as possible with respect to the hyperparameter space we have chosen. Therefore, first of all, we have to fix some hyperparameters and allow others to vary to define the space of the possible configurations. At this point, we try to sample the space as exhaustively as possible, creating all the possible configurations of the ANNs (or other methodology). In the space we choose some configurations; the choice of the percentage of samples to include in our dataset may be subject to time or hardware constraints. We take into account that the dataset creation phase is the most expensive phase of the methods. Once the samples that will constitute the dataset are chosen, their elements are trained until numerical convergence, thus collecting all the features that identify the examples associated with their accuracy. In particular, the accuracy at each epoch is also collected during the training phase. Training is carried out on the training set and accuracy is evaluated using the testing set.

5.1.1 Between SVM and curve fitting

Given any iterative Machine Learning methodology, the main aim of the method is to provide what its behaviour at convergence will be, based only on the performance relative to the initial training phase. To achieve this ambitious goal we have combined two methodologies well known in the literature: Support Vector Machine for Regression (SVR) [10] and curve fitting [48].̇We first present a theoretical background of the mentioned techniques, and then indicate how we used them.
The SVM methodology can be included in the supervised learning, where

the data are labelled. In case of binary classification problems, the basic idea of the methodology is to devise a classifier, named SVM classifier, given by a separating hyperplane (or separating hypersurface) through the selection of the most significant examples of the dataset, called support vectors. The optimal separating hyperplane provided by SVM is then used to classify new examples. In its simplest formulation, SVM defines a linear classifier but it can be generalised to obtain non-linear classifiers by exploiting the so-called kernel trick. By mapping the examples into a suitable high dimensional feature space and by looking for the SVM linear classifier in that space, the user defines a non-linear decision function in the original input space. To work in the new feature space, only the scalar product between examples in the feature space is necessary and this can be expressed by suitable kernel functions. Furthermore there is a version called SVM for Regression (SVR) which, using the similar principles, constitutes a methodology for regression. One of the most important features of SVM, both in classification and regression, is that the solution is represented by using only the examples belonging to a subset of the original training set and this can introduce significant cost reduction from a computational point of view.

SVM and SVR

We recall in a quick way the concepts related to SVM and SVR. Let's start by recalling the SVM linear classifier for binary classification. The decision function is represented by:

$$f(\mathbf{x}) = sign(\mathbf{w^T x} + b) \tag{5.1}$$

and the formulation of the problem is:

$$min \quad \frac{1}{2} \parallel \mathbf{w} \parallel + C \sum_{i=1}^{n} \xi_i \tag{5.2}$$

$$\text{s.t.} \quad y_i(\mathbf{w^T x_i} + \mathbf{b}) \geq 1 - \xi_i, \quad i = 1, \ldots, n \tag{5.3}$$

$$\xi_i \geq 0, \quad i = 1, \ldots, n$$

where $\mathbf{x}_i$ are the samples of the dataset and y_i the related labels, ξ_i are the slack variables and C is a constant that controls the trade-off between the error, on the training set, and the complexity of the model. Therefore $\mathbf{w}$ and $\mathbf{b}$ are learned from the method. The corresponding dual formulation of the

problem is:

$$\begin{aligned} \min_{\alpha \in \mathbb{R}^n} \quad & \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j - \sum_{i=1}^{n} \alpha_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, n \end{aligned}$$

where α_i are the dual variables. In this case the binary classifier results:

$$f(\mathbf{x}) = sign\left(\sum_{i=1}^{n} \bar{\alpha}_i y_i \mathbf{x}_i^T \mathbf{x} + \bar{b} \right) \tag{5.4}$$

where $\bar{\alpha}_i$ are the solution of the dual problem and $\bar{b}$ is obtained from Karush-Kuhn-Tucker conditions.
In the non-linear case, the dual formulation can be expressed as

$$\begin{aligned} \min_{\alpha \in \mathbb{R}^n} \quad & \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) - \sum_{i=1}^{n} \alpha_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, n \end{aligned}$$

where $K(\mathbf{x}_i, \mathbf{x}_j)$ is the kernel function.
In this case, the binary classifier results:

$$f(\mathbf{x}) = sign\left(\sum_{i=1}^{n} \bar{\alpha}_i y_i K(\mathbf{x}_i, \mathbf{x}) + \bar{b} \right) \tag{5.5}$$

where $\bar{\alpha}_i$ are the solution of the dual problem and $\bar{b}$ is obtained from Karush-Kuhn-Tucker conditions.
To understand how SVM works, we reformulate the problem (5.2), by introducing the definition of loss function. The loss function $V(y, f(\mathbf{x}))$ measures how well f estimates the input $\mathbf{x}$. Thus, we can determine the decision function by solving

$$\min_{f \in H} \quad \| f \|_H^2 + \frac{C}{n} \sum_{i=1}^{n} V(y_i, f(\mathbf{x}_i))$$

where the regularisation parameter C handles the trade-off between the complexity of the solution and the errors on the training set; H represents the

Hypothesis space where f must be chosen.
The SVM binary classifier can be obtained by using the soft-margin loss function, defined as:

$$V(y, f(\mathbf{x})) = \max(0, 1 - yf(\mathbf{x})) = \begin{cases} 0, & \text{if } yf(\mathbf{x}) \geq 1 \\ 1 - yf(\mathbf{x}), & \text{if } yf(\mathbf{x}) < 1 \end{cases}$$

In case of regression problems, the SVR prediction function is obtained by exploiting the $\epsilon-$insensitive loss function, whose formulation is:

$$V_\epsilon(y, f(\mathbf{x})) = |y - f((x))|_\epsilon = \max(0, |y - f(\mathbf{x})| - \epsilon)$$

