Università degli Studi di Modena e Reggio Emilia

Dottorato di Ricerca in Matematica

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

Ciclo XXXII, Coordinatore Prof. Cristian Giardinà

Interior Point Methods Meet Neural Networks: an Application to Image Deblurring

Settore Scientifico Disciplinare MAT/08

Dottoranda	**Tutore**
Dr. Carla Bertocchi	Prof. Marco Prato

Anni 2016/2019

Contents

Summary

The purpose of this thesis is to propose a novel Deep Learning model to address the problem of image reconstruction. For this, we summarize the general theory on machine learning and more particularly the mathematics underlying Deep Learning. We introduce neural networks starting from their basic units: the formal neurons. These are combined to form complex and different structures, such as convolutional neural networks. We also study various aspects related to the training of neural networks and the various learning algorithms used to train them.

The problem of image reconstruction, or deblurring problem, is a well known inverse problem. Before approaching it, we review the generalities of the problem, providing some details on the image formation process and on the noise deriving from data acquisition. Following a maximum a posteriori statistical approach, the problem is reformulated as a regularized optimization problem, in which the objective function to be minimized is a sum of a data discrepancy measure with a regularization term. Also additional contraints can be imposed to incorporate a priori knowledge on the desired solution. In our work we consider smooth data-fidelity and regularization terms and we include constraints in the objective function by means of a logarithmic barrier.
A proximal interior point method (IPM) is considered to address the minimization problem, in which the proximity operator is restricted only to the barrier function. Then, considering a finite number of iterations, the iterative algorithm is unfolded into a neural network, that we call iRestNet. In particular, the iterations of the IPM are incorporated into the layers of the network, whose training process merges with the optimization process. The key issue of our proposed approach is that the regularization parameter, the barrier parameter and the steplength needed in the iterations of the IPM are automatically chosen, by means of a particular Deep Learning strategy. In addition, we provide a stability result for the proposed network when the data fidelity term and the regularization function are quadratic.

We use benchmarks image datasets to perform the training and test the resulting neural network. For every experiment, a dataset is created with a different type of blur and corrupted with Gaussian noise. Comparisons with standard gradient projection methods, with recent machine learning algorithms and also with other unfolded methods have been performed and the tests show good performances and the competitiveness of our approach.

We have also extended the proposed approach to deblurring problems on data corrupted with Poisson noise. Although the model needs to be improved, preliminary results show that also in this case iRestNet achieves a good reconstruction quality.

Sintesi

In questa tesi si propone una nuova strategia di Deep Learning per affrontare il problema di ricostruzione di immagini digitali. Per far ciò, riassumiamo la teoria generale sul machine learning e più in particolare la matematica che sta alla base del Deep Learning. Introduciamo le reti neurali partendo dalle loro unità di base: i neuroni formali. Questi vengono combinati per formare strutture complesse e diverse, come le reti neurali convolutive. Inoltre studiamo vari aspetti relativi al training delle reti neurali e i vari algoritmi di learning utilizzati per addestrarle.

Il problema di ricostruzione di immagini, o deblurring, è un noto problema inverso. Prima di approcciarci ad esso, rivediamo le generalità del problema, fornendo alcuni dettagli sul processo di formazione delle immagini e sul rumore derivante dall'acquisizione dei dati. Seguendo un approccio statistico di tipo maximum a posteriori, il problema di deblurring viene solitamente riformulato come problema di ottimizzazione regolarizzato, in cui la funzione obiettivo da minimizzare è data dalla somma di un termine di data-fidelity e di un termine di regolarizzazione. È inoltre possibile imporre dei vincoli sulla funzione obiettivo, che permettono di integrare conoscenze a priori sulla soluzione desiderata. Nel nostro studio abbiamo considerato termini di data-fidelity e di regolarizzazione regolari e abbiamo incluso vincoli di tipo box sulla funzione obiettivo, per mezzo di una funzione di barriera logaritmica.
Per affrontare il processo di minimizzazione consideriamo un metodo interior point (IPM), in cui il proximity operator è applicato alla sola funzione di barriera. Partendo da un numero finito di iterazioni del metodo iterativo, queste vengono incorporate in una rete neurale, che chiamiamo iRestNet. La peculiarità dell'approccio proposto sta nel fatto che il parametro di regolarizzazione, il parametro di barriera e la lunghezza di passo, necessari nell'applicazione del metodo IPM, sono scelti automaticamente, mediante una particolare strategia di Deep Learning. Inoltre, è fornito un risultato teorico sulla stabilità della rete in particolari condizioni.

Per fare il training e testare la rete neurale risultante si sono utilizzati diversi dataset di immagini. Per ogni esperimento, viene creato un set di immagini derivanti da un diverso tipo di blur e corrotte con rumore gaussiano. In particolare ci si è confrontati con metodi standard di proiezione del gradiente, con recenti algoritmi di machine learning e con altri "unfolded methods". In conclusione, i test mostrano come iRestNet abbia buone prestazioni e sia competitiva rispetto allo stato dell'arte.

Abbiamo inoltre esteso l'approccio proposto a problemi di deblurring su dati corrotti con rumore poissoniano. Nonostante il modello debba essere perfezionato, risultati preliminari dimostrano la bontà di iRestNet anche in questo caso.

Introduction

Machine learning research seeks to provide learning models with the ability to learn and improve automatically through observations and interaction with the world. The main objective of a machine learning system is to generalize from experience, to be able to perform accurately on new, unseen examples after having experienced a learning dataset. The training examples come from some generally unknown probability distribution, representative of the space of occurrences, and the machine learning algorithm builds a mathematical model about this space that enables it to produce sufficiently accurate predictions on new cases.

In this last decade, machine learning is having a big impact in many areas of science, industry, and engineering, and recently Deep Learning attracted even more attention. Deep Learning is a particular branch of machine learning which involves the use of artificial neural network models. These are engineered systems inspired by the biological brain, consisting of the interconnection of basic units called neurons, arranged in layers. By increasing the number of layers, a deep network can represent functions of increasing complexity. In particular, most tasks that consist of mapping an input vector to an output vector, can be performed by Deep Learning, given sufficiently large models and sufficiently large datasets of labelled training examples. In recent years, Deep Learning popularity greatly increased, thanks to the availability of more powerful computers and larger datasets that allowed experimentation and the development of new techniques to train deeper networks. The training phase of a network require minimizing the empirical risk on the training set images, which is not a trivial problem. Most of the successful approaches are gradient-based learning methods, which aim to generate a sequence of iterates $\{\theta_k\}_{k\in\mathbb{N}}$, in the parameter space, so that the empirical risk is decreased along the iterations. In particular, the decrease of the objective function is obtained by moving along a descent direction, which is found exploiting first–order information.

The research activity presented in this thesis has dealt with the analysis of Deep Learning methods applied to inverse problems. We paid particular attention to image reconstruction problems, both by examining state-of-the-art deblurring algorithms, and by proposing our own deblurring method.
We consider inverse problems arising from the following mathematical model:

$$y = \mathcal{D}(H\bar{x}), \tag{1}$$

where $y \in \mathbb{R}^m$ is the observed data, $\bar{x} \in \mathbb{R}^n$ is the sought image, $H \in \mathbb{R}^{m \times n}$ is the observation operator, which is assumed linear and known, and $\mathcal{D}$ is the noise perturbation operator. In this context, both variational and Deep Learning approaches provide efficient methods for finding a good estimate of $\bar{x}$, while offering different benefits and drawbacks, which will be discussed below.

In order to find an adequate solution to an ill-posed inverse problem arising from (1), variational methods incorporate prior information on the sought variable $\bar{x}$, through constraints and regularization functions, such as the total variation and its various extensions [4]. Thus, the sought signal can be approximated by the minimizer of a regularized objective function, expressed as the sum of a data–fidelity term $f : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$, , which measures the fidelity of the solution to the observation model (1), and an additional regularization term $\mathcal{R} : \mathbb{R}^n \to \mathbb{R}$, which encodes some prior information on the sought image and improves stability to noise. This leads to

$$\min_{x \in \mathcal{C}} \; f(Hx, y) + \lambda \mathcal{R}(x), \tag{2}$$

where $\lambda \in \mathbb{R}_{>0}$ is a regularization parameter, which balances the role of the two terms in the objective function, and $\mathcal{C}$ is a subset of $\mathbb{R}^n$.
Over the past decades, iterative reconstruction methods have achieved remarkable results to solving inverse problems in imaging, like (2). Thanks to robust regularizers such as total variation, practical algorithms have appeared with good image quality and reasonable computational complexity. In particular, it has recently been shown that combining the interior point framework with a proximal forward–backward strategy [34, 36], leads to very competitive solvers [32, 38, 39]. Based on these considerations, we consider a particular proximal interior point algorithm for the development of our own approach. Each iteration of the algorithm is made up of two steps, namely a forward step, which is a gradient step on the differentiable term, and proximal backward step, which involves the evaluation of the proximity operator of a logarithmic barrier.
Although useful, this approach is sometimes limited by its complexity: solving (2) may be too slow for real-time applications, especially if we consider the time required for parameters estimation. For example, $\mathcal{R}$ is usually parametrized by one or several parameters, like λ, whose optimal choice may strongly depend on the data at hand. The parameters are often tuned manually or by following some heuristic rules. However, these methods are often time-consuming and their success is not always guaranteed, with consequent loss in efficiency and versatility of the resulting reconstruction scheme. An accurate setting of the regularization parameter is particularly critical, because the quality of the reconstruction obtained with regularized approaches highly depends on it. Existing approaches for the selection of λ, which are based on statistical considerations, generally lead to a substantial increase in computational cost.

Deep neural networks (DNNs), and in particular convolutional neural networks (CNNs)

have shown outstanding performance on object classification and segmentation tasks. Motivated by these successes, researchers have begun to apply CNNs to various applications related to inverse problems, such as denoising [149], non-blind and blind deblurring [124, 125, 144], super-resolution [92], or CT reconstruction [25, 76], and they have started to report improvements over state-of-the-art methods.
Applying neural networks directly to the blurred image seems not to be the best way to deal with image reconstruction problems. Conversely, DNNs for inverse problems are often preceded by a pre-processing step, where a rough estimation of $\bar{x}$ can be found by using the inverse or pseudoinverse of H. The latter tends to strongly amplify noise. Hence, in this context, DNNs are used as denoisers and artifact-removers. A drawback of these metods is that, since DNNs are used as black-box, their explainability and reliability could be questioned. Furthermore, the pre-processing step, in itself, can include a penalty, thus amounting to solving a problem of the form (2), where the regularization weight strongly depends on the noise level, e.g., [22, 124].

Our research mainly concerned the study of recent strategies that merge Deep Learning techniques with variational methods, in order to take the strengths of both. One straightforward way to combine the benefits of both variational-based methods and DNNs is to unfold an iterative method into a network structure, by turning each iteration into a layer of the network. In our work, we propose a novel neural network architecture called iRestNet [13, 37], which is obtained by unfolding the aforementioned proximal interior point algorithm over a finite number of iterations. One key feature of this algorithm is that it produces only feasible iterates thanks to a logarithmic barrier. This barrier enables prior knowledge to be directly incorporated into iRestNet. The stepsize, barrier parameter, and regularization weight are untied across the network and learned for each layer, in a supervised fashion. To train iRestNet we use gradient backpropagation. The latter requires the derivatives of the involved proximity operator with respect to its input and to the parameters which are to be learned. Therefore, we conducted an analysis of the proximity operator of the barrier and of its derivatives, for three examples of interest. Once the network has been trained, its application on test images requires only a short execution time per image without any parameter search, as opposed to traditional variational methods.

Related works apply deep unfolding to probabilistic models, such as Markov random fields [68], topic models [31], and to different algorithms like primal-dual solvers [139] or the proximal gradient method [45, 97]. Classic optimization algorithms can be unfolded to perform many different tasks in image processing. For instance, FISTA and ISTA can be unfolded to perform sparse coding [59, 78], while the same ISTA and ADMM can be unfolded for image compressive sensing [127, 147]. Deep unfolding is also used to learn shrinkage functions, which can be viewed as proximity operators of sparsity-promoting functions [123, 128], or to optimize hyperparameters in nonlinear reaction diffusion models [30]. In the aforementioned works, some functions and operators are learned. To the best of our knowledge, iRestNet is

the first architecture corresponding to a deep unfolded version of an interior point algorithm with untied stepsize and regularization parameter. As opposed to other unfolding methods like [45, 97], the proximity operator and the regularization term are kept explicit, which establishes a direct relation between the original algorithm and the network.
Only a few works so far have considered combining interior point methods (IPMs) with Deep Learning. Every layer of the network from [2] solves a small quadratic problem using an IPM, while in [134], hard constraints are enforced on weights by using the logarithmic barrier function during training. It is worth noticing that, the goal of these approaches is to minimize the given objective function. Conversely, in our case a better indicator of perceptual quality (SSIM) is optimized during the training of iRestNet. For this reason, the output of the trained network is not necessarily a solution to problem (2). In addition, iRestNet appears to have more flexibility since the regularization weight can vary among layers.
One critical issue concerning neural networks is to guarantee that their performance remains acceptable when the input is perturbed. In this direction, a recent work [35] provides a theoretical framework which enables to evaluate the robustness of a network. We then conduct a stability analysis of the proposed network when the data fidelity term and the regularization function are quadratic, and we give explicit conditions under which the robustness of the proposed architecture is ensured.

Also numerical experiments are conducted to validate theoretic results. We test iRestNet on a set of non-blind image deblurring problems, where the images to be reconstructed are natural color images blurred with different types of blur, and corrupted with Gaussian noise. In this case, the problem is formulated as the constrained minimization of an objective function expressed as the sum of a mean–squared loss function and a smoothed Total Variation regularizer. The choice of the data–fidelity function is strictly related to the noise that affects the images, while the regularization function is chosen for its edge–preserving properties. From this problem formulation we derive the IPM iteration, and we use it to construct the layers of iRestNet. The network is then trained in a supervised fashion, in order to find an optimal reconstruction quality, and it is tested on a test set of images. The performance of our approach are compared with state-of-the-art methods.
Further work is devoted to extend iRestNet to data corrupted with Poisson noise, with the aim of extending the approach to a wider class of problems. This may be useful, because in imaging applications such as emission tomography, microscopy and astronomy, the main source of noise corrupting the images is photon counting [145], which is well described by a Poisson process. For Poisson noise case, we presents the preliminary results we obtained in numerical experiments, knowing that further work must be done in order to improve these results.

This thesis is organized as follows.

In Chapters 1 and 2, we introduce the general theory about machine learning and Deep Learning, respectively. In fact, it is helpful to fix some knowledge about learning theory, in order to use neural networks in an useful way, so as to exploit their potential in application applied to inverse problems. Specifically, in Chapter 1 machine learning framework is presented from a mathematical point of view; while paying particular attention to concepts of supervised learning, empirical risk minimization and generalization. The purpose of Chapter 2 is to introduce the mathematical theories underlying Deep Learning, the algorithms used to train neural network structures, as well as providing some historical perspectives about neural networks development. We explain neural networks by starting from their basic units: the formal neurons. We see how to combine them to form complex and different structures, like convolutional neural networks. Then we see various aspects of the optimization process involved in the training of these deep models, in order to tune their internal weights.

In Chapter 3, we give the generalities of image reconstruction problems, providing some details on the image formation process and on the noise arising during the data acquisition. Moreover, we resume the basics of the statistical framework underlying the solution of inverse problems, focusing particularly on the maximum a posteriori approach and on the regularization functionals which will be employed in numerical experiments. We describe the proximal interior point method which is at the core of our approach, and we conclude the chapter by conducting an analysis of the proximity operator of the barrier and of its derivatives, for three examples of interest.

In Chapter 4, after a brief introduction about unfolding methods, we present the proposed neural network architecture and its associated backpropagation method. In Section 4.3, we conduct a stability analysis of the proposed network when the data fidelity term and the regularization function are quadratic. The rest of the chapter is dedicated to numerical experiments and comparison to state-of-the-art methods for image deblurring, both in case of Gaussian noise and Poisson noise. Starting from the experiments conducted on Gaussian noise, we introduce the variational formulation of the problem approached, the specific structure we defined for iRestNet and how we conducted the training of the network. All the details about the experimental set-up are given, from the training, to the test of iRestNet, and its comparison with other methods. Also, numerical and visual results are reported. In the last section of the chapter we adapt iRestNet to address the image reconstruction problem on Poisson data. Also in this case, we provide numerical experiments and the preliminary results we obtained. Finally, we draw conclusions by discussing these results.

In Appendix A, we report a summary of basic concepts of functional analysis and convex analysis, which can be useful for the reader.

The research presented in this PhD thesis appears in the following publications:

1) C. Bertocchi, E. Chouzenoux, M.-C. Corbineau, J.-C. Pesquet, and M. Prato. Deep unfolding of a proximal interior point method for image restoration. *Inverse Problems*, 36(3):034005, Feb. 2020.

2) M.-C. Corbineau, C. Bertocchi, E. Chouzenoux, M. Prato, and J.-C. Pesquet. Learned Image Deblurring by Unfolding a Proximal Interior Point Algorithm. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 4664-4668, IEEE, Sep. 2019.

All the codes and a demo for training and testing iRestNet are avaliable at the link:
`https://github.com/mccorbineau/iRestNet`

Notations

- $\mathbb{R}^n$ denotes the n-dimensional Euclidean space endowed with the standard scalar product $(.,.)$ and the norm $\|.\|$.
- $\|\cdot\|$ denotes the Euclidean norm (also called ℓ^2-norm): $\|x\| = \|x\|_2 = \sqrt{x^T x}$.
- Given a normed vector space X, $\|\cdot\|_X$ denotes the norm in X induced by the inner product $(.,.)_X$: $\|x\|_X = \sqrt{(x,x)_X}$.
- $M \in \mathbb{R}^{m\times n}$ denotes a real matrix of m rows and n columns.
- $I_n \in \mathbb{R}^{n\times n}$ denotes the $n\times n$ identity matrix, i.e., the square matrix with ones on the main diagonal and zeros elsewhere. We will simply denote it by I, if the size can be trivially determined by the context.
- Id_n denotes the identity operator of $\mathbb{R}^n$.
- $e \in \mathbb{R}^m$ and $0 \in \mathbb{R}^m$ denote m-vectors with all entries equal to 1 and 0, respectively.
- $|x|_+ = \max(0, x)$, that is x if $x > 0$, 0 otherwise.
- Given $A, B \in \mathbb{R}^{m\times n}$, $A \odot B$ denotes the Hadamard product of A and B.
- $\prod$ denotes the product operation, e.g., $\prod_{i=1}^n i = 1 \cdot 2 \cdot \ldots \cdot n = n!$.
- If $x, y \in \mathbb{R}^n$, then $x^T y = \sum_{i=1}^n x_i y_i$ denotes the scalar product.
- If $x \in \mathbb{R}^n$, $x \geq 0 \Leftrightarrow x_i \geq 0,\ i = 1, \ldots, n.$ An analogous notation holds for $>$, $\leq$, $<$.
- $\bar{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$ is the extended real numbers set.
- $\mathbb{R}_{\geq 0} = \{x \in \mathbb{R} :\ x \geq 0\}$ and $\mathbb{R}_{>0} = \{x \in \mathbb{R} :\ x > 0\}$ are the sets of non negative and positive real numbers, respectively.
- Given $\rho \in \mathbb{R}_{>0}$ and $c \in \mathbb{R}^n$, $B(c, \rho) = \{x \in \mathbb{R}^n :\ \|x - c\| \leq \rho\}$ denotes the closed Euclidean ball of center c and radius ρ.
- Given $S \subset \mathbb{R}^n$, $\mathrm{int} S = \{x \in S :\ \exists \rho \in \mathbb{R}_{>0}, B(x, \rho) \subseteq S\}$ is the interior of the set S.

- For every $q \in \mathbb{N}$, $\Gamma_0(\mathbb{R}^q)$ denotes the set of functions which take values in $\mathbb{R} \cup \{+\infty\}$ and are proper, convex, lower semicontinuous on $\mathbb{R}^q$.
- $R(K) = \{y \in Y \mid y = K(x) \text{ for some } x \in X\} \subseteq Y$ is the range of the operator $K : X \to Y$.
- $N(K) = \{x \in X \mid K(x) = 0\} \subseteq X$ is the kernel of the operator $K : X \to Y$.
- $\mathcal{L}(X, Y)$ denotes the space of the linear and continuous operators between two Hilbert spaces X and Y.
- The matrix D is used to denote a discrete gradient operator, while D_v and D_h are the vertical and horizontal gradient operators, respectively.

Chapter 1

Machine Learning Theory

We revisit in this chapter the basic principles of machine learning. This general theory will be applied throughout the rest of the thesis, when we will talk about the more specific context of deep learning and when we will use neural networks applied to a specific problem of image deblurring.

1.1 Machine Learning Algorithms

First of all, a machine learning algorithm is usually defined as an algorithm able to "learn" from empirical data. But what does it mean that an algorithm learns? In [102] Mitchell gives the following definition:

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

In this definition, learning is the way to become capable of performing a task. Many kinds of problems have been addressed with machine learning. Tasks like classification, regression, translation, speech recognition, anomaly detection, self-driving are only some of the problems that have contributed to the success of machine learning in recent years.

In order to evaluate the abilities of a machine learning algorithm on a given task, we also need a quantitative measure of its performance. Usually this measure P is specific to the addressed task T. For example, for classification tasks we often measure the accuracy of the model, which is the proportion of examples for which the model gives the correct output, among the total number of examples [101]. For tasks such as regression other kind of measures are more suited, like metrics that give a continuous-valued score for each example. For image denoising, many related works evaluate the learning performances using the peak signal-to-noise ratio, often abbreviated PSNR, or the structural similarity measure (SSIM), which are measures specific for the quality of signal reconstruction. In particular, SSIM will be treated in the experimental part of this thesis, in Section 4.4, dealing with the problem of image deblurring.

Regarding the experience E, usually a dataset is available, which is a collection of examples of objects or events related to the task to be performed. If we consider a dataset of M elements, $\{x_1, \dots, x_M\}$, every example, represented as a vector $x_i \in \mathbb{R}^n$, is a collection of features that have been quantitatively measured from the object or event, and that we want the machine learning system to process. In this case, each entry of the single example x_i is a feature. Considering a neural network performing image denoising, for example, it is usual to work with a dataset of images, where the features to be processed are the values of the pixels.

Different types of learning paradigms are possible, depending on the type of dataset which is available during the learning process. The most common ones are classified as

- Unsupervised learning, in which the machine must learn to make sense of the data provided without a guide. This means that the algorithm classifies and organizes a series of input, studying common characteristics. For example, for the classification problem, the classes may not be known a priori, but must be learned by the machine, as the input to the system are unclassified examples.
  Unsupervised algorithms are useful if we want to learn properties of the dataset structure: by observing several examples of a random vector x, they can get information on the probability distribution $p(x)$ that generated the dataset. This is helpful for learning to draw new samples from the distribution, finding a manifold to which the data lies near, or clustering the data into groups of related examples.
  A classic unsupervised learning task is to find the best representation of the data, that preserves as much information as possible about the input x, while it is simpler or more accessible than x itself. An example of algorithm used for this, is given by the Principal Component Analysis (PCA) [56], which transforms data into a representation with a lower dimensionality than the original input.

- Supervised learning, aims to learn a function that maps an input to a desired output by looking at many examples of input-output pairs. It is like the target for each training input is provided by an instructor, who shows the machine what to do.
  Thus, in supervised learning, each example is a pair consisting of an input object x, as described above, and an associated target y. The target can be of different types, depending on the task to be performed. For example, if we want to classify an object, the target could be either a scalar symbolizing the correct class, or the probability to belong to that class. Instead, if we perform signal denoising, the target is usually the clean signal.
  The input-output pairs thus provided are then analyzed by the learning algorithm, and used to infer the underlying function which maps the inputs to the correct outputs. In this way, the hope is that a good training on the given examples will allow for the algorithm to determine the correct outputs also for unseen instances. This is not obvious, as explained in Section 1.2.1, because it requires the learning algorithm to be able to

generalize from the training data to unseen situations.
Typical tasks treated with supervised learning include classification and regression. These are useful in many different applications, like medical diagnoses, voice identification, handwriting recognition, spam detection and many other tasks. Some traditional supervised learning algorithms are Support Vector Machines (SVM) [19, 40], the k-nearest neighbors algorithm [1], or the decision tree [115].

Other learning paradigms exist. Some machine learning algorithms do not just experience a fixed dataset. For example, a reinforcement learning algorithm learns to perform a given task by interacting with the environment, so that there is a feedback loop between the learning system (the agent) and its own experience. In particular, during the learning process these algorithms find the best possible behaviour or path to take in a specific situation by being penalized when they make wrong decisions or rewarded when they make right ones. In this sense, the experience is not provided as a fixed dataset of examples, but as a system of positive and negative rewards to the agent actions.
Due to its generality, the field is studied in many disciplines, such as game theory, control theory, operations research, multi-agent systems, statistics and genetic algorithms. See [14, 130] for reference. Also other learning paradigms are possible, like semi-supervised or weakly supervised learning, but they are not treated in this thesis. Instead, from now on we will work in a purely supervised learning framework.

1.2 Function Estimation Model

Supervised learning involves learning from a set of data, that we call training set. Every element of the training set is an example of input-output pair, where the input maps to an output. In this context, the learning problem consists of inferring the function that maps between an input and its output, such that the learned function can be able to predict the output from future inputs, which may be different from those used to train the model.
Generalization is this ability we want to acquire, to perform accurately on new, unseen examples after having experienced the training set. To estimate this generalization ability, we typically measure the performance of the model on a test set of new examples, different from those used in training. However, in practice, when training a machine learning model, we have access to a single dataset of examples. The standard procedure is to set aside part of this original dataset, and to use it as a test set on which to evaluate the performance of the model, i.e., its generalization ability. This is known as the holdout method [85]. In this way, the test set is composed of examples, independent from the examples on which we train the model, but that follow the same probability distribution.

1.2.1 Expected vs Empirical Risk Minimization

To put it more formally, take $X \subseteq \mathbb{R}^n$ to be the vector space of all possible inputs drawn independently from a fixed but unknown distribution $P(x)$. Also, consider Y to be the vector space of all possible outputs, such that, for every input vector x an output $y \in Y$ is returned, according to a conditional distribution function $P(y|x)$, also fixed but unknown.
From the statistical learning theory perspective, we assume that there is a joint probability distribution $P(x, y)$ over $X \times Y$, called the data generating distribution, such that $P(x, y) = P(y|x)P(x)$. In this case, the dataset consists of random independent identically distributed (i.i.d.) samples drawn from $P(x, y)$. We take M such samples and we use them as training set,

$$\mathcal{S} = \{(x_1, y_1), ..., (x_M, y_M)\},$$

where every x_i is an input vector from the training data, and y_i is the corresponding output. The remaining samples are kept aside as test set.
Note that it is important to make i.i.d. assumptions, which imply that the examples in each dataset are independent from each other, and that the training set and test set are identically distributed, drawn from the same probability distribution.

The main goal of supervised learning is to infer the function which maps the inputs to the correct outputs, without specifically learning the assigned data, but generalizing beyond the examples of the training set. To do this, we select a family $\mathcal{H}$ of prediction functions, which are all the functions the algorithm will search through,

$$\mathcal{H} := \{h(., \theta)|\theta \in \Omega\}$$

parametrized by θ in the space of possible parameter values Ω. This is called the hypothesis space. The inference problem consists of finding the function $h(x, \theta^*) \in \mathcal{H}$ such that, for every sample (x, y) drawn from $P(x, y)$, the value $h(x, \theta^*)$ will represent an accurate prediction of the output value y.
Reformulating the problem, we look for the function that minimizes the errors caused by incorrect predictions. For this purpose we choose a cost function, or loss function, $L : Y \times Y \to \mathbb{R}$, which is a cost measure $L(h(x, \theta), y)$ of the errors between the output $h(x, \theta)$ predicted by the system and the real output y. The objective function we want to minimize is called generalization error or expected risk, and it is the expectation of the loss function with respect to the underlying probability distribution $P(x, y)$, given by

$$\mathcal{J}(\theta) = \mathbb{E}[L(h(x, \theta), y)] = \int_{X \times Y} L(h(x, \theta), y)dP(x, y). \tag{1.1}$$

In this definition, the variable θ represents the part of the learning system which must be adapted to find the function $h(x, \theta^*) \in \mathcal{H}$ that minimizes the expected risk $\mathcal{J}(\theta)$ on every possible input-output pair.

Now, even if we would like to minimize the expected risk, it is not possible to do this directly, because the joint probability distribution $P(x; y)$ is unknown by hypothesis, and the only available information are contained in the training set $\mathcal{S}$.

In order to solve this problem, the induction principle of empirical risk minimization (ERM principle), from [136], suggests to minimize an approximation of the expected risk $\mathcal{J}(\theta)$, constructed by averaging the loss function on the available examples of the training set $\mathcal{S}$. This estimate is called the training error, or empirical risk, $\mathcal{J}_M(\theta) : W \to \mathbb{R}$,

$$\mathcal{J}_M(\theta) = \frac{1}{M}\sum_{i=1}^{M} L(h(x_i; \theta), y_i). \tag{1.2}$$

The ERM principle assumes that the function $h(x, \theta^*)$ which minimizes $\mathcal{J}_M(\theta)$ over $\theta \in W$, results in an expected risk $\mathcal{J}(\theta^*)$ which is close to its minimum. This is shown in [135], where general theorems prove that the minimization of the empirical risk $\mathcal{J}_M(\theta)$, can provide a good estimate of the minimum expected risk, given that the size of the training set is large enough. Theoretically, the system is therefore able to generalize, to learn results that have general validity, from a training set of finite dimensions.

Note that generalization being the goal has interesting consequences for machine learning. In this case we do not have access to the function $\mathcal{J}(\theta)$ we want to optimize, instead, we have to use the empirical risk as a surrogate. But this also means that, since the objective function is only an approximation for the true goal, we may not need to fully optimize it; in fact, a low value of $\mathcal{J}_M(\theta)$ may yield better generalization performance than the global optimum $\mathcal{J}_M(\theta^*)$. This argumentation justifies the introduction of early stopping techniques in the learning process, as we will see in Section 2.3.1.

1.2.2 Model Capacity

To train a machine learning model, we search the system parameters θ that minimize the empirical risk on a given training set, then we test the model with the trained parameters on the test set, to evaluate its generalization ability.
In this process, the expected error on the test set is greater than or equal to the expected value of the training error, that is, the empirical risk $\mathcal{J}_M(\theta)$. The factors determining how well a trained model will perform are its ability to:

- Minimize the training error.

- Minimize the gap between training and test error.

These two factors correspond to the two central challenges in machine learning: underfitting and overfitting. In particular, underfitting occurs when the model is not able to obtain a sufficiently low error value on the training set. Overfitting occurs when the gap between the training error and test error is too large.

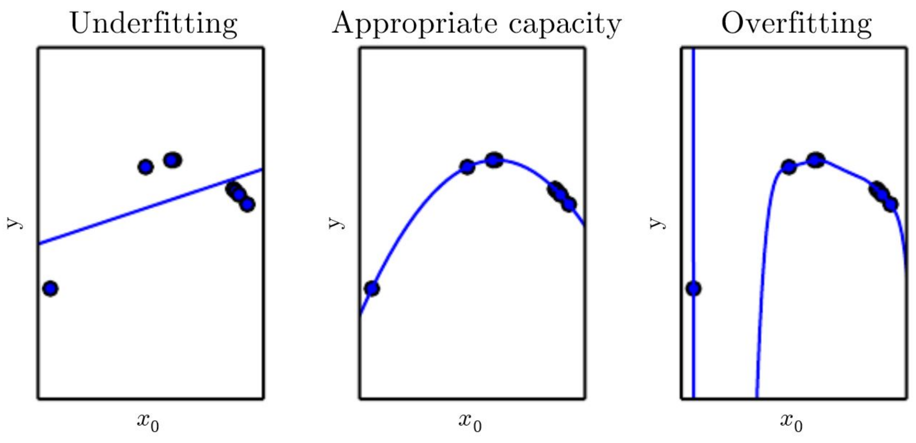


Figure 1.1: Consider training data generated by evaluating a quadratic function. *Left*: a linear function cannot capture the data curvature (underfitting). *Center*: a quadratic function generalizes well to unseen points. *Right*: The solutions of a polynomial of degree 9, fit to the data, pass through all the training points exactly, but it does not extract the correct structure (overfitting). Example from [56].

We can control whether a model is more likely to overfit or underfit by modifying its capacity. Informally, a model's capacity is its ability to fit a wide variety of functions. Models with insufficient capacity may struggle to fit the training set, falling into underfitting. Models with high capacity can solve complex tasks, but when their capacity is higher than needed they may overfit by memorizing properties of the training set that are not useful on the test set. Instead, machine learning algorithms will generally perform best when their capacity is adequate to the complexity of the task, and to the amount of training data they are provided with.

An example of this principle is shown in Figure 1.1. Here we compare a linear, a quadratic and a degree-9 polynomial predictors attempting to fit a training set drawn from a quadratic underlying function. The linear function is unable to capture the curvature in the data, so it underfits. The degree-9 predictor is capable of representing many functions that pass exactly through the training points, including the correct function, but there is little chance of choosing a solution that generalizes well when so many different solutions exist. In this example, the quadratic model matches the true underlying structure, so it generalizes well to new data.

From another point of view, this means that empirical risk and expected risk behave differently, depending on the model capacity. Typically, increasing the model capacity, training error decreases until it asymptotes to its minimum error value. Instead, generalization error assumes a U-shaped curve. The gap between these two errors increases, and eventually outweighs the decrease in training error, entering the overfitting regime above the optimal capacity. This general behaviour is illustrated in Figure 1.2.

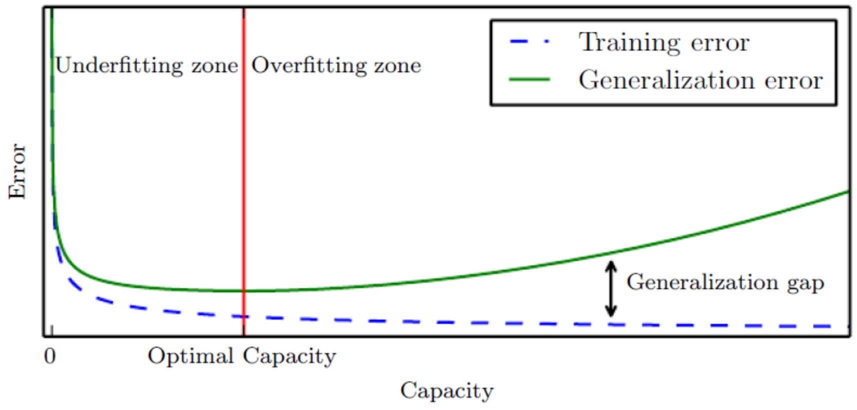


Figure 1.2: Relationship between model capacity and error, from [56].

As we can see, in general, while simpler functions are more likely to generalize, we must still choose a sufficiently complex model to achieve low training error.
The model's capacity can be modified by selecting the hypothesis space $\mathcal{H}$ of the functions the learning algorithm can choose from. In particular $\mathcal{H}$ determines the representational capacity of the model [56]. However, once the hypothesis space is chosen, empirical risk minimization can be a very difficult optimization problem, and in practice the learning algorithm does not find a minimizer, but only a function that significantly reduces the training error. This means that the effective capacity of the model may be less than its representational capacity. Considering the previous polynomial example, we choose $\mathcal{H}$ as the family of the polynomials of a particular degree n, and then the optimization process looks for the polynomial of degree n that best approximates the data.