As a consequence, the primal problem can be formulated as follows:

$$\min \frac{1}{2} \parallel \mathbf{w} \parallel^2 + C \sum_{i=1}^{n} V_\epsilon(y_i, \mathbf{w^T x_i} + b)$$

while the dual formulation can be written as

$$\begin{aligned} \min_{\alpha, \alpha^* \in \mathbb{R}^n} \quad & \frac{1}{2} \sum_{i,j=1}^{n} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) \mathbf{x}_i^T \mathbf{x}_j + \epsilon \sum_{i=1}^{n} (\alpha_i + \alpha_i^*) - \sum_{i=1}^{n} y_i (\alpha_i - \alpha_i^*) \\ \text{s.t.} \quad & \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) = 0 \\ & 0 \leq \alpha_i \leq C, 0 \leq \alpha_i^* \leq C \quad i = 1, \ldots, n \end{aligned}$$

In this case the prediction function is

$$f(\mathbf{x}) = \sum_{i=1}^{n} (\bar{\alpha}_i - \bar{\alpha_i}^*) \mathbf{x}_i^T \mathbf{x} + \bar{b} \tag{5.6}$$

where $\bar{b}$ is obtained from Karush-Kuhn-Tucker conditions.
Both in the optimisation problem and in the prediction function, the vectors of the training set appear only inside inner products; therefore, the algorithm can be generalised to Hypothesis spaces of non-linear functions through a kernel. An example is the polynomial kernel $K(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + 1)^s$, where s is the polynomial degree or the Gaussian kernel $K(\mathbf{x}, \mathbf{x}') = \exp(- \parallel \mathbf{x} - \mathbf{x}' \parallel^2 / (2\sigma^2))$, where σ is the variance.
The dual formulation of the non-linear SVR problem has the form:

$$\begin{aligned} \min_{\alpha, \alpha^* \in \mathbb{R}^n} \quad & \frac{1}{2} \sum_{i,j=1}^{n} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) K(\mathbf{x}_i, \mathbf{x}_j) + \epsilon \sum_{i=1}^{n} (\alpha_i + \alpha_i^*) - \sum_{i=1}^{n} y_i (\alpha_i - \alpha_i^*) \\ \text{s.t.} \quad & \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) = 0 \\ & 0 \leq \alpha_i \leq C, 0 \leq \alpha_i^* \leq C \quad i = 1, \ldots, n \end{aligned}$$

In this case the prediction function results:

$$f(\mathbf{x}) = \sum_{i=1}^{n} (\bar{\alpha}_i - \bar{\alpha}_i^*) K(\mathbf{x}_i, \mathbf{x}) + \bar{b} \tag{5.7}$$

where $\bar{b}$ is obtained from Karush-Kuhn-Tucker conditions.
The use of the ϵ-intensive loss function ensures the existence of a global minimum. The ϵ parameter controls the width of the ϵ-insensitive zone, used to fit the training data.

Curve-fitting

A drawback in the use of SVR is that its output values can be outside the interval $[0, 1]$. Indeed, the predicted values denote the accuracy of a learning algorithm and, consequently, the output values of the method must be strictly positive and less than one. Therefore to better manage the constraints on the accuracy, we have also introduced the curve-fitting methodology. Curve-fitting is the process that enables to determine the curve that best fits a series of data points, satisfying a set of constraints. In this case, given a family of functions dependent on a set of parameters, the goal is to find the parameters defining the function that allow you to best describe the data to be approximated. The values of these parameters are usually calculated using the method of Least Squares (LS), which minimises the sum of the square errors between the original data and the model values. Also with this approach, it is possible to define non-linear prediction function. If the curve is linear with respect to the parameters, then the problem reduces to solving a system of linear equations. This is referred to as linear least squares.

In the general formulation of least squares, we have to solve a non-linear system of equations. Indeed, we have to minimise the sum of the square of the deviations between the curve-fit and n actual data points $(epoch_i, acc_i)$, $i = 1, ..., n$:

$$LS = \sum_{i=1}^{n} (g(epoch_i) - acc_i)^2 \tag{5.8}$$

where $g(epoch_i)$ is the curve-fit function, which may depend non-linearly on the parameters (non-linear LS). In our case the data points are represented to the first epochs $epoch_i$ and the related accuracies acc_i. Setting the gradient of the objective function with respect to the parameters, denoted by $\gamma_1, \gamma_2, \ldots$, equal to 0, we obtain a non-linear system of equations:

$$\nabla LS(\gamma_1, \gamma_2, ...) = 0. \tag{5.9}$$

Non-linear least squares is significantly more difficult to solve than linear least squares, because of the difficulties associated with solving sets of non-linear equations.

Regression problems, linear and non-linear, can be single variable or multi-variable. A difficulty of regression is that the functional relationships must be specified a priori, and this difficulty may be increased for multi-variable regression.

In this work, the fitting function $g : \mathbb{R} \to \mathbb{R}$, is chosen to have the form:

$$g(epoch) = \alpha \; epoch^{\beta} \tag{5.10}$$

where α and β are the parameters to be calculated from non-linear LS.
In our case the variable *epoch* represents the epoch number of interest and $g(epoch)$ is the related accuracy.

5.1.2 Role of SVR and curve-fitting in the proposed methodology

After the first step aimed to generate a suitable database, an SVR is trained on this database. As input features of the SVR, in addition to the characteristics of the method, i.e., the hyperparameters, there will also be the accuracy of the early epochs; the final accuracy represents the label of the example. At this point, the trained SVR methodology will be able to predict the final accuracy of a method given its performance in the initial phase of the learning process. Therefore, having both the results of SVR and curve-fitting available, we must have a procedure to choose which one to use. In general, when the accuracy constraints are respected, SVR prediction is used; if constraints are not respected, curve-fitting is used. There are two constraints to respect. Taking into account that what we are predicting is an accuracy, this value must always be positive and less than 1. The second constraint comes from the fact that the final accuracy must always be greater than or equal to the best accuracy seen in early training epochs. When the SVR prediction is greater than 1, for the final accuracy prediction we will use the curve-fitting methodology, using (5.10) as prediction function. In this case, the parameters α and β are computed and the obtained function g is used to predict the value of the accuracy at convergence of the methodology under examination. The parameters of the curve-fitting methodology are subject to the following constraints:

$$\begin{cases} \frac{acc\_max}{fin\_epoch} \leq \alpha \\ 0 \leq \beta \leq 1 \end{cases} \tag{5.11}$$

where acc_max is the maximum value of the accuracy in the early epochs and fin_epoch is the maximum number of epochs that can be performed by the learning algorithm. In particular, the first constraint forces the curve to reach an accuracy at the final epoch probably higher than that already observed in the first epochs. With these constraints, the methodology can be useful to discriminate between promising models and models that already show poor performance at the early epochs.
The method described above can be particularly convenient from a computational point of view. This is because at the pure cost of creating the database, it generates a tool capable of predicting the accuracy of a new methodology based only on its initial training phase. An advantage of the method is its great scalability. The dataset that must be created does not depend on the complexity of the problem nor on the time taken by the single training (which can often be variable).

5.2 Application: from prediction to hyperparameter optimisation

Although the method is suitable for any Machine Learning methodology from now on we will focus on ANNs. In particular we will discuss how to set some of the hyperparameters of a CNN. So our goal will be to apply the method in the context of Hyperparameter Optimisation (HO).

5.2.1 Other approaches

In addition to the information provided in Chapter 2, we recall two of the most common approaches to tackling the HO problem. The most intuitive approaches are grid search and random search [31]. Both methods require to drive many ANNs to convergence, making the methods very expensive from the computational point of view. As a consequence, Sequential Model-Based Optimisation (SMBO) [30] algorithms have been employed in many settings when the performance evaluation of a model is expensive. They approximate the black-box objective function *accuracy* that has to be maximised by a surrogate function, cheaper to evaluate. At each iteration of the algorithm, the new point where the surrogate has to be evaluated is chosen by maximising a chosen criterion. Several SMBO algorithms have been proposed in the literature, and differ in the criteria by which they optimise the surrogate, and in the way they model the surrogate given the observation history. Two of the most famous SMBO approaches are the Bayesian Optimisation approach [44, 43] and the Tree-structured Parzen Estimator strategy [5].

More recently, new hyperparameter optimisation methods based on reinforcement learning have emerged [2, 3, 33, 12]. The goal for most of them was to find the ANN or CNN architectures that are likely to yield an optimised performance. Thus, they were seeking the appropriate architectural hyperparameters, as the number of layers or the structure of each convolutional layer. In this analysis many other hyperparameters, such as the learning rate and regularisation parameters, are manually chosen; therefore these last parameters are not the subject of the optimisation that we are describing. In any case, while all the above-mentioned strategies aim to evaluate the expensive objective function *accuracy* (which, in the case of a ANN or CNN is the prediction accuracy) as seldom as possible, very few algorithms offer a method to reduce the evaluation cost of *accuracy*. Furthermore the method needs to bring to convergence only few samples to built the dataset. The proposed approach therefore aims to combine these two aspects: the prediction of the final accuracy at low cost, and the automatic setting of hyperparameters.

5.2.2 The method applied to HO

Before proceeding with the proposed method it is necessary to make some premises. The method, in brief, aims to do meta-learning, i.e., learn by learn. This means that we want to use one learning technique to model another learning technique. The convenience lies in the fact that the supervisor of the method is a technique known in the literature and very robust, i.e., SVR, while the supervised method is more unstable with respect its hyperparameters, studied recently and not very robust, i.e., any ANN. For this reason, even if we have to take care of the best setting of SVR and curve-fitting hyperparameters, this task is enormously easier than the same one in the ANNs. Specifically, in the case of SVRs, particular attention will be paid to the choice of the kernel to use and the hyperparameters related to the kernel itself. In the case of curve-fitting we must identify the form of (5.10) and the hyperparameters related to the chosen function.
From now on we will focus on a CNN for image classification. What we are looking for is the optimal configuration of some of its hyperparameters. Obviously, the first step of the algorithm must be the decision of the hyperparameters to be analysed and the choice of all the others. We will focus on the choice of mini-batch size, steplength and optimiser type. The same method can however be extended to the choice of other hyperparameters such as number of layers, number of neurons per layer or more generally characteristics related to the ANN architecture.
We start by explaining the procedure in detail; first of all we have to create the dataset as described before. As can be seen in Algorithm 6, the entire

Algorithm 6 Hyperparameters space

1: Choose hyperparameters to optimise: (1) learning rate, (2) mini-batch size and (3) optimiser
2: Choose bounds for the hyperparameters and possible configurations:

- V_1 is the set of 16 values logaritmically spaced in $[1e-5, 1e-2]$
- $V_2 = \{2, 4, 8, 16, 32, 64, 128, 256, 512\}$
- $V_3 = \{SGD, MOMENTUM, AdaM\}$

dataset is composed of 432 samples. In the proposed case-study, we decided to sample the space of the hyperparameters estimating 10% of all possible configurations. These configurations are chosen with uniform distribution in the hyperparameter space. Next, an SVR is trained on this dataset, using for any example the final accuracy as label. We use cross validation techniques to set up kernels. Even if it takes time, this preliminary step can lead to a less expensive hyperparameter optimisation process than those already known in the literature.