The most important results in statistical learning theory show that the discrepancy between empirical risk and expected risk is bounded from above by a quantity that grows as the model capacity grows, but shrinks as the training set dimension increases [135, 137]. These bounds provide a theoretical justification for machine learning algorithms, but they are rarely used in the context of Deep Learning, in which we will present our results. This is in part because these bounds are often quite loose and in part because it can be difficult to determine the capacity of deep neural networks, [56]. Also, in Deep Learning, there is little theoretical understanding of the non-convex optimization problems that are involved in the training of neural networks, and the effective capacity of the models is often limited by the capabilities of the optimization algorithms used to train them.

1.2.3 Bias and Variance Decomposition

The field of statistics gives us many tools to understand generalization. We recall concepts such as parameter estimation, bias and variance, which are useful to formally characterize the notions of underfitting and overfitting.

Point estimation is the attempt to provide the best prediction of a certain quantity, that can be a single parameter, a vector of parameters in some parametric model, such as the weights in a neural network, or it can also be a whole function. In this case we will talk about function estimation, which is the same as a point estimator in a function space.
Let $\{x_1, ..., x_M\}$ be a set of M independent and identically distributed (i.i.d.) data points. A point estimator, or statistic, is any function of the data:

$$\Theta_M = g(x_1, ..., x_M). \tag{1.3}$$

This definition is very general, while almost any function qualifies as an estimator, a good estimator is a function whose output is close to the true underlying $\hat{\Theta}$ that generated the training data. Also, assuming that the true $\hat{\Theta}$ is fixed but unknown, since the data is drawn from a random process, the estimate Θ is a random variable.

Let us recall now some properties of point estimators.
The bias of an estimator is defined as the expected deviation from the true value of $\hat{\Theta}$,

$$\text{bias}(\Theta_M) = \mathbb{E}(\Theta_M) - \hat{\Theta}, \tag{1.4}$$

where the expectation is over the data (seen as samples from a random variable) and $\hat{\Theta}$ is the true underlying value used to define the data generating distribution. An estimator Θ_M is said to be unbiased if $\text{bias}(\Theta_M) = 0$, which implies that $\mathbb{E}(\Theta_M) = \hat{\Theta}$. An estimator is said to be asymptotically unbiased if $\lim_{M\to\infty} \text{bias}(\Theta_M) = 0$, which implies $\lim_{M\to\infty} \mathbb{E}(\Theta_M) = \hat{\Theta}$.

Another property of the estimator that we consider is how much we expect it to vary as a function of the data sample. Just as we computed the expectation of the estimator to determine its bias, we can compute its variance. The variance of an estimator is simply the variance

$$\text{Var}(\Theta_M) = \mathbb{E}[(\Theta_M - \mathbb{E}[\Theta_M])^2].$$

Alternately, the square root of the variance is called the standard error, denoted as $\text{SE}(\Theta_M)$. The variance, or the standard error, of an estimator provides a measure of how we would expect the estimate we compute from data to vary, as we independently sample the dataset many times from the underlying data-generating process.

Just as we might like an estimator to exhibit low bias, we would also like it to have relatively low variance. Bias and variance measure two different sources of error in an estimator. Bias measures the expected deviation from the true value of the function. On the other hand, variance provides a measure of the deviation from the expected estimator value that any particular sampling of the data is likely to cause. A way to negotiate this trade-off is to compare the mean squared error (MSE) of the estimates. In fact, the MSE measures the overall expected deviation between the estimator Θ_M and the true value of the parameter $\hat{\Theta}$,

$$\begin{aligned}
\text{MSE} &= \mathbb{E}\left[(\Theta_M - \hat{\Theta})^2\right] \\
&= \mathbb{E}\left[\Theta_M^2 + \hat{\Theta}^2 - 2\hat{\Theta}\Theta_M\right] \\
&= \mathbb{E}[\Theta_M^2] + \hat{\Theta}^2 - 2\hat{\Theta}\mathbb{E}[\Theta_M] \\
&= \text{Bias}(\Theta_M)^2 + \text{Var}(\Theta_M)
\end{aligned} \tag{1.5}$$

As we can see in equation (1.5), the evaluation of MSE incorporates both the bias and the variance of Θ_M. From this, to keep these quantities monitored, estimators for which the mean squared error is small are preferred.

In the context of machine learning, several researches have analysed overfitting through the decomposition of the generalization error, measured by MSE, into bias and variance [41, 52]. In this case, the bias is a measure of how much the model output, averaged over all possible datasets, differs from the desired function. The variance is a measure of how much the output varies between different datasets.
Increasing the model capacity tends to increase variance and decrease bias. At the beginning of training, the data still had little influence, so the variance is small, while the bias is large because the model output is far from the desired function (underfitting). Late in training, the bias is small because the algorithm has learned the underlying function. However, if trained too long, the algorithm will also learn the noise specific to the dataset, and the variance will be large. This is referred to as overfitting. It can be shown that the minimum total error will occur when the sum of bias and variance is minimal [91]. This is illustrated in Figure 1.3, where we see again the U-shaped curve of generalization error as a function of capacity.

To summarize, in machine learning it is important to choose the appropriate model capacity: if the model is too complex, it will generalize poorly to unseen data (overfitting); if the capacity is too low the model won't capture all the information in the data (underfitting). This is often referred to as the bias-variance trade-off, since a complex model exhibits large variance while an overly simple one is strongly biased. Empirically, there are several techniques to control this trade-off. Cross-validation, for example, is highly successful on many real-world tasks. Specific to machine learning, methods like early stopping (see Section 2.3.1), or regularization are well-known, as in support vector machines and regularization networks [51, 65].

Cross–Validation Method

Dividing the data into a fixed training set and a fixed test set can be problematic if the test set results small. In fact, a small test set implies statistical uncertainty around the estimated average test error, making it difficult to claim that an algorithm works better than another on the given task.

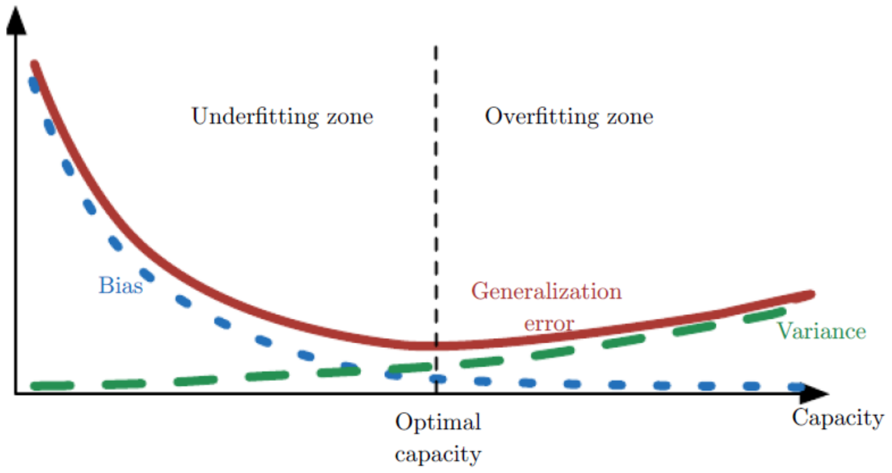


Figure 1.3: Example from [56], of the relationship between model capacity and generalization error. Here the dependence of the latter on the concepts of bias and variance is highlighted.

When the dataset has hundreds of thousands of examples, this is not a serious issue. When the dataset is too small, alternative procedures enable one to use all of the available data for the estimation of the mean test error, at the price of increased computational cost. These procedures are based on the idea of repeating the training and testing computation on different randomly chosen splits of the original dataset. The most common is the $k-$fold cross-validation procedure, in which a partition of the dataset is formed by splitting it into k non-overlapping subsets. The test error may then be estimated by taking the average test error across k trials. On trial i, the i-th subset of the data is used as the test set and the rest of the data is used as the training set.

Regularized Models

So far, the method we have discussed to modify a learning algorithm, is to increase or decrease the model's representational capacity, by adding or removing functions from the hypothesis space $\mathcal{H}$ of solutions the learning algorithm is able to choose.

The behavior of our algorithm is strongly affected not just by how large we take $\mathcal{H}$, but by the specific functions we pick from $\mathcal{H}$. We can thus control the performance of our algorithm by preferring certain solutions in the hypothesis space over others. This means that all the functions are eligible, but maybe one is preferred over another, and the unpreferred solution will be chosen only if it fits the training data significantly better than the preferred one.

More specifically, we can construct a regularized model that learns a function $h(x, \theta)$ by adding to the cost function a penalty term, for example on the norm of θ, which has the effect of narrowing the set within which the parameters are chosen. This is essentially equivalent to imposing regular conditions on the class of functions realized by the model [54]. For example, in the case Tikhonov regularization, we modify the training criterion to include weight decay. In this case the empirical risk we minimize is a sum of both the error on the training and a regularizer, that expresses a preference for the parameters to have smaller squared ℓ^2-norm. Then, given a training set $\mathcal{S} = \{(x_1, y_1), ..., (x_M, y_M)\}$, the function we minimize is of the type,

$$\mathcal{J}_M(\theta) = \frac{1}{M}\sum_{i=1}^{M} L(h(x_i;\theta), y_i) + \lambda\theta^T\theta \tag{1.6}$$

where λ is a value that controls the strength of our preference for smaller parameters. When $\lambda = 0$, we impose no preference, while larger λ forces the parameters to become smaller. Minimizing $\mathcal{J}_M(\theta)$ then results in a choice of parameters that make a trade-off between fitting the training data and being small. Even though the model is capable of representing functions with much more complicated shape, weight decay encourages it to use a simpler function described by smaller coefficients.
Adding a penalty term to the cost function is a classic regularization strategy, used in different contexts. This strategy is introduced from a different and more formal point of view in Chapter 3. In particular, Tikhonov regularizer and other regularization terms will be reviewed in Section 3.3.3, when approaching the problem of image reconstruction.

In Machine Learning other ways of expressing preferences for different solutions exist, both implicitly and explicitly. Together, these approaches are known as regularization. Then, in this context, we can say that regularization is any modification we make to a learning algorithm in order to reduce its generalization error, i.e., in order to solve overfitting.

1.2.4 Loss Functions

An important aspect of the design of a machine learning model is the choice of the loss function. In fact it has great influence on the solution $h(., \theta^*)$ found by the learning algorithm, and it also affects the convergence rate of the minimization process.
Given an input-output pair (x, y), the loss function $L : Y \times Y \to \mathbb{R}$ represents the price $L(h(x, \theta), y)$ we are willing to pay by predicting $h(x, \theta)$ instead of y.

There are various factors involved in choosing a loss function, such as the task we approach, and the model we use. Also, we may want the loss to have specific properties. For example, the loss may require a gradient large and predictable enough to be used in the training process by gradient based learning algorithms.

We can see in Figure 1.4 some of the most common loss functions, which can be classified into two major categories depending on the type of learning task we are dealing with: regression losses and classification losses. In particular, some of the most used for regression are:

- the mean squared loss function (also known as the ℓ^2-norm):

$$L(h(x, \theta), y) = (y - h(x, \theta))^2 \tag{1.7}$$

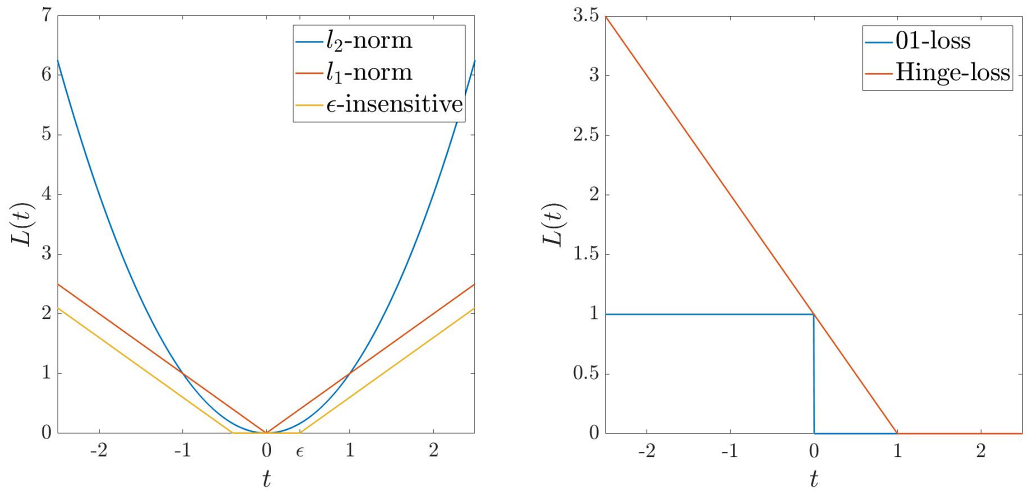


Figure 1.4: Different types of loss function in one variable t, with: *left)* $t = h(x,\theta) - y$ for regression; *right)* $t = yh(x,\theta)$ for binary classification.

- the absolute value loss (also known as the ℓ_1-norm):

$$L(h(x,\theta), y) = |y - h(x,\theta)| \tag{1.8}$$

- the ϵ–insensitive loss proposed by Vapnik, if $\epsilon > 0$:

$$L(h(x,\theta), y) = \begin{cases} 0 & \text{if } |y - h(x,\theta)| \leq \epsilon \\ |y - h(x,\theta)| - \epsilon & \text{otherwise} \end{cases} \tag{1.9}$$

For classification tasks, given the binary nature of classification, a natural loss function would be the 0-1 indicator function, which takes the value 0 if the predicted output $h(x,\theta)$ is the same as the true output y, 1 if the predicted output is different. For binary classification with $Y = \{-1, 1\}$, the 0-1 indicator function can be written as

$$L(h(x,\theta), y) = H(-yh(x,\theta)) \tag{1.10}$$

where H is the heaviside step function, defined as

$$H(t) = \begin{cases} 0 & \text{if } t < 0 \\ 1 & \text{if } t \geq 0. \end{cases} \tag{1.11}$$

However, this loss function is non-convex and non-smooth, and finding the optimal solution is an NP-hard combinatorial optimization problem. As a result, it is better to substitute continuous, convex loss functions which are tractable for commonly used learning algorithms. For example, with output space $Y = \{-1, 1\}$, a more tractable loss is the Hinge loss,

$$L(h(x,\theta), y) = \max(0, 1 - yh(x,\theta)) =: |1 - yh(x,\theta)|_+ \tag{1.12}$$

where $h(x,\theta)$ is the classifier score. This loss is used for maximum-margin classification, most notably for Support Vector Machines. In fact, the Hinge loss penalizes predictions such that $yh(x,\theta) < 1$, corresponding to the notion of a margin in a SVM.

Finally, the most common loss for classification problems may be the cross-entropy loss function. This loss measures the performance of multi-class classification models, whose outputs are the probabilities of the different classes. In this case, given an object x, y is usually chosen as the one-hot vector, with entry 1 corresponding to the true class of x. Then, if the probability of class i estimated by the model is $p_i = h_i(x,\theta)$, we can use cross-entropy to get a measure of dissimilarity between y_i and p_i,

$$L(h(x,\theta), y) = -\sum_i y_i \log p_i \tag{1.13}$$

In particular, as we can see in Figure 1.5, cross-entropy loss increases as the predicted probability diverges from the actual label, and heavily penalizes the predictions that are confident but wrong.

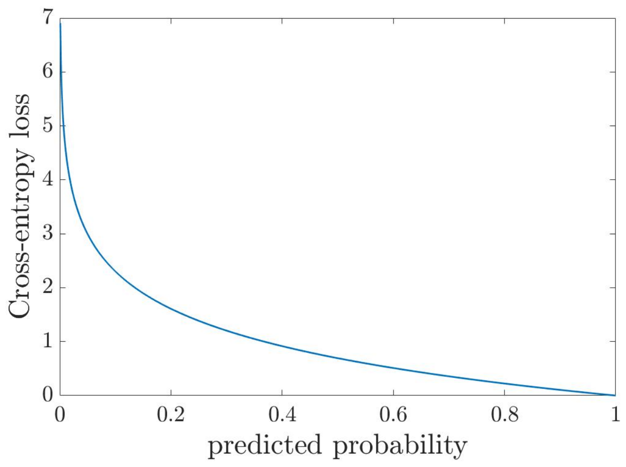


Figure 1.5: Cross–entropy loss function for multi-class classification, when true label is equal to 1.

1.2.5 Hyperparameter Selection

A common trait in machine learning systems is that they are usually parametrized by a set of hyperparameters that we can use to control the behavior of the learning algorithm. These are used to configure various aspects of the learning algorithm, so that they significantly affect the resulting model's performance, and the time required to train and test the model [33]. Examples of hyperparameters are the number and the size of layers, that define the structure of neural networks; but also the threshold ϵ, when using a $\epsilon-$insensitive loss function, or the step size of the iterative learning algorithm are all different types of hyperparameters.

The values of these hyperparameters are not adapted by the learning algorithm itself, like the other trainable parameters. This applies, for example, to all hyperparameters that control model capacity. If learned on the training set, such hyperparameters would aggressively increase the model capacity, resulting in overfitting.
To make this clearer, let's consider the polynomial example in Figure 1.1. If we treat the degree of a polynomial equation fitting a regression model as a trainable parameter, this would just raise the degree up until the model perfectly fit the data, giving small training error, but bad generalization performance. Instead the polynomial degree is a hyperparameter, chosen outside of the training.

Since the goal is to find the model having the best performance on new data, the simplest approach to compare different models (with different hyperparameters) is to evaluate their performance on new data, different from that used for training, as we do with the test set.
The subset of data used to guide the hyperparameters selection is called the validation set. However, validation set cannot contain examples from the test set. In fact, we discussed how a test set is used to estimate the generalization error, after the learning process has been completed. It is important that the test examples are not used to make choices about the model, including hyperparameters tuning.
Therefore, after putting the test set aside, we split again the training data into two disjoint subsets. Typically, one uses about 80% of the training data for training and 20% for validation [56]. The one used to learn the trainable parameters is typically called training set, even though this may be confused with the larger pool of data used for the entire training process. The other subset is the validation set, used to estimate the generalization error during or after training, allowing for the hyperparameters to be updated accordingly. Finally, when the entire learning process, comprising both empirical risk minimization and hyperparameter optimization, is complete, the trained model with hyperparameters having the best performance on the validation set is selected [15], and its generalization error is estimated on the test set.

In the polynomial example, the learning process implies that we train polynomial of different degrees on the examples of the training set. The polynomial that achieves the minimum generalization error on the validation set is selected as learning algorithm, and its final performance is evaluated on the test set.

Hyperparameter selection is an open research field. Specially in deep learning, the training process already requires a great amount of computational resources and time, and hyperparameter search is commonly performed manually, via rules-of-thumb [70, 73], or by grid-search methods.

Chapter 2

Deep Learning Theory

Deep Learning is a particular branch of machine learning, with great power and flexibility, which provides a powerful framework for supervised learning.
Deep Learning models, also referred to as artificial neural networks (ANNs), are engineered systems inspired by the biological brain, consisting of the interconnection of neurons. Neurons can be seen as nodes of an oriented network, provided with processing capacity. They receive weighed signals from the environment or from other neurons, as inputs that are processed. Then, the output of the neuron is sent to other neurons or to the network output, through other weighed connections.
A network is specified by i) the type of operations the neurons perform, ii) the values of the weights in the connections, which are determined by learning algorithms, iii) its structure: number of nodes, their arrangement in multiple layers, type of connections. By adding more layers and more neurons within a layer, a deep network can represent functions of increasing complexity. Most tasks that consist of mapping an input vector to an output vector, can be performed by Deep Learning, given sufficiently large models and sufficiently large datasets of labelled training examples. In particular, in recent years, Deep Learning popularity greatly increased, thanks to the availability of more powerful computers and larger datasets that allowed experimentation and the development of new techniques to train deeper networks.

The purpose of this chapter is to introduce the mathematical theories underlying Deep Learning, the algorithms used to train neural network structures, as well as providing some historical perspectives about neural networks development. For this purpose, we start by introducing the network's units: the formal neurons. We see how to combine them to form complex and different structures, like convolutional neural networks, which will be applied in the experimental part of this thesis. Then we see various aspects of the optimization process involved in the training of these deep models, in order to tune their internal weights.

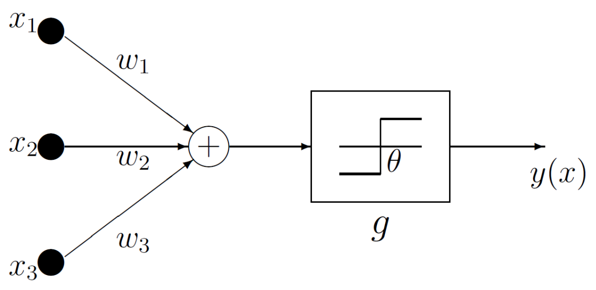


Figure 2.1: Formal neuron structure, based on the model proposed by [99], with $g(t) \equiv sign(t)$. Picture taken from [60].

2.1 Neural Networks History

Even though Deep Learning only recently acquired global acknowledgement, its history dates back to the early 1940's with the first mathematical model of an artificial neuron proposed by McCulloch and Pitts in 1943 [99], and revisited by Rosenblatt in 1958 [116].

2.1.1 The Formal Neuron

In the simplest version proposed by McCulloch and Pitts, the artificial neuron, or formal neuron, is based on the principle of functioning of the biological neuron. The inputs to the formal neuron are multiplied by weights, representative of the synaptic connections, and the weighted sum of the inputs is compared to a threshold value. By applying an activation function, the neuron gives output 1, if the sum of the inputs is greater than the threshold, -1 (or 0) otherwise.
In mathematical terms, given an input vector $x \in \mathbb{R}^n$, denoting with $\omega \in \mathbb{R}^n$ the weights vector, and with $y \in \{-1, 1\}$ the output of the neuron, and indicating with $\xi \in \mathbb{R}$ the threshold value, the output of the neuron w.r.t. its input can be written as

$$y(x) = g\left(\sum_{i=1}^{n} \omega_i x_i - \xi\right) \equiv g(\omega^T x - \xi),$$

where g is the neuron activation function, in this case taken as the *sign* function:

$$g(t) \equiv sign(t) = \begin{cases} 1 & \text{if } t \geq 0 \\ -1 & \text{if } t < 0 \end{cases}$$

This model of formal neuron is reported in Figure 2.1.

2.1.2 Rosenblatt's Single Layer Perceptron

In 1958, Rosenblatt reinterpreted the formal neuron as a binary classifier, able to sort the elements of a given dataset into two groups.
He sees the input $x \in \mathbb{R}^n$ as a generic vector, whose components are the features of the object to be classified. Assigning proper values to the weights vector $\omega \in \mathbb{R}^n$, the neuron is able to perform a binary classification, returning $y(x) = 1$ or $y(x) = -1$, based on the sign of the linear function $a = \omega^T x - \xi$.
Moreover, Rosenblatt's major achievement is the intuition that this model can actually learn from data. In fact, he devised a fairly simple algorithm that allows to learn the correct synaptic weights ω from the dataset itself. The weights and the threshold value are determined by a learning process starting from a training set of M input-output pairs,

$$\mathcal{S} = \{(x_m, y_m) \mid x_m \in \mathbb{R}^n,\ y_m \in \{-1, 1\},\ m = 1, ..., M\}$$

where y_m is the correct classification associated to the input x_m. This learning algorithm is called Perceptron, and it is reported in Algorithm 1. Once the formal neuron is trained on the samples of $\mathcal{S}$, we can use its generalization skills to classify new inputs not belonging to $\mathcal{S}$.

Now, let's see how the Perceptron algorithm works. We want to train the neuron to correctly classify the samples of the training set, and this is done if the weights ω and the threshold ξ are determined in such a way that

$$\begin{cases} \omega^T x_m - \xi \geq 0 & if \quad y_m = \ \ 1 \\ \omega^T x_m - \xi < 0 & if \quad y_m = -1 \end{cases} \qquad m = 1, ..., M \tag{2.1}$$

From a geometric point of view, system (2.1) search for the hyperplane $H = \{x \in \mathbb{R}^n \mid \omega^T x = \xi\}$, which separates the two sets

$$\mathcal{A} = \{x_m \mid (x_m, y_m) \in \mathcal{S},\ y_m = 1\} \qquad \mathcal{B} = \{x_m \mid (x_m, y_m) \in \mathcal{S},\ y_m = -1\}.$$

This way, the existence of ω and ξ, which solve (2.1) can be assured if and only if the sets $\mathcal{A}$ and $\mathcal{B}$ are linearly separable. Also, (2.1) admits solution if and only if the system below admits solution,

$$\begin{cases} \omega^T x_m - \xi > 0 & x_m \in \mathcal{A} \\ \omega^T x_m - \xi < 0 & x_m \in \mathcal{B} \end{cases} \qquad m = 1, ..., M \tag{2.2}$$

We can add dummy components $x_{m_0} = -1$ and $\omega_0 = \xi$, so that the input vectors and the weights vector become vectors with $n + 1$ components, $x_m = (-1, x_{m_1}, \ldots, x_{m_n})^T$, and $\omega = (\xi, \omega_1, \ldots, \omega_n)^T$. In this way, the threshold is considered as an additional bias term with weight ξ, and finding the correct synaptic weights and threshold corresponds to solving the system

$$\begin{cases} \omega^T x_m > 0 & x_m \in \mathcal{A} \\ \omega^T x_m < 0 & x_m \in \mathcal{B} \end{cases} \qquad m = 1, ..., M \tag{2.3}$$

w.r.t $\omega \in \mathbb{R}^{n+1}$. Here we can presume, without loss of generality, that

$$||x_m|| = 1, \quad m = 1, \dots, M.$$

To solve system (2.3), Rosenblatt proposes to find ω by using an algorithm, which uses only one sample of the training set at each iteration. As we can see in Algorithm 1, at each cycle, ω is updated in correspondence of the incorrectly classified examples, adding to the current value of ω a correction term: if the example $x_m \in \mathcal{B}$ and the Perceptron wrongly predicts $sign(\omega(k)^T x_m) = 1$, the weight update is given by $\omega(k+1) = \omega(k) - x_m$, while the threshold is increased by 1. If x_m belongs to class $\mathcal{A}$ and the Perceptron wrongly predicts $sign(\omega(k)^T x_m) = -1$, the weight correction is $\omega(k+1) = \omega(k) + x_m$, while the threshold is decreased by 1.

Algorithm 1 Perceptron training algorithm, from [60].

```
Given the input $x_m \in \mathbb{R}^n$, with $||x_m|| = 1$ and the target $y_m \in \{-1, 1\}$, $m = 1, \dots, M$. Set
$\omega(0) = 0$, $k = 0$, classified = 0.
while classified < $M$ do
  for $m = 1, \dots, M$ do
    if $sign(\omega(k)^T x_m) = y_m$ then
      classified = classified + 1
    else
      $\omega(k+1) = \omega(k) + y_m x_m$
      $k = k + 1$
    end if
  end for
  if classified < $M$ then
    classified = 0
  end if
end while
```

This means that along the iterations the update rule can make samples, that were previously well classified, not correctly classified; the algorithm needs to go through the whole training set $\mathcal{S}$ as long as all the samples are not correctly classified. However, it can be shown (see [60]) that if the sets $\mathcal{A}$ and $\mathcal{B}$ are linearly separable, the algorithm determines a weight vector $\bar{\omega}$ in a finite number of iterations, such that all the training set samples are correctly classified, i.e. it is satisfied

$$y_m = sign(\bar{\omega}^T x_m) \quad m = 1, \dots, M. \tag{2.4}$$

Perceptron suffers from major limitations, because the samples sets $\mathcal{A}$ and $\mathcal{B}$ are not linearly separable for many classification problems. A simple example is given by the XOR function, an operation on two binary inputs, that returns 1 when exactly one of these input values is equal

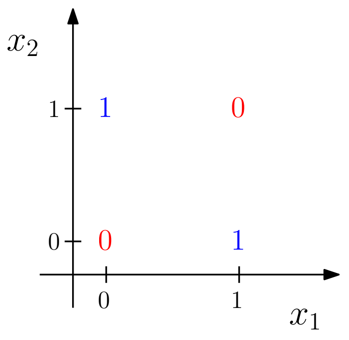


Figure 2.2: XOR function.

to 1. Otherwise, it returns 0. Figure 2.2 shows how a linear model is not able to represent the XOR function. In fact there is no straight line capable of separating the different outputs.

It was the inability of the basic Perceptron to solve such simple problems that led, in part, to a reduction in interest in neural network research during the 1970s. To overcome the limitations of Perceptron, it is necessary to use more complex structures, which learn a different feature space in which a linear model is able to represent the solution.
This is achieved, for example, by Multilayer Perceptrons, which can solve arbitrary classification problems.

2.1.3 Multilayer Feed-Forward Neural Networks

The limits of the Perceptron motivated the study of more complex architectures consisting of several layers of formal neurons connected in chain. These architecture are called Multilayer Perceptrons (MLPs), or feedforward neural networks, in which it is assumed that there are no feedback connections, nor connections between the neurons of the same layer.

The first layer is a set of n input nodes, without processing capacity, that are associated to an input vector $x \in \mathbb{R}^n$. The other formal neurons are organized in $K \geq 2$ different layers, arranged in a chain structure, with each layer being a function of the layer that preceded it. In particular, each of the first $K-1$ internal hidden layers consists of neurons (the hidden units) that act in parallel, each representing a vector-to-scalar function. Every neuron receives input from the neurons of the previous layer, computes its own activation value, and then the output contributes to the inputs of the neurons of the successive layer. In this sense, MLPs are said to be fully connected: all the neurons in a layer are connected to each neuron in the successive layer. Finally, the last layer is the output layer, which return the outputs $x^{(K)}$ of the network.

The length K of the chain gives the depth of the model. Since each layer of the network may have a different number of neurons, we denote with d_k the dimension of the k^{th} layer, and the maximum layer's size determines the width of the model.
Recalling Section 1.2.5, the number of layers and their size are examples of hyperparameters

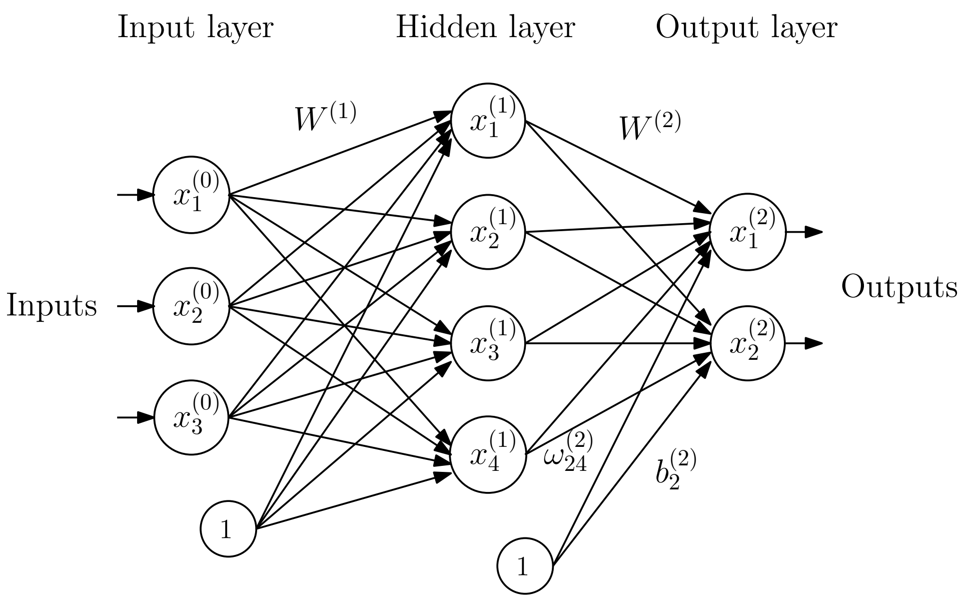


Figure 2.3: Feedforward neural network structure with depth $K = 2$.

that are usually determined by performing different simulations, and evaluating the different performances on the validation set.

The Multilayer Perceptron structure is shown in Figure 2.3. As we can see, for $k = 1, ..., K$, a weight $\omega_{ji}^{(k)}$ is associated to each oriented arc between the neurons $x_i^{(k-1)}$ and $x_j^{(k)}$. This weight represents the entity of the synaptic connection between the two neurons. Furthermore, like in Rosenblatt's Perceptron, biases are added as additional input $x_0^{(k)} = 1$ with weight $b_j^{(k)}$.

Suppose that every formal neuron of the k^{th} layer applies an activation function $g^{(k)} : \mathbb{R} \to \mathbb{R}$ to the weighted sum of the layer's inputs; this is usually a nonlinear activation function like, for example, a ReLU or a sigmoid function (see Section 2.1.3). Given an input vector $x^{(0)} \in \mathbb{R}^{d_0}$, the output of each neuron in the network can be computed as

$$x_j^{(k)} = g^{(k)} \left(\sum_{i=1}^{d_{k-1}} \left(\omega_{ji}^{(k)} x_i^{(k-1)} + b_j^{(k)} \right) \right) \qquad \text{for } k = 1, ..., K, \quad j = 1, ..., d_k \tag{2.5}$$

where d_k is the number of neurons in the k^{th} layer. Reformulated in matrix form it becomes

$$x^{(k)} = g^{(k)}(W^{(k)} x^{(k-1)} + b^{(k)}) \qquad \text{for } k = 1, ..., K, \tag{2.6}$$

where $W^{(k)} \in \mathbb{R}^{d_k \times d_{k-1}}$ is the weight matrix of the k^{th} layer and $b^{(k)} \in \mathbb{R}^{d_k}$ is the bias vector. Thus, if $x^{(0)} \in \mathbb{R}^{d_0}$ is the input to the network, equation (2.6) gives the expression to calculate the final output $x^{(K)}$, as a composition of functions connected in chain.

In the context of deep learning, a training set of M examples of input-output pairs, $\mathcal{S} = \{(x_1, y_1), ..., (x_M, y_M)\}$ is available. By entering the network one of these examples x_i, the network will give in output $x_i^{(K)} = h(x_i; \theta)$, where θ is the parameters vector containing all the weights and biases $\{(W^{(1)}, b^{(1)}), ..., (W^{(K)}, b^{(K)})\}$.
The training examples directly specify what the output layer should do at each point x_i: it must produce a value $h(x_i, \theta)$ that matches y_i. Instead, the behavior of the hidden layers is not directly specified by the training data, but it is the learning algorithm that has to modify the internal parameters to produce the desired output. Thus, chosen a cost function L, the problem becomes

$$\underset{\theta}{\text{minimize}} \, \frac{1}{M} \sum_{i=1}^{M} L(h(x_i; \theta), y_i) \tag{2.7}$$

which is a nonlinear and non-convex problem, due to the nature of the function $h(x_i; \theta)$.