Now, we use the Algoritm 7 [23] to perform the hyperparameters space exploration to estimate the optimal configuration, using the SVR or curve-fitting method to predict the final accuracy. In this approach, at the step 1 of Algorithm 7, each hyperparameter to be set is described by the letter i associated with the vector V_i which contains all possible configurations. Each element of the V_i vectors is associated with a probability. These probabilities are initially assigned with uniform distribution and stored in the P_i vectors. At each step of the exploration algorithm a value is chosen for each hyperparameter; this choice is made according to the probability distribution. At this point, the characteristics of the new CNN, which will be trained only in the very first epochs, have been fixed. Therefore, based on the vector containing the characteristics that define the CNN and the accuracies in the earliest epochs, the SVR or curve-fitting methodology predicts the accuracy of the CNN reached at the end of the learning process. This final accuracy, is considered as a reward, which we will indicate as $r^{(t)}$. At this point, based on the last two rewards, the probabilities are updated. In particular, if the reward $r^{(t)}$ relative to the iteration t is greater than the reward $r^{(t-1)}$ relative to the previous time, the probability of the values of the hyperparameters selected at time t is increased penalising the other configurations. On the opposite if $r^{(t)} < r^{(t-1)}$ we penalise the probabilities relating to the selected values for the hyperparameters at time t and we increase the others. Obviously, each

Algorithm 7 The hyperparameter space exploration algorithm

For each V_i sets the associated probabilities P_i, with $i \in \{1, \ldots, n_{param}\}$.
$V_i = \{a_1^i, \ldots, a_{v_i}^i\}$ for each i.
$P_i = \{p_1^i, \ldots, p_{v_i}^i\}$, with $p_j^i = \frac{1}{v_i}$ for each i and j.
Choose an initial state $x^{(0)} = \{a_{i_0}^1, \ldots, a_{i_0}^{n_{param}}\}$.
Training CNN($x^{(0)}$) for some epochs.
Use SVR and curve-fitting to predict final accuracy r^0.
$t = 1$.
while $t \leq maxiter$ and convergence=false **do**
    Generate a realisation of the random state $x^{(t)}$ according with probabilities.
    Training CNN($x^{(t)}$) for some epochs.
    Use SVR and curve-fitting to predict final accuracy r^t.
    if $r^t > r^{t-1}$ **then**
        $p_{i_t}^i = p_{i_t}^i + \epsilon$
        $p_j^i = p_j^i - \frac{\epsilon}{v_i - 1}$, for $j \in \{1, \ldots, v_i\} - \{i_t\}$
    else
        $p_{i_t}^i = p_{i_t}^i - \epsilon$
        $p_j^i = p_j^i + \frac{\epsilon}{v_i - 1}$, for $j \in \{1, \ldots, v_i\} - \{i_t\}$
    end if
    if $\exists\, \hat{j} : p_{\hat{j}}^i \geq threshold^i \; \forall\, i \in \{1, \ldots, n_{param}\}$ **then**
        convergence = true
    end if
    $t = t + 1$
end while

update must always keep a probability space in the P_i, i.e. all values must remain in the $[0, 1]$ range and the probability of each vector must add up to 1. In this way during the hyperparameter space exploration the probability vectors are dynamically modified, the exploration stops if a value for each hyperparameter reaches a probability higher than a certain threshold s_i. At this point the process stops, because it is believed that the exploration has reached convergence or a maximum number of iterations is achieved. In the final step, the 10 best configurations, according to the prediction technique, are trained to convergence and the hyperparameter configuration that leads to the highest accuracy is chosen.

5.3 Numerical experiments

Several numerical tests have been performed to validate the proposed method. The ANN we are considering is a CNN for multiple image classification. As dataset we have chosen MNIST, which we have already discussed in previous chapters and CIFAR10 (`https://www.cs.toronto.edu/~kriz/cifar.html`). The CIFAR10 dataset consists of 6000032 $\times$ 32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 testing images. The classes of CIFAR10 are: airplane, car, bird, cat, deer, dog, frog, horse, ship and truck.

5.3.1 CNN architectures

For the MNIST dataset, the CNN has five levels in addition to input and output. The hidden states of the network are two convolutions, each followed by a max-pool; a fully-connected state with dropout connected to the output completes the network. As activation function we use the ReLU for each layer.

In the case of CIFAR10 the network is more complex, being composed of twelve hidden layers. In particular, the input is connected to four blocks composed of a convolution and a max-pool that connect to four final fully-connected states before exiting with the output state. ReLU has been used as activation function, in addition to batch normalisation and dropout techniques in the fully-connected layers. To avoid the phenomenon of overfitting, we have used various techniques. As known in the literature, all Machine Learning methodologies, if the learning phase is too long, can lead the method to be too adherent to the data and unable to generalise. For this reason, in addition to using dropout, as mentioned before, in the learning phase we

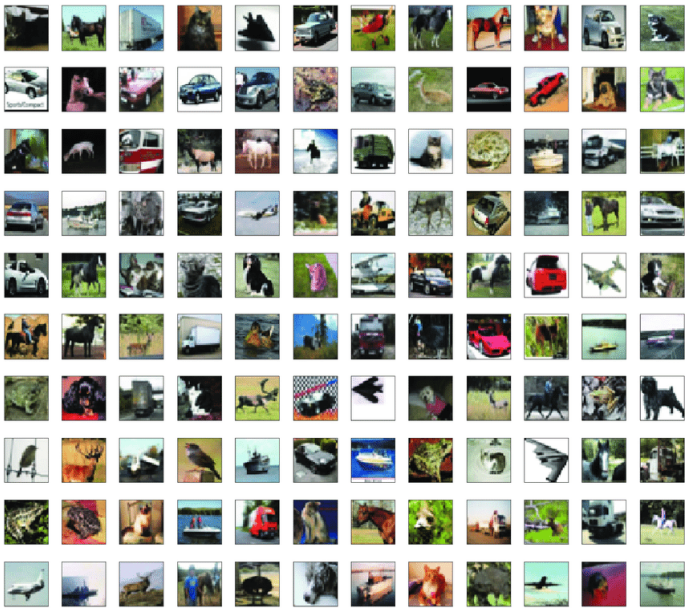
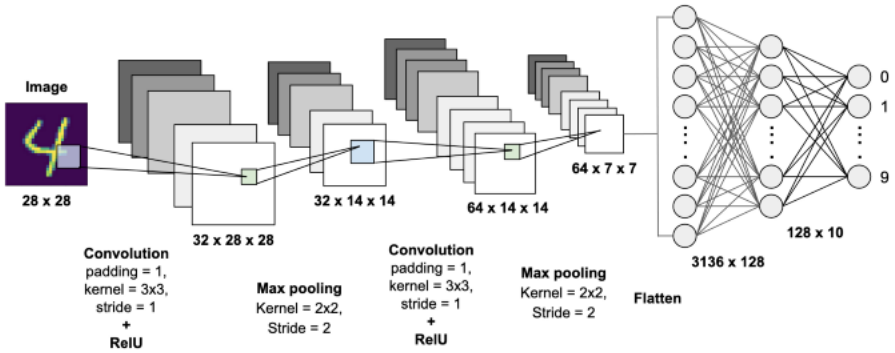