In theory, MLPs allow, under appropriate hypotheses on the activation functions, to approximate almost any function on a compact set. Specifically, the universal approximation theorem [72] states that two-layer feedforward networks, with sigmoid transfer functions in the hidden layer, can be trained to approximate any continuous function on a compact set with the desired level of accuracy, provided sufficiently many hidden units are available. In particular, this means that, unlike Perceptron, MLPs are able to solve classification problems on linearly non-separable set.
However, even if we know that a large MLP will be able to represent any particular function, we are not guaranteed that the training algorithm will be able to learn that function. In fact, determining the parameters through the training process becomes a nonlinear and non-convex optimization problem, that can be very difficult to deal with.
Nevertheless, over the years considerable progress has been made in the use of deep neural networks to calculate approximate solutions based on gradient methods. This has been possible because the gradient of the objective function in (2.7) with respect to the parameter vector θ can be calculated by using the chain rule of calculus. We will talk about this differentiation rule, known in this context as backpropagation, in Section 2.3.3. Before, we conclude the analysis of the architecture by saying something about the hidden units activation functions, and by introducing a specific type of feedforward neural network, the convolutional neural network, that will be used in the experimental part of this thesis.

Neuron's Activation Functions

Activation functions are what give neural networks their nonlinear capabilities. Apart from the fact that they are supposed to be nonlinear differentiable function, there are multiple choices for them and the design of hidden units is an extremely active research field. Here, we review two common families of activation functions, which are the most used in Deep Learning practice.
The first one is given by the sigmoids. These functions are defined in $\mathbb{R}$, and monotonically increase between two finite values, with a characteristic S-shaped curve, or sigmoid curve. In

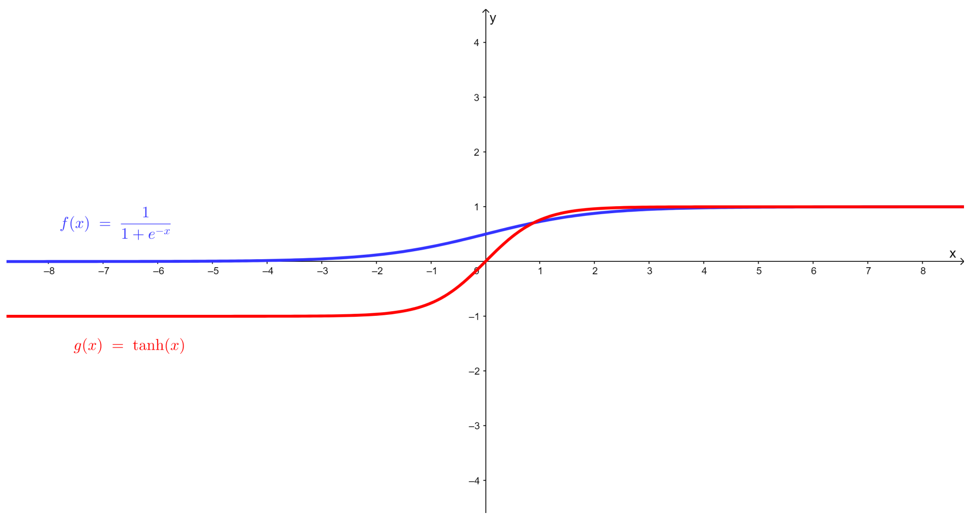


Figure 2.4: Graphics of logistic function (in blue) and tanh function (in red).

particular, the most used are the logistic function, which gives an output in the interval $(0, 1)$,

$$g(t) \equiv \frac{1}{1+e^{-t}}, \tag{2.8}$$

and the hyperbolic tangent, which gives an output in $(-1, 1)$,

$$g(t) \equiv \tanh(t/2) = \frac{1-e^{-t}}{1+e^{-t}}. \tag{2.9}$$

as shown in Figure 2.4.
Sigmoid functions have been frequently used since they have a nice interpretation as the firing rate of a neuron. However, in practice, the sigmoid units have recently been put aside. In fact, they have a major drawback: they saturate across most of their domain, when $|t|$ is large, and are strongly sensitive to their input only when t is near 0. The widespread saturation of sigmoid units can make gradient-based learning very difficult, the gradient in these regions being almost zero. For this reason, their use as hidden units in feedforward networks is now limited.
Instead, the most used activation function in modern neural networks is the rectified linear unit, or ReLU, defined by,

$$g(t) \equiv \max\{0, t\}. \tag{2.10}$$

Being a piecewise linear function of two pieces, it preserves many of the properties that make linear models easy to optimize with gradient-based methods. In [88], it is empirically demonstrated that deep convolutional neural networks with ReLUs train several times faster than their equivalents with tanh units. For example, ReLUs have the desirable property that they do not require input normalization to prevent them from saturating. One drawback to ReLUs is that they cannot learn via gradient-based methods on examples for which their activation is negative. For this, generalizations of ReLU are available, that use a nonzero slope α when, in equation (2.10), $t < 0$. For example, leaky ReLu [95] fixes α to a small value like 0.01, while a parametric ReLU, or PReLU, treats α as a learnable parameter [66]. In these cases, the activation function can be written as

$$g(t, \alpha) \equiv \max\{0, t\} + \alpha \min\{0, t\}. \tag{2.11}$$

It is necessary to specify that, although ReLU and its generalizations are not differentiable at $t = 0$, they are still efficiently used with gradient based learning methods. In fact, in practice one can safely disregard the non-differentiability of these hidden unit activation functions, because deep learning libraries, and software implementations of learning algorithms, usually return one of the one-sided derivatives, rather than raising an error. This may be heuristically justified by observing that gradient-based optimization on a digital computer is subject to numerical error anyway. When a function is asked to evaluate $g(0)$, it is very unlikely that the underlying value is really 0. In fact, it is likely to be some small value that is rounded to 0. In some contexts, more theoretically pleasing justifications are available, but these usually do not apply to neural network training. Also, a smooth version of ReLU is introduced in [48] to deal with this problem. This is the Softplus function,

$$g(t) \equiv \ln(1 + e^t), \tag{2.12}$$

whose output is strictly positive. Still this function has not empirically shown advantages over the standard rectified linear units.
The graphics of ReLU and Softplus are reported in Figure 2.5.

2.1.4 Convolutional Neural Networks

As we saw in Section 2.1.3, MLPs transform the input through a series of hidden layers. Each hidden layer is made up of a set of neurons that work independently and do not share any connections; also every neuron is fully connected to all neurons in the previous layer. This means that, giving an RGB image of size $(200 \times 200 \times 3)$ as input to an MLP, with the pixels passed as units in the input layer, every single fully connected neuron in the first hidden layer has 120000 weights, one for every connection to the input units. Depending on the number of layers, and number of units in every layer, the training process can become computationally too expensive. Also, the huge number of parameters can quickly lead to overfitting.

Convolutional neural networks, or CNNs, from [90], are a specialized kind of neural network for processing data with a clear grid-structured topology. In particular, they are developed for computer vision tasks, with the explicit assumption that the inputs are multichannel digital images. This allows to encode certain properties into the architecture that reduce the amount of parameters, and make the forward function more efficient.

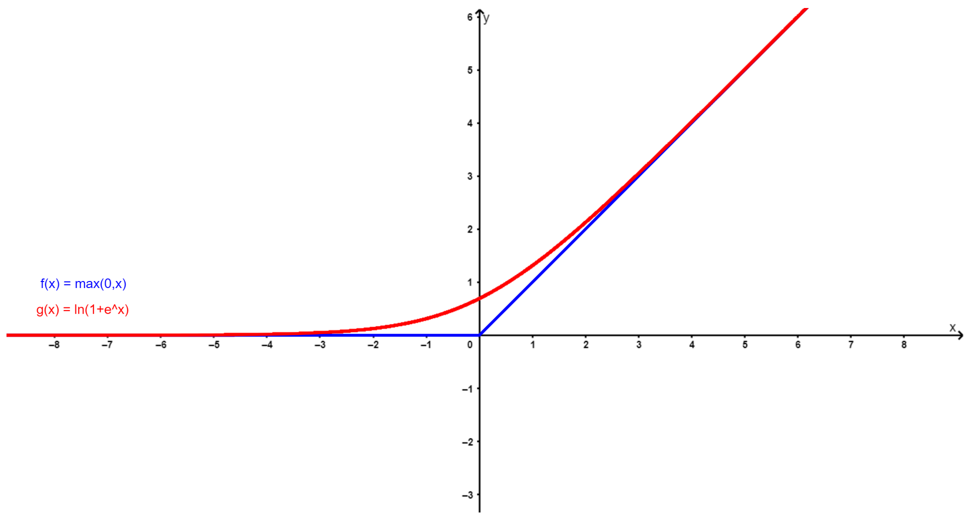


Figure 2.5: Graphics of ReLU function (in blue), and Softplus function (in red).

First, the neurons in the layers of a CNN are arranged in three dimensions: width and height, which denote the spatial dimensions, and depth, as seen in Figure 2.6. Every layer transforms the 3D input volume to a 3D output volume.
Secondly, CNNs are not fully connected: instead they use specialized patterns of sparse connections that are very effective in computer vision, so that each unit in a layer is connected only to a local region of the previous layer. In particular, the neurons still compute a dot product of their weights with the input, followed by a nonlinearity. However, using a weights matrix smaller than the input, the number of connections with the previous layer is limited. The spatial extent of the connectivity is a hyperparameter called the receptive field of the neuron.

Sparse connections are useful because decreasing the number of connections means that we need to store fewer parameters. This reduces the memory requirements of the model, improves its statistical efficiency and requires fewer operations computing the output at the same time. In fact, considering a single layer only, if there are m inputs and n outputs, then fully connected matrix multiplication requires $m \times n$ parameters, and the algorithms used in practice have $O(m \times n)$ runtime per example. If the number of connections of each output is limited to k, then the the approach requires only $k \times n$ parameters and $O(k \times n)$ runtime. For many practical applications, it is possible to obtain good performance on the task while keeping k several orders of magnitude smaller than m.

We use three main types of layers to build CNN architectures: convolutional layer, pooling layer, and eventually, fully-connected layer as in MLPs. These layers are stacked to form a full CNN architecture. The most common model of CNN stacks a few convolutional layers, follows them with pooling layers for downsampling, and repeats this pattern until the image has been reduced spatially to a small size. In some cases it is also useful to insert fully-connected layers.

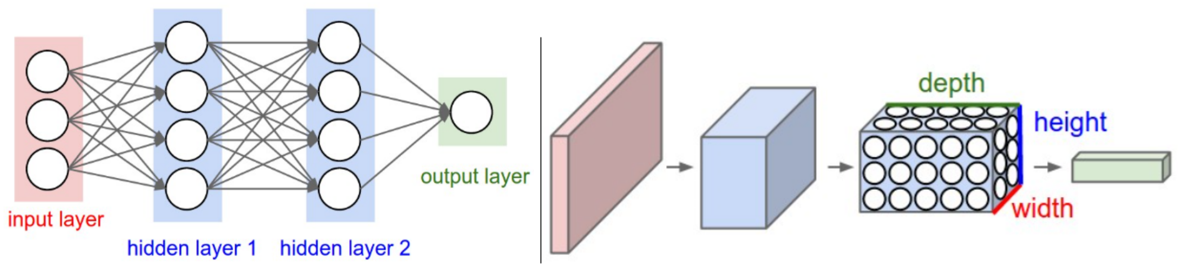


Figure 2.6: *Left*: a fully connected feedforward neural network structure, with 3 layers. *Right*: A CNN structure, where the neurons of every layer are arranged in three dimensions (width, height, and depth), as depicted in one of the layers. The input layer holds an image, so its width and height are the image dimensions, while the depth gives the color channels (RGB).

In classification tasks, for example, the last fully-connected output layer holds the class scores. In this way, CNNs transform the original image, layer by layer, to the final output.

Convolutional Layer

Convolutional networks are neural networks that use convolution in place of general matrix multiplication in at least one of their layers, the so called convolutional layer. This kind of layer consists of two stages. In the first stage, the layer performs several convolutions in parallel to produce a set of linear activations, and add some bias terms to them. In the second stage, each linear activation is run through a nonlinear activation function, such as the ReLU.

In its most general form, convolution is an operation on two functions of a real-valued argument, and it is typically denoted with an asterisk " $*$ ". Its formal definition is reported below.

Definition 2.1. *Let $f, h \in L^1(\mathbb{R})$. The convolution of f and h is defined as*

$$(f * h)(t) = \int f(t-t')h(t')dt' = \int f(t')h(t-t')dt'. \tag{2.13}$$

In practice, convolving h with f produces a new function, $f * h$, whose values are a sort of (integral) mean of the values of f, where the values of h play the role of weights. More in detail, for each t, we have to shift f by t and multiply it with h, then integrate the product over the real line. The resulting function will be a transformed version of f, whose shape depends on the features of h.

Working with digital data, we assume that f and h are defined only on integers t, so that a discrete version of convolution (2.13) is required,

$$(f * h)(t) = \sum_{t'=-\infty}^{\infty} f(t')h(t-t') = \sum_{t'=-\infty}^{\infty} f(t-t')h(t'). \tag{2.14}$$

In convolutional network terminology, the first argument is often referred to as input, and the second argument as the kernel or filter, while the output is usually referred to as feature map. Because each element of the input and kernel are variables to be stored, we usually assume that these functions are zero everywhere but in the finite set of points for which we store the values. This means that in practice, we can implement the infinite summation as a summation over a finite number of array elements.

Finally, we often use convolutions over more than one axis at a time. In particular, discrete convolutions are applied between inputs x, which are usually multidimensional array of data (the multichannel images), with kernels W, which are multidimensional array of parameters (the filters) that are adapted by the learning algorithm. In Deep Learning context, we refer to these multidimensional arrays as tensors.
In the case of a multidimensional input tensor x and kernel W, a 2-D convolution operator, applied on their spatial dimensions, becomes

$$(x * W)(i,j) = \sum_m \sum_n x(m,n)W(i-m,j-n) = \sum_m \sum_n x(i-m,j-n)W(m,n). \qquad (2.15)$$

An example of 2-D convolution is reported in Figure 2.7.

A typical filter is spatially small, but extends through the full depth of the input volume. For example, a filter on a first layer of a CNN might have $5 \times 5 \times 3$ size (i.e. 5 pixels width and height, and depth 3, because of the color channels).
During the forward pass, the convolutional layer convolves the input with the filter, by sliding it across the width and height of the input volume, and computing a weighted sum at any position. We add a bias term to these sums (the same for all positions). Then, each linear activation is run through a nonlinear activation function, such as the ReLU, and a 2-D activation map, the feature map, is produced.
Usually, there is an entire set of filters in each convolutional layer, and each of them produces a separate 2-D feature map. Stacking these feature maps along the depth dimension, the output volume of the layer is produced.

To be more specific, the output volume, as a function of the spatial input volume ($w_{in} \times h_{in}$), is controlled by four hyperparameter: the number of filters (Q), their spatial extent (F), the stride with which they are applied (S), and the amount of padding (P) used on the input border.
The stride with which we slide the filter is important because, when the stride is 1 then we move the filters one pixel at a time. When the stride is 2 or more, then the filters jump 2 or more pixels at a time as we slide them around. This will produce spatially smaller output volumes. Conversely, using padding around the input borders allows to control the changes in spatial sizes, and to preserve the information of the input at the borders. In fact, without padding, the spatial sizes would be reduced by a small amount for each convolutional layer, and the information at the borders would be lost. Most commonly, the usual setting is zero padding $P = (F-1)/2$, with stride $S = 1$, to ensure that the input volume and output volume

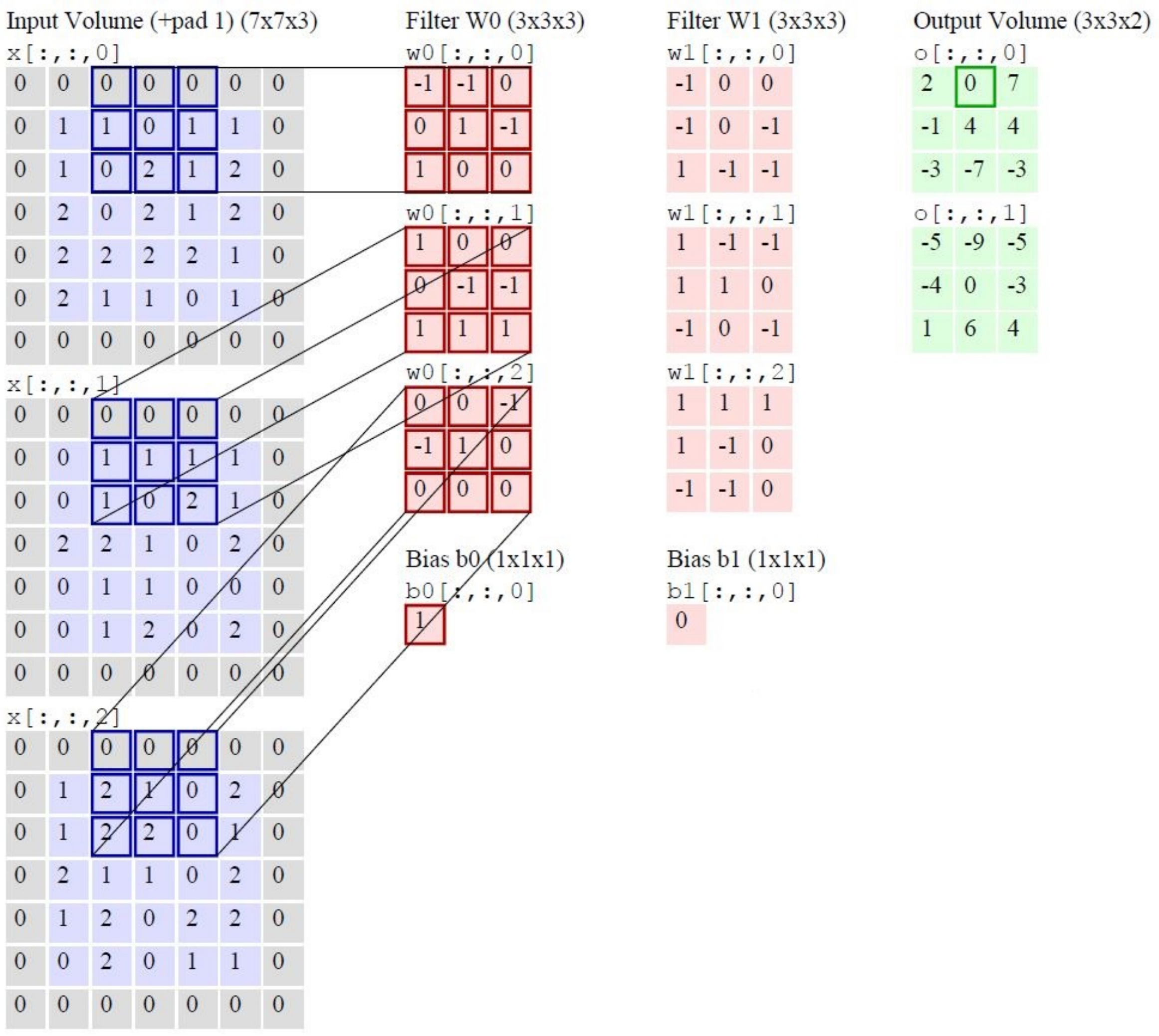


Figure 2.7: Example of 2-D convolution of an input volume $(7 \times 7 \times 3)$ with 2 filters of size $(3 \times 3 \times 3)$. The padding is set as $P = 1$, and stride $S = 2$, so that the output dimension becomes $(3 \times 3 \times 2)$. It is highlighted the linear combination to compute the element $x_{out}(1, 2, 0)$. Example from [56].

will have the same sizes spatially.
In general,

- The number of filters we convolve gives the depth of the output, $d_{out} = Q$.
- The width and height of the output can be calculated with the formula

$$\begin{aligned} w_{out} &= (w_{in} - F + 2P)/S + 1 \\ h_{out} &= (h_{in} - F + 2P)/S + 1. \end{aligned}$$

It is to be noted that two peculiar properties of convolutional neural networks derive from the use of the convolution operator. As already mentioned, the first is the sparse connectivity. When the convolution is made with a kernel of spatial dimension (3×3), only $(3 \times 3 \times d_{in})$ input units in x affect one output unit. These units are called the receptive field of that particular output unit.
Also CNNs perform parameter sharing, meaning that each element of a filter is used at every position of the input, except perhaps some of the boundary pixels, depending on padding. This does not affect the runtime of forward propagation but it does further reduce the storage requirements of the model to $F \times F \times d_{in}$ parameters for each filter, for a total of $(F \times F \times d_{in}) \cdot Q$ weights and Q biases for each layer. Convolution is thus more efficient than dense matrix multiplication in terms of memory requirements and statistical efficiency.

Remark 2.1. When working with images, we usually think of the input and output of the convolution as being 3-D tensors. Software implementations usually work in batch mode, meaning that they process more images in parallel. So they actually use 4-D tensors, with the fourth axis indexing different examples in the batch. For simplicity, we omitted the batch axis in our description.

Pooling Layer

In a convolutional neural network architecture, it is common to periodically insert a pooling layer between successive convolutional layers. Pooling layers do not have internal parameters to be trained, during the learning process. Instead, their function is to reduce the spatial sizes of the representation, without loosing too much information, to further reduce the amount of parameters and computation required in the network, and hence to control overfitting.

A pooling function replaces the input unit at a certain spatial location with a summary statistic of the nearby input units. For example, the max pooling operation [151] reports the maximum output within a rectangular neighborhood. Other popular pooling functions include the average of a rectangular neighborhood, the ℓ^2-norm, or a weighted average based on the distance from the central pixel. In [23], guidance is provided on what types of pooling should be used in various situations.
Eitherway, the pooling layer performs a downsampling operation along the spatial dimensions,

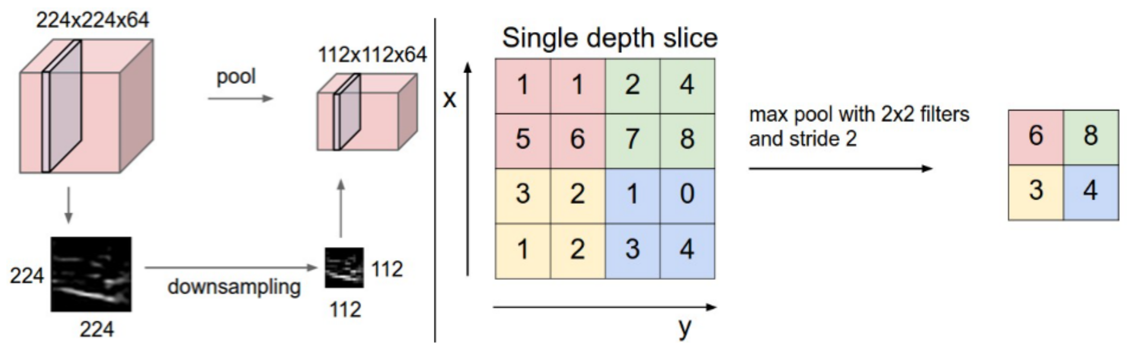


Figure 2.8: *Left*: The input volume of size $(224 \times 224 \times 64)$ is downsampled by a pooling of rectangular regions of size (2×2) and stride $S = 2$, into an output volume of size $(112\times112\times64)$. *Right*: example of max-pooling operation, performed by applying the max on spatial regions of size (2×2), with stride $S = 2$. Picture from [80].

while the depth dimension remains unchanged. To deduce the spatial sizes of the output, it is necessary to know two hyperparameters: the rectangular neighbor in which to apply the downsampling operation (F), and the stride (S). Given these values and the input dimension $(w_{in} \times h_{in})$, the width and height of the output can be calculated as

$$\begin{aligned} w_{out} &= (w_{in} - F)/S + 1 \\ h_{out} &= (h_{in} - F)/S + 1. \end{aligned}$$

An example of max pooling is reported in Figure 2.8.

Pooling can improve the computational efficiency of the network because the next layer has fewer inputs to process. Also, when the number of parameters in the next layer is a function of its input size, this reduction can also result in improved statistical efficiency and reduced memory requirements for storing the parameters.

Furthermore, pooling helps to make the representation approximately invariant to small translations of the input. This means that if we translate the input by a small amount, the values of most of the pooled outputs do not change. Invariance to local translation can be a useful property if we care more about whether some feature is present than exactly where it is. For example, when determining whether an image contains a face, the exact position of the eyes is not required. We just need to know that there are two eyes, and their approximate location.

Efficiency in Convolution

Modern convolutional neural network applications often involve architectures containing more than one million units. Powerful implementations exploiting parallel computation resources are essential. For convolutional neural network training and inference, specialized libraries provide highly tuned implementations for standard routines such as discrete convolution, exploiting parallelization and GPUs support.
At the same time, we observe in Section 3.2.1 that, for the convolution theorem, the convolution operation is equivalent to convert both the input and the kernel to the frequency domain using a Fourier transform, to perform point-wise multiplication of the two signals, and to convert the result back to the image domain with an inverse Fourier transform. For some problem sizes, this can be faster than the naive implementation of discrete convolution.

In the experimental part of this thesis we approach the image reconstruction problem. All the operations concerning network training and inference, convolutions included, are implemented employing libraries specific for neural networks. Conversely, we use the conversion to the frequency domain, to apply the blur on the images of the dataset, as explained in Section 3.2.1.

2.2 Neural Networks Training

When adapting the general theory of Section 1.2.1 in neural networks context, to obtain good performance we need to choose an appropriate neural network architecture and we need to train the network, to determine the vector θ of all the internal parameters, the weights and biases, that give the best results on a particular task.

In particular, for a fixed architecture the parameters choice is generally performed by defining a suitable subset of the available data, the training set,

$$\mathcal{S} = \{(x^{(i)}, y^{(i)}) \mid x^{(i)} \in X,\ y^{(i)} \in Y,\ i = 1, \dots, M\},$$

with $X \subseteq \mathbb{R}^n$ the vector space of all possible inputs, and Y the vector space of all possible outputs, and by solving an optimization problem of the type

$$\underset{\theta}{\operatorname{argmin}}\, \mathcal{J}_M(\theta), \quad \text{with } \mathcal{J}_M(\theta) = \frac{1}{M} \sum_{i=1}^{M} L(h(x^{(i)}; \theta), y^{(i)}) \tag{2.16}$$

where the loss function $L(h(x^{(i)}; \theta), y^{(i)})$ is the error related to the i^{th} sample of the training set, which measures the discrepancy between the desired data $y^{(i)}$ and the output of the network $h(x^{(i)}; \theta)$. Different measures are used to calculate the error, depending on the type of variable y, and on the addressed task, as seen in Section 1.2.4. In particular, the loss function must be differentiable, because neural networks are usually trained with gradient-based learning methods.

As for the other machine learning models, the purpose of network training is not to learn an exact representation of the training data, by interpolating it, but rather to build a statistical model of the process which generates these data. Assuming this, the choice of the network architecture, the loss function, and the training strategy are crucial to obtain a neural network that has a good generalization ability.
For example, talking about the depth of the network, we said that a feedforward network with a single hidden layer is sufficient to represent any function, but the layer may be infeasibly large and may fail to learn and generalize correctly. In many circumstances, using deeper models can reduce the number of units required to represent the desired function and can reduce the amount of generalization error. In fact, it is empirically shown that greater depth seems to result in better generalization for a wide variety of tasks, [8, 57, 88].

In practice, the depth and width of the architecture, and the other hyperparameters are usually chosen with heuristic techniques, through experimentation guided by monitoring the error on the validation set. In the case of feedforward neural networks, for example, a basic strategy for choosing the architecture is structural stabilization, from [15]. This strategy consists in choosing the number of units and layers, through the training of different networks in which the number of neurons (layers) is increased or decreased. For each structure the internal parameters are determined by solving (2.16), and the performances of the different networks are compared on a validation set, not included in $\mathcal{S}$. Then, the best network on the validation set is chosen and its generalization capacity is finally evaluated using a test set, which must not have been used nor for the choice of architecture and hyperparameters, nor for the determination of internal parameters.

2.2.1 Optimization Challenges

Among all of the optimization problems involved in Deep Learning, the most difficult is neural network training, which, fixed an architecture, aims at finding the internal network parameters that minimize the empirical risk, as in equation (2.16). The minimization of training error is, in general, a nonlinear optimization problem, in which we encounter many computational difficulties.

- Many challenges arise from the fact that we are approaching problems of non-convex optimization. When minimizing highly non-convex error functions we must avoid getting trapped in suboptimal local minima. In fact, in the case of neural networks, it is possible to find a very large number of them. If local minima with high cost are common, this could pose a serious problem for gradient-based optimization algorithms.
  It remains an open question if there are many local minima with high cost in neural networks of practical interest. The current idea is that, for sufficiently large neural networks, most local minima have low cost and are therefore acceptable solutions in parameter space [43, 58, 120]. Dauphin et al., [43], argue that the difficulty arises in fact not from local minima but from saddle points.

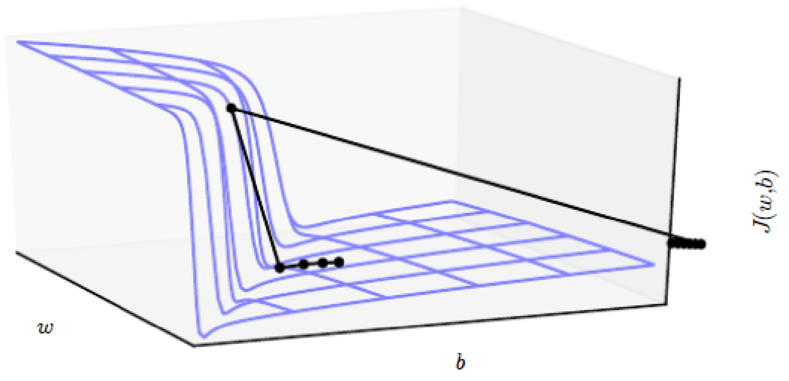


Figure 2.9: The objective function for very deep neural networks often contains sharp nonlinearities in parameter space, that are presented as cliffs. Figure from [56].

Strong nonlinearities of $\mathcal{J}_M(\theta)$ create steep valleys and flat regions in the surface of the objective function. While empirical data show that the gradient descent algorithms are in many cases capable of escaping from saddle points [58], a major problem is due to large flat regions, where the cost function has a constant value. In these regions the gradient and the Hessian are null and do not provide a guide for the optimization algorithms. In a non-convex optimization problem, as happens in Deep Learning, these regions can therefore be a problem, when they correspond to a high value of the objective function. Another structure present in the surface of the objective function, when using neural networks with many layers, are steep regions that look like cliffs, as illustrated in Figure 2.9. These result from the multiplication of several large weights together. When the parameters approach such cliffs, a gradient descent algorithm can move them very far, losing much of the work done, and slowing down the optimization process, [56].

- Ill-conditioning of the Hessian can manifest by causing gradient based methods to get stuck, in the sense that even very small steps increase the cost function. The result is that learning becomes very slow.

- Complex global structure makes it impossible to ensure the global convergence of the algorithms if the descent direction, which is locally the best, does not point towards regions of lower cost, albeit distant. The local steps performed by the optimization algorithm can therefore lead along a path that moves downhill, but far from any solution, or along a trajectory that leads to the solution, but is unnecessarily long. How to deal with problems that have a complex global structure is currently an active research field. Currently, studies are moving towards the search of good initial points. This derives from the idea that many of the problems related to the trajectory can be avoided, if one is able to initialize the learning within a region of space directly connected to a solution, through a path that the local descent can follow.

- Exploding gradients, and vanishing gradients, are another difficulty that optimization algorithms must overcome when neural networks are extremely deep.
  In some cases, gradients will be vanishingly small, effectively preventing the weights from changing their values. In the worst case, this may completely stop the neural network from further training. This phenomenon can arise for example when using sigmoid activation functions, such as the hyperbolic tangent, which has gradients in the range (0, 1), and corresponds to the presence of flat regions in the parameters space.
  On the other hand, when activation functions are used whose derivatives can take large values, the related exploding gradient problem can occur. This problem implies the multiplication of several large weights together, so that learning becomes unstable. The cliff structures described earlier are an example of exploding gradient.

- Finally, high dimensionality of the vector θ containing all the internal parameters, and of the training set $\mathcal{S}$ can lead to serious computational problems.
  For example, in machine learning it is known that for a good generalization a large training set is needed, but this leads to higher computational costs. In fact, deep learning algorithms minimize loss functions which can be decomposed into a sum over all the examples of the training set. So, a big training set can cause long training times, or in the worst case, it can create memory issues, which makes training impossible. Similar reasoning can be applyed to θ.
  These problems can be treated as hyperparameters tuning, via experimentation guided by monitoring the minimization process. Also, specialized optimization algorithms are able to deal with very large training sets, as we will see in Section 2.3.1.

These difficulties in neural network training have caused the development of specialized optimization techniques for solving the optimization problem (2.16).

2.3 Gradient-based Learning Methods

Minimizing the empirical risk in equation (2.16), when using neural network models, is not a trivial problem. Most of the successful approaches are gradient-based learning methods, which aim to generate a sequence of iterates $\{\theta_k\}_{k\in\mathbb{N}}$, in the parameter space Ω, so that $\mathcal{J}_M(\theta_k)$ is decreased along the iterations. In particular, the decrease in the objective function is obtained by moving along a descent direction, which is found exploiting first–order information.

The simplest learning procedure consists of using the standard gradient descent algorithm, which iteratively adjusts the internal parameters θ as follows:

$$\theta_{k+1} = \theta_k - \gamma_k \nabla_\theta \mathcal{J}_M(\theta_k). \tag{2.17}$$

where γ_k is the stepsize at the k^{th} iteration, called learning rate in Deep Learning context, and where $\nabla_\theta \mathcal{J}_M(\theta_k)$ is the gradient of the objective function w.r.t the internal parameters.
At each iteration, a forward pass through the network is required for all the examples of the training set, in order to compute the objective function, and its gradient. For this reason, standard gradient descent is usually referred to as batch gradient descent, since an entire "batch" of data must be considered before updating the parameters θ.
In fact, the gradient of the empirical risk function is an average over all the examples of the training set,

$$\nabla_\theta \mathcal{J}_M(\theta) = \frac{1}{M} \nabla_\theta \sum_{i=1}^{M} L(h(x^{(i)}, \theta), y^{(i)}), \tag{2.18}$$

which can be computed by applying a specialized Backpropagation algorithm. How to compute this gradient w.r.t the networks internal parameters will be treated in detail in Section 2.3.3. Standard batch gradient descent method is reported in Algorithm 2.

Algorithm 2 Batch Gradient Descent update.

```
Input: Training set $\mathcal{S} = \{(x^{(i)}, y^{(i)}) \mid x^{(i)} \in X,\ y^{(i)} \in Y,\ i = 1, \dots, M\}$
  Learning rate schedule $\gamma_1, \gamma_2, \dots$
  $\theta_0$ initial set of parameters

  for $k = 0, 1, \dots$ do
    Compute gradient estimate: $g_k = +\frac{1}{M} \nabla_\theta \sum_{i \in \mathcal{S}} L(h(x^{(i)}, \theta_k), y^{(i)})$
    Apply update: $\theta_{k+1} = \theta_k - \gamma_k \, g_k$
  end for
```

The properties of batch gradient descent are well known. When the optimization problem is convex, if the learning rate is sufficiently small, the algorithm converges to a global minimum of the objective function $\mathcal{J}_M(\theta)$; to a local minimum, in the non-convex case, [20].

However, one drawback of batch gradient descent is that the average of the gradients over the entire training set must be calculated at each iteration, as in (2.18), with computational cost $O(M)$. This means that when the training set contains many examples, it becomes difficult to apply gradient descent, for the high computational cost and the memory requirement.