Figure 5.1: Random images from dataset CIFAR10.

Figure 5.2: CNN scheme for MNIST database.



have used the Early stopping technique, which we have already discussed in detail in previous chapters.

5.3.2 Results

As explained in the previous sections the first phase of the proposed technique is to create the dataset on which to train SVM and curve-fitting. Since this is the most expensive part from a computational point of view, it has been carried out on CINECA parallel architectures, in particular on Marconi cluster (`https://www.cineca.it/en/content/marconi`). Once the various architectures of the networks that will make up the database have been generated, they can be trained independently of each other. This high degree of parallelisation of the method makes it particularly suitable for node infrastructures such as CINECA. For both MNIST and CIFAR10 the dataset consists of a table in which each row corresponds to a complete training of a network. As hyperparameters in these experiments, steplength, optimiser type and mini-batch size have been chosen; these too have been stored together in the table representing the hyperparameters space. In the case of MNIST database, the number of networks to be trained was rather small: 44 examples, of which 35 were used for SVM training and 9 for the testing phase. Once the database was established, we tested several kernels for SVM, in particular: linear, polynomial and Gaussian. We chose Gaussian kernel because it performed better than the others in terms of error, evaluated with Mean Squared Error (MSE).

	Ground-truth	Linear	Polynomial	Gaussian
0	0.7129	0.816350	0.854324	0.713435
1	0.1871	0.197152	0.207737	0.175775
2	0.1000	0.200523	0.204122	0.200774
3	0.6820	0.704606	0.599408	0.781832
4	0.4340	0.381075	0.301048	0.456969
5	0.6369	0.572801	0.489510	0.638199
6	0.7315	0.747803	0.682245	0.805067
7	0.1000	0.190538	0.203211	0.175411
8	0.4783	0.410684	0.325788	0.491291
MSE	0	0.04136	0.1138	0.0320

Table 5.1: Predictions on random test values of the three different SVM kernels for MNIST dataset: linear, polynomial and Gaussian.

After acquiring the accuracy in the first three epochs, the method pre-

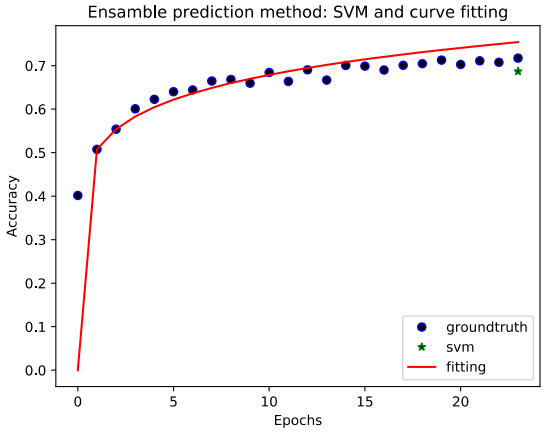


Figure 5.3: Prediction of curve-fitting and SVM with three epoches, compared with the real accuracies for MNIST dataset.

dicts what the final accuracy will be. The tests have been carried out with the most well known kernels: linear, polynomial and Gaussian. In the Table 5.1 we represent all the nine examples showing the predictions of the different kernels in the MNIST case. In this case, predictions are made on the random test values and are based on three epochs, for the MNIST dataset; we report also the MSE values, in the last row; the obtained results show that the kernel providing the best predictions is the Gaussian one. The MSE values are reported in Table 5.1, where is shown that the Gaussian one is the kernel that dominates the other two. As far as the SVR is concerned, we have tried different combinations of input parameters. In addition to the parameters that characterise the network, we have included accuracies after 2/3/4 epochs. We then decided to fix the ideas on the 3 epochs because it seemed to us a good compromise between training time and SVR performance. Same thing can be said for the curve-fitting: different functions were tested to find the most suitable one. Figure 5.3 shows a graphical example of the prediction provided by SVM and curve-fitting. In the graph, the full points represent the real values of accuracy, period by period. The star in the final epoch is the SVM forecast, while the line is the curve obtained with the fitting. For both SVM and curve-fitting, only the accuracies acquired during the first three epochs are used to predict the accuracy in the final epoch with the MNIST dataset. As explained before, curve fitting is a recovery procedure, used only when SVR predicts a value greater than one, in all the experiments we use this recovery procedure only in the 3% of the cases.

Figure 5.4 shows the results of the proposed technique applied to the prediction of the final CNN accuracy for the CIFAR-10 dataset. In this case, we

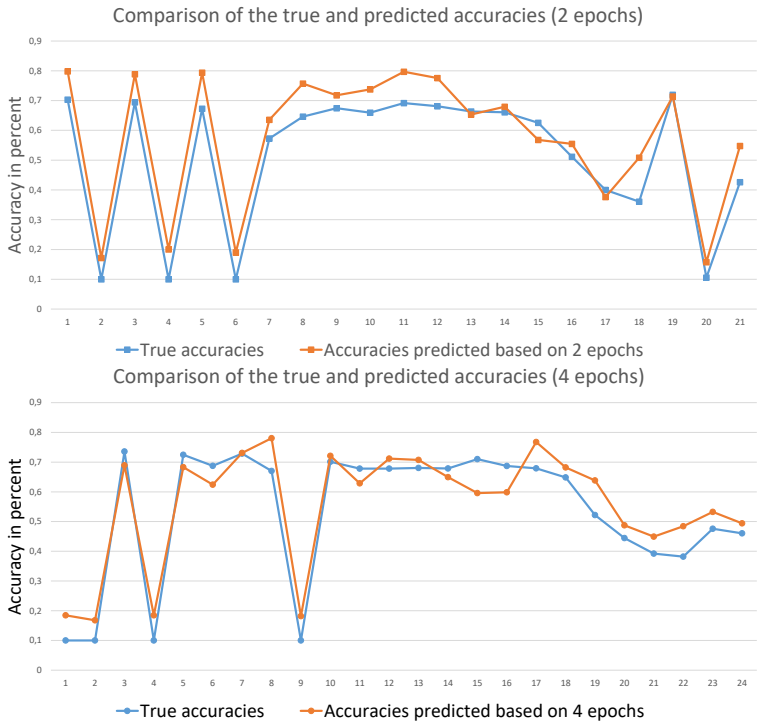


Figure 5.4: Comparison. with CIFAR dataset, of the predicted final accuracy with SVM and curve-fitting and the ground truth, setting the method with only two training epochs (top panel) and four (bottom panel).

use only the accuracies of the first two or four training periods to predict the final one. The orange line represents the prediction made by our method, the blue line the true accuracy (i.e., the net is fully trained to convergence with the same hyperparameters). As we can see, the method can effectively provide a satisfactory prediction of the CNN's final behaviour after only a few epochs. At this point we proceed with the last phase, i.e., we use the predictor, just set, to guide us in the hyperparameter space exploration. Since the results are satisfactory, at this point, we have used the SVR as predictor for the final accuracy of the ANN. This predictor has been included in a research method as described above. We performed this additional test using the first two epochs to predict the final one. After 200 iterations, of the method, we were able to find the parameters leading to the best final predicted accuracy. To confirm this result we train until convergence the CNN with the best final predicted accuracy.
The hyperparameters provided by our method are: the learning rate is 0.0008425, the mini-batch size is 128 and the optimiser chosen is AdaM.

Conclusions and future works

In this work, several techniques for setting the hyperparameters of a stochastic optimization method for deep learning problems have been analyzed and discussed in order to make the tuning phase of the method less expensive and more adaptive/automatic. In particular, in Section 3, we propose a strategy for a dynamic increase of the size of the mini-batch used to estimate the stochastic gradient; this technique leads to a considerable improvement in the accuracy, without increasing the computational cost. Indeed, the test controlling an increase of the mini-batch size is included in that performed for Early Stopping, so that the computational costs for each epoch are unchanged. We observe that a larger starting mini-batch size does not allow to obtain an improvement in the accuracy as the strategy to set an adaptive size during the network learning process. For future experiments, the idea is to apply the same methodology to larger datasets and more complex ANN, where Early Stopping has already been effectively tested. In such contexts, the proposed approach could be easily inserted.
In Section 4, we propose to tailor the steplength selection rule based on the Ritz-like values, used successfully in the deterministic gradient schemes, to a stochastic scheme, recently suggested by Bollapragada et al. [8]. This SG method includes an adaptive sub-sampling strategy, aimed to control the variance of the stochastic directions. We observed that the theoretical properties of this approach hold under the assumption that the steplength selection rule obeys to the assumption $\alpha_k \in (\alpha_{min}, \alpha_{max}]$, where α_{max} is proportional to the inverse of the Lipschitz parameter of the objective function gradient. Consequently, we reformulate the procedure for obtaining the Ritz-like values in the stochastic framework, by using the stochastic gradients instead of the standard gradients. It is required that these stochastic directions, although based on different sub-samples, satisfy two conditions (the inner product test and the orthogonality test), ensuring the descent property in expectation.
We propose two different ways to select the current steplength, by simply toggling the Ritz-like values with the harmonic Ritz-like values (**A-R** method) or using the harmonic Ritz-like values only when the size of the sub-sample is in-

creased (**AA-R** method). The numerical experimentation highlight that the proposed methods enable to obtain an accuracy similar to the one obtained with SG mini-batch with fixed best-tuned steplength. Although also in this case it is necessary to carefully select a thresholding range for the steplengths, the proposed approach appears sligthly dependent on the bounds imposed on the steplengths, making the parameters setting less expensive with respect to the SG framework. In conclusion, the proposed technique provides a guidance on the learning rate selection and it allows to perform similarly to the SG approach equipped with the best-tuned steplength. For the future, the idea is to include in our scheme a projection or a proximal step, in order to manage the presence of constraints or non-smooth term of the objective function. In the last section, we propose and implement a novel approach to predict the final behaviour of a learning process. This method exploits Support Vector Regression (or a recovery curve-fitting procedure to foresee the resulting accuracy of a time-consuming iterative algorithm, using only the information on its behaviour acquired after few iterations. We apply this technique to a CNN, in order to quickly understand if the training of the network will end up with a satisfactory accuracy. The results show that the predictions achieved with our technique are quite similar to the ground-truth, and confirm that this strategy can be of particular interest in the hyperparameter optimisation domain. Future developments will concern the design of a procedure for an (almost) automatic tuning of hyperparameters, such as the number of layers or the activation function of a network, exploiting our SVR-curve-fitting predictor.

Bibliography

[1] Peter Mattson et al. "MLPerf: An industry standard benchmark suite for machine learning performance". English (US). In: *IEEE Micro* 40.2 (Mar. 2020), pp. 8–16. ISSN: 0272-1732. DOI: 10.1109/MM.2020.2974843.

[2] Zoph B. and Le Q. V. "Neural architecture search with reinforcement learning". In: *Int. Conf. Learning Representations, Toulon, France, pp. 1-16* (2017).