To reduce these costs, stochastic techniques have been studied, which calculate the average over a smaller number of examples than those of the entire training set.

2.3.1 Stochastic Gradient Descent algorithm

The problem can be solved using a stochastic gradient descent method (SGD), from [113]. Instead of calculating the gradient of $\mathcal{J}_M(\theta)$ exactly, the idea is to approximate it, by calculating the average in (2.18) over a smaller number of examples than those available.

In particular, the stochastic gradient descent algorithm, in Algorithm 3, makes an approximation of the gradient on a single example of the training set $\mathcal{S}$. Specifically, at each iteration

k, a single example $(x^{(i_k)}, y^{(i_k)}) \in \mathcal{S}$ is randomly chosen. An estimate of the true gradient is computed based on the error $\mathcal{J}^{(i_k)}(\theta)$ on that example, as

$$\nabla_\theta \mathcal{J}^{(i_k)}(\theta_k) = \nabla_\theta L(h(x^{(i_k)}, \theta), y^{(i_k)}), \tag{2.19}$$

and the parameters are updated according to this approximation,

$$\theta_{k+1} = \theta_k - \gamma_k \nabla_\theta \mathcal{J}^{(i_k)}(\theta_k). \tag{2.20}$$

In this way each iteration is very economical from the computational point of view, involving only the calculation of the gradient corresponding to the selected example.
Usually, to randomly select the examples the training set is shuffled in advance, and as the algorithm sweeps through the mixed training set, it performs the above update for each example. Several passes can be made over the training set until the algorithm converges. If this is done, the data can be shuffled for each pass to prevent cycles.

Algorithm 3 Stochastic Gradient Descent update.

```
Input: Training set $\mathcal{S} = \{(x^{(i)}, y^{(i)}) \mid x^{(i)} \in X,\ y^{(i)} \in Y,\ i = 1, \dots, M\}$
  Learning rate schedule $\gamma_1, \gamma_2, \dots$
  $\theta_0$ initial set of parameters

  for $k = 0, 1, \dots$ do
    Randomly select an example $(x^{(i_k)}, y^{(i_k)})$ from the training set $\mathcal{S}$.
    Compute gradient estimate: $g_k = \nabla_\theta L(h(x^{(i_k)}, \theta_k), y^{(i_k)})$
    Apply update: $\theta_{k+1} = \theta_k - \gamma_k\, g_k$
  end for
```

It is to be noted that this algorithm is not deterministic. Rather, $\{\theta_k\}_{k=1,\dots}$ is a stochastic process whose behaviour is determined by the random sequence $\{i_k\}_{k=1,\dots}$.
Also, hopefully this procedure has the same results as the batch gradient descent algorithm, despite the random noise introduced by the simplification of the gradient. In fact, while each direction $-\nabla_\theta \mathcal{J}^{(i_k)}(\theta_k)$ may not be a descent direction of $\mathcal{J}_M(\theta)$ in θ_k, if it is descent direction in average, then the sequence $\{\theta_k\}_{k=1,\dots}$ can go towards a minimum of $\mathcal{J}_M(\theta)$.
This hope is supported by empirical results. Parameter updates have a greater variance that can cause large fluctuations in the objective function. It is not guaranteed that the optimization algorithm will reach a minimum (even local) in a reasonable time. However, empirical results often shows that SGD can find a value of the function that is low enough to be useful. Also, in [20, 84] it is shown that by gradually decreasing the learning rate, the SGD algorithm shows the same results as the batch gradient descent: it almost certainly converges to a global or local minimum, respectively in the convex and non-convex case.

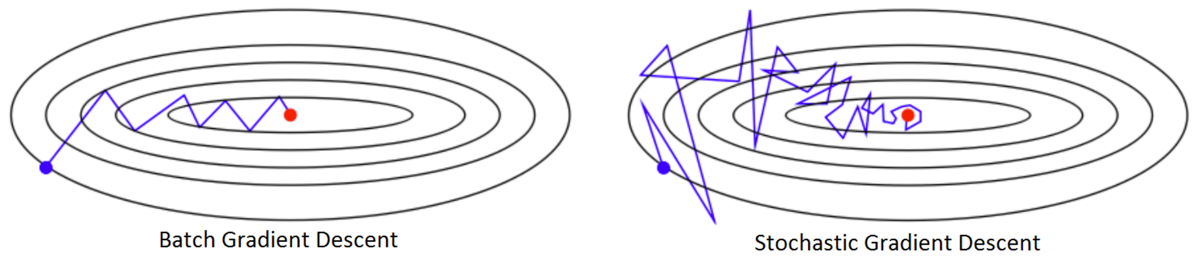


Figure 2.10: *Left*: Batch Gradient descent. *Right*: Stochastic Gradient Descent. Heavy fluctuations are visible in this case.

Batch vs Stochastic Gradient Descent

Calculating the true gradient in the deterministic approach is more expensive than calculating $\nabla_\theta \mathcal{J}^{(i_k)}(\theta_k)$ in the stochastic gradient method, although it can be expected that, by using all the examples in a single iteration, the update is qualitatively better.
This is true, in fact we already said that the SGD update has a greater variance that can cause large fluctuations in the objective function and can prevent full convergence to the minimum, depending on the level of the noise in the gradient estimate. Figure 2.10 shows these fluctuations for SGD; the weights do not move precisely down the gradient at each iteration, because of its noisy estimate.
Another advantage of deterministic methods is that, due to the additive structure of the objective function $\mathcal{J}_M(\theta_k)$, they can easily benefit from parallelization since most of the calculations are in the evaluation of the objective function and in the gradient computation.

Despite these advantages, there are practical and theoretical reasons for following a stochastic approach, that we discuss through the comparison between the iterate of SGD (2.20) and that of the deterministic method (2.17).

- First of all, the noise at each iteration can be useful to find better solutions. Nonlinear networks usually have multiple local minima of different depths. While batch gradient descent converges to the minimum of the basin in which the parameters are initially placed, even if its value is high with respect to that of the global minimum, in stochastic learning the noise present in the updates can result in the weights jumping into the basin of another local minimum, possibly deeper than before. This has been demonstrated in simplified cases [69]. In particular it has been suggested that noise helps to escape ”sharp minima” which generalize poorly, [28, 71, 82].

- Given a training set of M examples, a whole passage through it (i.e., an epoch) is required to allow the deterministic method to perform a parameter update, while SGD makes M steps in an epoch, by performing a parameter update for each training example $(x^{(i)}, y^{(i)}) \in \mathcal{S}$. As we need to calculate the gradients for the whole dataset to perform just one update, batch gradient descent can be very slow or intractable for datasets that do not fit in memory. In this case, SGD becomes the only possible choice.

- It is known how a deterministic approach can minimize $\mathcal{J}_M$ with a rapid convergence speed; from [21], if $\mathcal{J}_M$ is strongly convex, then there exists a constant $\rho \in (0;1)$ such that, for every $k \in \mathbb{N}$, the empirical risk will satisfy

$$\mathcal{J}_M(\theta_k) - \mathcal{J}_M^* \leq \mathcal{O}(\rho^k), \tag{2.21}$$

where $\mathcal{J}_M^*$ is the minimum value of $\mathcal{J}_M(\theta)$. This convergence speed is known in optimization as R-linear speed [109]; we will simply refer to this convergence by calling it linear. From (2.21), [21] shows that, in the worst case, the total number of iterates in which the error can be above a given $\epsilon > 0$ is proportional to $\log(1/\epsilon)$. This means that, with a cost per iterate proportional to M, the total work required to obtain an ϵ-optimality for a deterministic gradient method is proportional to $M \log(1/\epsilon)$.
The convergence speed of a basic stochastic method is slower than a deterministic method; for example if $\mathcal{J}_M$ is strictly convex and each i_k is uniformly chosen from $\{1, ..., M\}$, for each $k \in \mathbb{N}$, the iterates of SGD defined in (2.20) satisfy the sublinear convergence property

$$\mathbb{E}[\mathcal{J}_M(\theta_k) - \mathcal{J}_M^*] = \mathcal{O}\left(\frac{1}{k}\right). \tag{2.22}$$

However, it is to be noted that neither the per-iteration cost nor the right-hand side of (2.22) depend on the size M of the training set. This means that the total work required to obtain an ϵ-optimality for SGD is proportional to $1/\epsilon$. Even if $1/\epsilon$ can be greater than $M \log(1/\epsilon)$ for small values of M and ϵ, the comparison is in favor of SGD when we use a large amount of data, so when M is very large. This is usually the case in Deep Learning problems.
Also, it should be noticed that the asymptotic analysis does not highlight the advantages that the SGD algorithm obtains after a small number of steps. Especially with large datasets, the rapid progress that is obtained by evaluating the gradient on the first examples, is more important than a slow asymptotic convergence.

- Another important characteristic of SGD is that, if we knew the underlying probability distribution, we would have the same convergence speed expressed by (2.22) also for the error on the expected risk, $\mathcal{J}(\theta_k) - \mathcal{J}^*$, where $\mathcal{J}^*$ is the minimum value of $\mathcal{J}(\theta)$. Specifically, by applying the k-th iterate (2.20) of SGD with the gradient computed on a example chosen independently from the underlying distribution $P(x, y)$, we find that

$$\mathbb{E}[\mathcal{J}(\theta_k) - \mathcal{J}^*] = \mathcal{O}\left(\frac{1}{k}\right). \tag{2.23}$$

that is, a sublinear convergence speed on the expected risk. This means that the behavior of SGD in terms of minimizing the empirical risk $\mathcal{J}_M(\theta)$ or the expected risk $\mathcal{J}(\theta)$ is practically identical until all the M examples of the training set are used the first time (for the first epoch), because it is like selecting samples without reintroduction until the whole training set is exhausted. So, if M is very large, the SGD algorithm can be seen as an optimizer simultaneously of $\mathcal{J}_M(\theta)$ and $\mathcal{J}(\theta)$.

- Finally, when using batch gradient descent methods that compute the average of the gradients over the training set, there can be redundant calculations in case the dataset contains equal examples, or in case many examples give a similar contribution.
  SGD eliminates this redundancy by performing one update at a time. It is therefore usually much faster on large redundant datasets. For instance, consider the simple case where a training set of size 1000 is composed of 10 copies of a set of 100 examples. Averaging the gradient over all the 1000 samples gives the same result as computing the gradient based on just the first 100 samples. Thus, using batch gradient, it is like computing the same quantity ten times, before doing one parameter update. In practice, examples rarely appear more than once in a dataset, but there are usually clusters of patterns that are very similar. This redundancy can make batch learning much slower than stochastic learning.

2.3.2 Minibatch Gradient Descent

A compromise between the exact calculation of the gradient of $\mathcal{J}_M(\theta)$ and its approximation made on a single example, is given by the calculation of the average of the gradients over a small subset of examples, which are fewer in number than the entire training set.
This type of algorithm is typically the algorithm of choice when training neural networks, because it combines the best properties of deterministic and stochastic methods. Known as minibatch gradient descent, however, the term stochastic gradient descent is now commonly employed also when minibatches are used.
In a minibatch approach, for every iteration k, a minibatch of examples,

$$S_k = \{(x^{(1)}, y^{(1)}), ..., (x^{(M')}, y^{(M')})\} \subset \mathcal{S} \tag{2.24}$$

with $|S_k| = M'$, is drawn uniformly from the training set $\mathcal{S}$. These selections are done without repetitions, until the training set is exhausted, i.e. until an epoch is complete. Therefore, at each iteration, gradient estimation is done on the examples of the minibatch S_k,

$$\nabla_\theta \mathcal{J}_{M'}(\theta_k) = \frac{1}{M'} \sum_{i \in S_k} \nabla_\theta L(h(x^{(i)}, \theta_k), y^{(i)}) \tag{2.25}$$

and the algorithm uses this estimated gradient to take the next step, as reported in Algorithm 4.

This approach, proposed by Robbins and Monro [113], allows some degree of parallelization that can be exploited by the modern parallel computing architectures, and by state-of-the-art deep learning libraries. These are optimized to compute the gradient on minibatches, making

this calculation very efficient.
In addition, the minibatches contain a relatively small number $M' < M$ of examples, up to a few hundred, and remain fixed even as the training set grows, making it possible to work with datasets of thousands of elements. Still, this relative small number of examples is able to reduce the variance of stochastic method, which can lead to more stable convergence. Common minibatch sizes range between 50 and 256, but can vary for different applications.

Algorithm 4 Minibatch Gradient Descent update.

```
Input: Training set $\mathcal{S} = \{(x^{(i)}, y^{(i)}) \mid x^{(i)} \in X,\ y^{(i)} \in Y,\ i = 1, \dots, M\}$
  Learning rate schedule $\gamma_1, \gamma_2, \dots$
  $\theta_0$ initial set of parameters
  $M' < M$, size of the minibatches

  for $k = 0, 1, \dots$ do
    Sample a minibatch $S_k = \{(x^{(1)}, y^{(1)}), \dots, (x^{(M')}, y^{(M')})\}$ from the training set.
    Compute gradient estimate: $g_k = +\frac{1}{M'} \nabla_\theta \sum_{i \in S_k} L(h(x^{(i)}, \theta_k), y^{(i)})$
    Apply update: $\theta_{k+1} = \theta_k - \gamma_k\, g_k$
  end for
```

Minibatch Properties

Minibatch dimensions are chosen following some practical guidance:

- larger batches give a more accurate gradient estimation, but the goodness of the estimate does not grow linearly with the dimensions [56].
- Multicore architectures are not best exploited by selecting very small minibatches. This encourages to use some absolute minimum batch size, under which the processing time of the minibatch is not reduced.
- If all the examples in the batch are to be processed in parallel, as often happens, the amount of memory required grows with the size of the batch. In many applications, this is the determining factor in the choice of the size of the minibatch.
- Some types of hardware achieve a better execution time with specific-sized arrays. In particular, GPUs have better execution times working with minibatches of size a power of 2. Usually this power is chosen between 32 and 256.
- The generalization error is often optimal for batches of size 1, i.e. using the online algorithm. However, in this case the training may require a very small learning rate to ensure the stability of the algorithm, since the gradient estimates have a greater variance. Therefore, the algorithm requires a greater number of steps, both because of the reduced

learning rate, and because more steps are needed to explore the entire training set. This can greatly increase the total execution time.

Another important aspect is that the choice of the minibatch must be random. To have an unbiased estimate of the gradient we require that the examples of the minibatch on which it is calculated are independent. Furthermore, we desire for the estimates of subsequent gradients to be independent from each other, so that independence between the various minibatches is also required.
In cases where the order of the data in the dataset has a meaning, it is therefore necessary to shuffle the examples before selecting the minibatches, otherwise the effectiveness of the algorithm can be greatly reduced. Most implementations shuffle the dataset initially and then pass through it multiple times, i.e. for more epochs. On the first epoch, each minibatch is used to calculate an estimate of the generalization error that does not contain bias, since each example is a sample from the distribution that generates the data. From the second step onward, the estimate takes a bias because it is formed using values that have been used already. However, the subsequent passages through the training set provide quite a good result, due to the reduction of training error, that compensates for the increase in the gap between training error and generalization error.

Learning Rate Choice

As previously said, in non-convex optimization, the gradient noise can be beneficial to escape poor local minima. Despite this, noise can prevent full convergence to the minimum, as the algorithm will keep overshooting.
In [20, 106] it is shown that the variance of the fluctuations around a local minimum is proportional to the learning rate γ. This means that a good choice of γ is fundamental for the convergence of stochastic algorithms. If we set the learning rate too small, then the parameters update will be very slow and it will take a very long time to achieve an acceptable value for the loss function. If we set it too large, then the parameters will move all over the function and may never achieve acceptable loss at all. In particular, in order to reduce the fluctuations, it has been proposed to use a decreasing learning rate.

In the case of a strongly convex functions, it is often stated that to ensure the convergence to the global minimum, we should decay the learning rate, such that [113]:

$$\sum_{k=1}^{\infty} \gamma_k = \infty \tag{2.26}$$

and

$$\sum_{k=1}^{\infty} \gamma_k^2 < \infty, \tag{2.27}$$

where γ_k denotes the learning rate at the k^{th} gradient update. Intuitively, first condition (2.26) is necessary in order for θ to reach a basin of attraction of a minimum, while the second condition (2.27) ensures that the learning rate decays sufficiently quickly for the convergence to the minimum, rather than bouncing around it due to gradient noise [141].

Again, decaying learning rates are also empirically successful in non-convex optimization. The initial noisy optimization phase allows us to explore a larger fraction of the parameters space without becoming trapped in local minima. Once a promising region of parameters space is found, we want to reduce the noise to fine-tune the parameters.

In practice, the learning rate may be chosen by trial and error, by monitoring the progress of the objective function along the epochs. Also, a common type of decay used is step decay, which reduces the learning rate by some factor $q < 1$ every t epochs as

$$\gamma_k = \gamma_0 \cdot q^{k_{drop}} \tag{2.28}$$

where $k_{drop} = \lfloor \frac{epoch}{t} \rfloor$. The choice of t depends heavily on the type of problem and the model. Again, one heuristic approach used in practice is to watch the validation error while training with a fixed learning rate, and reduce the learning rate by a constant whenever the validation error stops improving.

Initial Parameters Choice

Learning algorithms are iterative algorithms, which require the choice of initial parameters. Especially in the training of neural networks, where the objective functions are complex, the initial point in the parameters space can have significant effect on the learning process. It can determine the convergence of the algorithm, the convergence speed and the type of point to which the algorithm converges, with high or low cost value. For example, if in a neural network sigmoid activation functions are used, and the weights are initialized large, then during the process the sigmoids saturate resulting in very small gradients, and the learning process can be very slow. Because of this, it is essential to find a good initial point, and many research are made in this direction [56].