[3] Naik N. Baker B. Gupta O. and Raskar R. "Designing neural network architectures using reinforcement learning". In: *Int. Conf. Learning Representations, Toulon, France, pp. 1-18* (2017).

[4] S. Bellavia et al. "Adaptive Regularization Algorithms with Inexact Evaluations for Nonconvex Optimization". In: *SIAM Journal on Optimization* 29.4 (2019), pp. 2881–2915.

[5] James Bergstra et al. "Algorithms for Hyper-Parameter Optimization". In: *Advances in Neural Information Processing Systems*. Ed. by J. Shawe-Taylor et al. Vol. 24. Curran Associates, Inc., 2011, pp. 2546–2554. URL: https://proceedings.neurips.cc/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf.

[6] D.P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2015.

[7] Liao H.-Y. M. Bochkovskiy A. Wang C.-Y. "YOLOv4: Optimal Speed and Accuracy of Object Detection". In: *arXiv:2004.10934* (2020).

[8] R. Bollapragada, R. Byrd, and J. Nocedal. "Adaptive sampling strategies for stochastic optimization". In: *SIAM Journal on Optimization* 28.4 (2018), pp. 3312–3343.

[9] L. Bottou, F. E. Curtis, and J. Nocedal. "Optimization Methods for Large-Scale Machine Learning". In: *SIAM Review* 60.2 (2018), pp. 223–311.

[10] Christopher J. C. Burges. "A Tutorial on Support Vector Machines for Pattern Recognition". In: 2.2 (June 1998), pp. 121–167. ISSN: 1384-5810. DOI: 10.1023/A:1009715923555. URL: https://doi.org/10.1023/A:1009715923555.

[11] R. H Byrd et al. "Sample size selection in optimization methods for machine learning". In: *Mathematical Programming* 1.134 (2012), pp. 127–155.

[12] Han Cai et al. "Reinforcement Learning for Architecture Search by Network Transformation". In: *CoRR* abs/1707.04873 (2017). arXiv: 1707.04873. URL: http://arxiv.org/abs/1707.04873.

[13] C. Cartis and K. Scheinberg. "Global convergence rate analysis of unconstrained optimization methods based on probabilistic models". In: *Mathematical Programming* 1 (2015), pp. 1–39.

[14] F. E. Curtis and W. Guo. "Handling nonpositive curvature in a limited memory steepest descent method". In: *IMA Journal of Numerical Analysis* 36.2 (Apr. 2016), pp. 717–742. ISSN: 1464-3642. DOI: 10.1093/imanum/drv034.

[15] A. DÃ©fossez et al. *A Simple Convergence Proof of Adam and Adagrad*. 2020. arXiv: 2003.02395 [stat.ML].

[16] Y. H. Dai and Y. Yuan. "Alternate minimization gradient method". In: *IMA J. Numer. Anal.* 23 (3 2003), pp. 377–393.

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

[18] Thomas Elsken, Jan-Hendrik Metzen, and Frank Hutter. *Simple And Efficient Architecture Search for Convolutional Neural Networks*. 2017. arXiv: 1711.04528 [stat.ML].

[19] Matthias Feurer and Frank Hutter. "Hyperparameter Optimization". In: ed. by Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. Springer, 2019. Chap. 1, pp. 3–38.

[20] R. Fletcher. "A limited memory steepest descent method". In: *Mathematical Programming, Ser. A* 135 (2012), pp. 413–436.

[21] G. Franchini, V. Ruggiero, and L. Zanni. "On the steplength selection in Stochastic Gradient Methods". In: *Numerical Computations: Theory and Algorithms (NUMTA, 2019)*. Ed. by Kvasov D. E. Sergeyev Y. D. Vol. 11973. Lecture Notes in Computer Science. 2020.

[22] Giorgia Franchini, Paolo Burgio, and Luca Zanni. "Artificial Neural Networks: The Missing Link Between Curiosity and Accuracy". In: *Intelligent Systems Design and Applications*. Ed. by Ajith Abraham et al. Cham: Springer International Publishing, 2020, pp. 1025–1034.

[23] Giorgia Franchini, Mathilde Galinier, and Micaela Verucchi. "Mise en abyme with Artificial Intelligence: How to Predict the Accuracy of NN, Applied to Hyper-parameter Tuning". In: *Recent Advances in Big Data and Deep Learning*. Ed. by Luca Oneto et al. Cham: Springer International Publishing, 2020, pp. 286–295.

[24] G. Frassoldati, G. Zanghirati, and L. Zanni. "New Adaptive Stepsize Selections in Gradient Methods". In: *J. Indutrial and Management Optimization* 4.2 (2008), pp. 299–312.

[25] M. P. Friedlander and M Schmidt. "Hybrid Deterministic-Stochastic Methods for Data Fitting". In: *SIAM Journal on Scientific Computing* 34.3 (2012), A1380–A1405.

[26] Schmidt M. Friedlander M. P. "Hybrid Deterministic-Stochastic Methods for Data Fitting". In: *arXiv:1104.2373* (2011).

[27] Jacqueline Gottlieb et al. "Information-seeking, curiosity, and attention: computational and neural mechanisms". In: *Trends in Cognitive Sciences* 17.11 (2013), pp. 585–593. ISSN: 1364-6613. DOI: https://doi.org/10.1016/j.tics.2013.09.001. URL: http://www.sciencedirect.com/science/article/pii/S1364661313002052.

[28] F. Hashemi, S. Ghosh, and R. Pasupathy. "In adaptive sampling rules for stochastic recursions". In: *Simulation Conference (WSC), 2014 Winter* (2014), pp. 3959–3970.

[29] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, eds. *Automatic Machine Learning: Methods, Systems, Challenges*. Springer, 2019.

[30] Hoos H. Hutter F. and Leyton-Brown K. "Sequential model-based optimization for general algorithm configuration". In: LION-5. *Extended version as UBC Tech report TR-2010-10* (2011).

[31] Bergstra J. and Bengio Y. "Random search for hyper-parameter optimization". In: *J. Mach. Learn. Res.**13**(1) (2012) 281-305* (2012).

[32] Chen L.-C. Collins M. Zhu Y. Papandreou G. Zoph B. Schroff F. Adam H. Shlens J. "Searching for efficient multi-scale architectures for dense image prediction". In: *Advances in Neural Information Processing Systems* (2018), pp. 8713–8724.

[33] Zhong Z. Yan J. Wei W. Shao J. and Liu C.-L. "Practical block-wise neural network architecture generation". In: *Conf. Computer Vision and Pattern Recognition* (2018).

[34] Rie Johnson and Tong Zhang. "Accelerating Stochastic Gradient Descent using Predictive Variance Reduction". In: *Advances in Neural Information Processing Systems 26*. Ed. by C. J. C. Burges et al. Curran Associates, Inc., 2013, pp. 315–323.

[35] H. Karimi, J. Nutini, and Schmidt M. "Linear Convergence of Gradient and Proximal-Gradient Methods Under the Polyak-Ãojasiewicz Condition". In: *Machine Learning and Knowledge Discovery in Databases. ECML PKDD 2016*. Ed. by P. Frasconi et al. Vol. 9851. Lecture Notes in Computer Science. 2016.

[36] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[37] Y. LeCun, Y. Bengio, and Hinton G. "Deep learning". In: *Nature volume 521, pages 436-444* (2015).

[38] J. Lehman and K. O. Stanley. "Abandoning Objectives: Evolution Through the Search for Novelty Alone". In: *Evolutionary Computation* 19.2 (2011), pp. 189–223. DOI: 10.1162/EVCO_a_00025.

[39] Y. Li, Colin Wei, and Tengyu Ma. "Towards Explaining the Regularization Effect of Initial Large Learning Rate in Training Neural Networks". In: *ArXiv* abs/1907.04595 (2019).

[40] N. Loizou and P. Richtarik. "Momentum and Stochastic Momentum for Stochastic Gradient, Newton, Proximal Point and Subspace Descent Methods". In: *arXiv:1712:09677v2* (2018).

[41] Zhou Lu et al. "The Expressive Power of Neural Networks: A View from the Width". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017, pp. 6231–6239. URL: https://proceedings.neurips.cc/paper/2017/file/32cbf687880eb1674a07bf717761dd3a-Paper.pdf.

[42] Dominic Masters and Carlo Luschi. *Revisiting Small Batch Training for Deep Neural Networks*. 2018. arXiv: 1804.07612 [cs.LG].

[43] Tiesis V. Mockus J. and Zilinskas A. "The application of Bayesian methods for seeking the extremum". In: *In L.C.W. Dixon and G.P. Szego, editors,* Towards Global Optimization*, volume 2, pages 117-129. North Holland, New York* (1978).

[44] Shahriari B. Swersky K. Wang Z. Adams R. P. De Freitas N. "Taking the human out of the loop: A review of bayesian optimization". In: *Proc. IEEE **104**(1) 148-175* (2012).

[45] Alfarra M. Hanzely S. Albasyoni A. Ghanem B Richtarik P. "Adaptive Learning of the Optimal Mini-Batch Size of SGD". In: *arXiv:2005:01097v* (2020).

[46] Nakkiran P. "Learning Rate Annealing Can Provably Help Generalization, Even for Convex Problems". In: *arXiv:2005.07360* (2020).

[47] Lutz Prechelt. *Early Stopping But When?* Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Springer Berlin Heidelberg, 2012, pp. 53–67.

[48] Lach Arlinghaus S. "PHB Practical Handbook of Curve Fitting". In: *CRC Press* (1994).

[49] D. di Serafino et al. "On the steplength selection in gradient methods for unconstrained optimization". In: *Applied Mathematics and Computation* 318 (2018), pp. 176–195.

[50] G. Serra R. Zanarini. *Complex Systems and Cognitive Processes*. Springer-Verlag Berlin Heidelberg, 1990.

[51] C. Tan et al. "BB Step Size for SGD". In: *Advances in Neural Information Processing Systems 29 (NIPS 2016)*. Ed. by D.D. Lee et al. 2016.

[52] Sebastian B. Thrun. *Efficient Exploration In Reinforcement Learning*. Tech. rep. 1992.

[53] Joel A. Tropp. "An Introduction to Matrix Concentration Inequalities". In: *Foundations and Trends in Machine Learning* 8.1-2 (2015), pp. 1–230. ISSN: 1935-8237. DOI: 10.1561/2200000048. URL: http://dx.doi.org/10.1561/2200000048.

[54] Franchini G. Ruggiero V. and Zanni L. "Ritz-like values in steplength selections for stochastic gradient methods." In: *Soft Comput 24* (2020), pp. 17573–17588. DOI: https://doi.org/10.1007/s00500-020-05219-6.

[55] Smith S. L. Kindermans P.-J. Ying C. Le Q. V. "Don't Decay the Learning Rate, Increase the Batch Size". In: *arXiv:1711.00489* (2017).

[56] Y. Wardi. "Stochastic algorithms with armijo stepsizes for minimization of functions." In: *J Optim Theory Appl 64* (1990).

[57] Zhuang Yang et al. "Mini-batch algorithms with BarzilaiÃ¢Borwein update step". In: *Neurocomputing* 314 (2018), pp. 177–185.

[58] B. Zhou, L. Gao, and Y. H. Dai. "Gradient Methods with Adaptive Step-Sizes". In: *Comput. Optim. Appl.* 35.1 (2006), pp. 69–86.

[59] Le Q. V. Zoph B. "Neaural Architecture Serach with Reinforcemente Learning". In: *arXiv:1611.01578v2* (2017).