Modern strategies for parameter initialization are simple and heuristic. Definitely, initial parameters must break the symmetry between the different units of the neural network. This means that if two hidden units with the same activation function are connected to the same inputs, then they must have different initial parameters, in order to not always be updated in the same way. This encourages a random choice of initial parameters. Typically, the weights are initialized to values that are randomly drawn from a Gaussian or a uniform distribution. Choosing one distribution or the other does not seem a key issue, but no exhaustive studies on the subject have been done.
What counts instead, both for the result of the optimization, and for the possibility of generalizing, is the choice of the scale of the initial distribution. Larger initial weights have a greater symmetry breaking effect, which helps to avoid redundant units. They may, however, lead to exploding values in the forward and backward propagation steps. Furthermore, large values of the weights can cause the saturation of certain types of activation functions, like sigmoid

activation functions, leading to vanishing gradient effects, as in the previous example.
All these observations influence the choice of the initial weight scale.

One heuristic approach proposes to initialize the weights of a layer with d_{in} input units and d_{out} output units in the weight tensor, by sampling each weight from the uniform distribution

$$W_{i,j} \sim U\left(-\frac{1}{\sqrt{d_{in}}}, \frac{1}{\sqrt{d_{in}}}\right). \tag{2.29}$$

Conversely, in [55] it is proposed a normalized version that draws samples from

$$W_{i,j} \sim U\left(-\frac{6}{\sqrt{d_{in}+d_{out}}}, \frac{6}{\sqrt{d_{in}+d_{out}}}\right). \tag{2.30}$$

In practice, these formulas are used as default for weights initialization in most used Deep Learning libraries, like Keras, or Pytorch. The biases are initialized in the same way.

When computational resources allow it, it is usually a good idea to treat the initial scale of values as a hyperparameter, consequently setting it with a manual search or an appropriate algorithm. Likewise, it is possible to initialize the parameters using machine learning techniques. A possible strategy, discussed in [56], is to initialize a supervised learning model with the parameters learned from an unsupervised model, trained on the same inputs as the previous one. These initialization strategies can provide faster convergence and better generalization as they encode in the intial parameters of the model information on the distribution that generates the data.

Early Stopping Criterion

In practice, the training process of deep neural networks is prone to overfitting, i.e. the network tends to interpolate training data, without generalizing well.
By training large models with sufficient capacity to induce overfitting, it is often observed that the training error decreases constantly, while the error on the validation set decreases in the first iterations, until it reaches a point where it rises again, as shown in Figure 2.11. This point highlights the beginning of the overfitting phase of the training. This means that we get a model with better generalization capacity, when the parameters return the lowest error on the validation set.

Early stopping criterion is based on the idea to periodically evaluate, during the minimization process, the error that the network makes on the validation set. Every time the error on the validation set decreases, a copy of the model parameters θ is saved, so that when the training algorithm ends, the saved parameters are returned. The algorithm ends when no set of parameters makes an improvement in the error on the validation set, for a certain number of iterations. Early stopping procedure is reported in Algorithm 5.

The early stopping strategy is the most common form of regularization in Deep Learning, being at the same time simple and effective. In fact, it does not require changes in the training procedure, in the objective function, or in the choice of the admissible parameters, for example

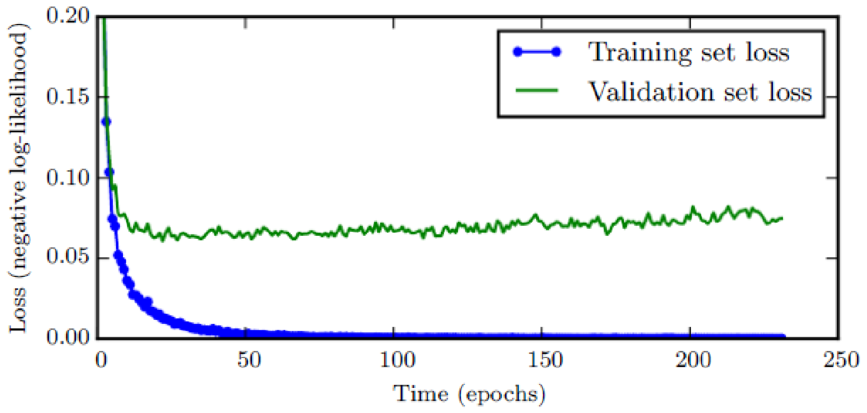


Figure 2.11: Behaviour, during training process, of a particular loss function on the training set and on the validation set, along the epochs. Example taken from [56].

by adding a penality term. In the case of early stopping, the actual capacity of the model is checked, choosing the number of steps that the algorithm can make and thus allowing the algorithm to stop when overfitting begins to occur. In particular, this means that the training can stop even if a minimum of empirical risk has not been reached, reducing the computational cost and the training time.
The only significant cost of early stopping is given by the periodic evaluation of the error on the validation set. However, this cost is generally negligible compared to the total training time. Moreover, it can be further reduced by using a small validation set compared to the training set, or by decreasing the frequency of these observations, at the cost of obtaining a worse estimate of the optimal parameters.

2.3.3 Backpropagation Algorithm

In the training process, the learning algorithm should adjust the network parameters in order to minimize the training error (2.16). Working with deep neural networks, and using gradient based methods as learning algorithms, the problem implies computing the gradient of the training error, which is an indirect function of the parameters in the hidden layers.

In this section we study the method used to perform this gradient computation in the specific case of MLP architectures, knowing that the theory behind is the same for different architectures.

Let's summarize a gradient based algorithm. The first step is to propagate the input through the network, as seen in equation (2.6) of Section 2.1.3, in order to calculate the training error. In this step, information flows forward through the network, in what is called the forward propagation (Algorithm 6).

Algorithm 5 Early Stopping meta-algorithm, from [56].

```
Input: Validation set $\{(x^{(i)}, y^{(i)}) \mid x^{(i)} \in X,\ y^{(i)} \in Y,\ i = 1, \dots, M_{\text{val}}\}$
  $n$, number of steps between evaluations.
  $p$, number of times to observe worsening validation set error before stopping the algorithm.
  $\theta_0$, initial set of parameters.

  $it = 0$
  $\theta_{it} = \theta_0$
  $j = 0$
  $v = +\infty$
  $\theta^* = \theta_{it}$
  $i^* = it$
  while $j < p$ do
    Update $\theta_{it}$ by running the training algorithm for $n$ steps.
    $it = it + n$
    Evaluate the validation error: $v' = \mathcal{J}_{M_{\text{val}}}(\theta_{it})$
    if $v' < v$ then
      $j = 0$
      $\theta^* = \theta_{it}$
      $i^* = it$
      $v = v'$
    else
      $j = j + 1$
    end if
  end while
  return $\theta^*$ with the best performance on the validation set, at iteration $i^*$.
```

The next step is to compute the gradient of the training error w.r.t the internal parameters. To calculate the gradient, the backpropagation algorithm [119], performs a backward step, allowing the information from the loss function to flow backwards through the network.
Finally, having the gradients, the weights and biases are updated using a stochastic gradient descent rule.

The term "backpropagation" is essentially linked to the technique used to calculate the derivatives of the objective function, called the chain rule of calculus. This is a derivation rule for composite functions, like $L(h(x, \theta), y)$.

Given $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$ and two functions $g : \mathbb{R}^n \to \mathbb{R}^m$, $f : \mathbb{R}^m \to \mathbb{R}$, if $y = g(x)$ and $z = f(y)$, then for the chain rule

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}, \qquad i = 1, ..., n \tag{2.31}$$

Algorithm 6 Forward propagation through a typical multilayer perceptron and computation of the loss function. For simplicity, only a single training example $(x, y) \in \mathcal{S}$ is used. Depending on the choice of the learning algorithm, practical applications should process multiple images at the same time, on which the final loss is averaged.

```
Input: $(x, y) \in \mathcal{S}$, training example
  $K$, network depth
  $\theta = \{(W^{(1)}, b^{(1)}), ..., (W^{(K)}, b^{(K)})\}$, network parameters (weight matrices and biases)

  $x^{(0)} = x$
  for $k = 1, \dots, K$ do
    $a^{(k)} = W^{(k)} x^{(k-1)} + b^{(k)}$
    $x^{(k)} = g^{(k)}(a^{(k)})$
  end for
  $h(x, \theta) = x^{(K)}$
  Compute the loss function: $L(h(x, \theta), y)$
```

or in vector notation,

$$\nabla_x z = J_g(x)^T \nabla_y z, \tag{2.32}$$

where $J_g(x)$ is the $m \times n$ Jacobian matrix of all the first-order partial derivatives of g, defined such that $J_{i,j} = \frac{\partial y_i}{\partial x_j}$.

The backpropagation algorithm consists of performing such Jacobian-gradient products for each operation in the computational graph associated to the network. In particular, the algorithm computes the chain rule, with a specific order of operations that is highly efficient.

Remark 2.2. Usually backpropagation algorithm is applied to tensors of arbitrary dimension, not merely to vectors, but conceptually this is the same. The only difference is how the numbers are arranged in a grid to form a tensor. We could imagine flattening each tensor into a vector before running backpropagation, computing a vector-valued gradient, and then reshaping the gradient back into a tensor.

Now, let us consider a multilayer perceptron with K layers. For simplicity, a single training example $(x, y) \in \mathcal{S}$ is used at a time. Again, we might need to work with multiple examples at the same time, depending on the learning algorithm we choose.
Keeping in mind the equations used in the forward step, with which the loss function is evaluated,

$$a^{(k)} = W^{(k)} x^{(k-1)} + b^{(k)}, \qquad x^{(k)} = g^{(k)}(a^{(k)}) \qquad \text{for } k = 1, ..., K \tag{2.33}$$

and saving both the resulting loss $L(h(x, \theta), y)$ and the layers outputs $x^{(k)}$, $k = 1, ..., K$, we can compute the gradient $\nabla_\theta L(h(x, \theta), y)$, by applying the chain rule for functions composition. First we are able to compute the gradient of the loss with respect to the output of the network, $\nabla_{x^{(K)}} L(h(x, \theta), y)$, and from this, the gradient related to a particular hidden layer is obtained by propagating, backwards along the network, the derivatives related to the successive layers, starting from the last layer. In particular, the gradients on weights and biases of the k^{th} layer are given by the formulas

$$\nabla_{W^{(k)}} L(h(x,\theta),y) = \nabla_{x^{(k)}} L(h(x,\theta),y) \cdot g'^{(k)}(a^{(k)}) \cdot x^{(k-1)T}, \tag{2.34}$$

and

$$\nabla_{b^{(k)}} L(h(x,\theta),y) = \nabla_{x^{(k)}} L(h(x,\theta),y) \cdot g'^{(k)}(a^{(k)}) \cdot 1. \tag{2.35}$$

In Algorithm 7 is reported the backpropagation procedure now described.

Algorithm 7 Backpropagation algorithm for calculating the gradients of the loss function $\nabla_\theta L(h(x,\theta),y)$ w.r.t. the MLP parameters, on a single training example $(x,y) \in \mathcal{S}$.

```
Input: $(x,y) \in \mathcal{S}$, training example
  $K$, network depth
  $\theta = \{(W^{(1)}, b^{(1)}), ..., (W^{(K)}, b^{(K)})\}$, network parameters (weight matrices and biases)
  $L = L(x^{(K)}, y)$ loss function, with $x^{(K)} = h(x,\theta)$, from the forward step

  Compute the gradient on the network output: $s = \nabla_{x^{(K)}} L$
  for $k = K, K-1, \ldots, 1$ do
    $\nabla_{a^{(k)}} L = s \cdot g'^{(k)}(a^{(k)})$
    Compute gradients on weights and biases:
    $\nabla_{W^{(k)}} L = \nabla_{a^{(k)}} L \cdot x^{(k-1)T}$
    $\nabla_{b^{(k)}} L = \nabla_{a^{(k)}} L \cdot 1$
    Propagate the gradient on the previous layer:
    $\nabla_{x^{(k-1)}} L = W^{(k)T} \cdot \nabla_{a^{(k)}} L$
    $s = \nabla_{x^{(k-1)}} L$
  end for
```

We chose to present Algorithms 6 and 7, because they are simple and straightforward to understand. However, they are specialized for one specific network architecture, the multilayer perceptron.

The ideas behind general backpropagation algorithm are the same. To compute the gradient of z w.r.t. one of its ancestors x into a generic network, we can compute the gradient w.r.t each parent of z, by multiplying the current gradient by the Jacobian of the operation that produced z. We continue multiplying by Jacobians, going backward through the network until we reach x. For any node that may be reached by going backward from z through two or more paths, we simply sum the gradients arriving from the different paths.

When initializing the network architecture, modern software implementations associate a variable to each node, which is a tensor structure. Also, for every operation between two nodes, software libraries, such as PyTorch, provide a structure to compute both the operation itself, `op.forward`, and the associated backward operation, `op.backward`, which compute the Jacobian-vector product as described by equation (2.32). This is how the backpropagation algorithm is able to achieve great generality. Each operation is responsible for knowing how to back-propagate through the edges in the graph that it participates in.
Then, the backpropagation algorithm itself does not need to know any differentiation rules. It only needs to call each operation's `op.backward` with the right arguments. Formally, `op.backward(inputs, x, g_out)` must return

$$\sum_i (\nabla_{\mathtt{X}} \mathtt{op.f(inputs)}_i) \cdot \mathtt{g\_out}_i$$

which is just an implementation of the chain rule. Here, `inputs` is a list of inputs that are supplied to the operation, `op.f` is the mathematical function that the operation implements, `x` is the input whose gradient we wish to compute, and `g_out` is the gradient on the output of the operation.

Remark 2.3. Users who need to add their own operation to an existing library must usually derive the `op.backward` method for any new operations manually. For example, this was necessary for the specific network proposed in the experimental part of this thesis.

2.3.4 Other Optimization Algorithms

While stochastic gradient descent remains a very popular optimization strategy, many other methods have been developed in last years. We now outline some of these learning algorithms, which are widely used by the Deep Learning community.

Momentum Methods

Although the SGD algorithm is a very popular optimization strategy, it can be slow.

The momentum method, introduced by Polyak [110], is a technique able to accelerate learning when there are small but constant, or noisy gradients, because gradients of previous iterations contribute to the calculation of the current direction. To be more specific, the SGD algorithm with momentum makes steps along directions that are obtained by accumulating previous gradients with an exponential decay.

Formally, the momentum algorithm introduces a momentum term v, which represents the velocity vector, i.e. it indicates the direction and the speed with which the parameters move in parameters space.

Starting from an initial velocity v, at each iteration, the update rule is given by

$$v = \alpha v - \frac{\gamma}{M'} \sum_{i=1}^{M'} \nabla_\theta L\left(h(x^{(i)}, \theta), y^{(i)}\right) \tag{2.36}$$

$$\theta = \theta + v, \tag{2.37}$$

where $\alpha \in [0, 1)$ is the decaying hyperparameter, which determines how quickly the contributions of previous gradients will decay, or how much we trust the accumulated previous gradients. The more α is larger than γ, the more previous gradients influence the current direction. The SGD method with momentum is presented in Algorithm 8.

Algorithm 8 Stochastic Gradient Descent (SGD) with momentum.

```
Input: Training set $\mathcal{S} = \{(x^{(i)}, y^{(i)}) \mid x^{(i)} \in X,\ y^{(i)} \in Y,\ i = 1, \dots, M\}$
  $M' < M$, size of the minibatches
  Learning rate schedule $\gamma_1, \gamma_2, \dots$
  $\theta_0$ initial set of parameters
  $\alpha$ momentum parameter, $v$ initial velocity

  for $k = 0, 1, \dots$ do
    Sample a minibatch $S_k = \{(x^{(1)}, y^{(1)}), \dots, (x^{(M')}, y^{(M')})\}$ from the training set.
    Compute gradient estimate: $g_k = \frac{1}{M'} \nabla_\theta \sum_{i \in S_k} L(h(x^{(i)}, \theta_k), y^{(i)})$
    Compute velocity update: $v = \alpha v - \gamma_k g_k$
    Apply update: $\theta_{k+1} = \theta_k + v$
  end for
```

In this algorithm, the sequence of previous gradients also determines the step in the current direction. A greater step is taken when many successive gradients point in the same direction. In this case, unlike the classical SGD algorithm, θ tends to proceed in the same direction, gaining convergence speed and avoiding strong oscillations.
In particular, this is useful to maintain stability when having noisy gradients, or when approaching areas where the surface curves much more steeply in one dimension than in another. In this scenario, SGD oscillates across the slopes of the objective function while only making small progress along the bottom. Instead, SGD with momentum is able to avoid strong oscillations, as shown in Figure 2.12.

In [129], a variant of the momentum has been introduced, called Nesterov momentum, which takes inspiration from the Nesterov Accelerated Gradient method [107, 108].
The difference between momentum method and Nesterov momentum is in the gradient computation phase. In momentum method, the gradient is computed using the current parameters θ, whereas in Nesterov momentum, first we apply the velocity v to the parameters θ to compute intermediate parameters $\tilde{\theta} = \theta + \alpha v$. We then compute the gradient using the intermediate

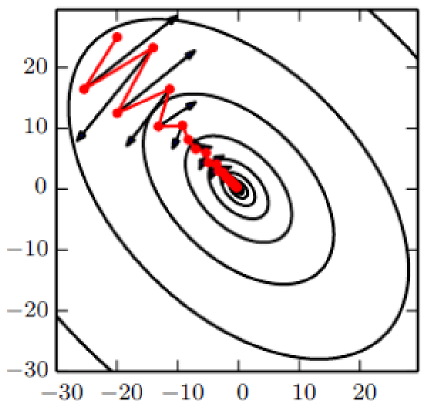


Figure 2.12: Path followed by the SGD algorithm with momentum (in red), in minimizing a quadratic loss function that presents an ill-conditioned Hessian matrix. At each step the black arrows indicate the direction that a gradient descent algorithm would take. Figure from [56].

parameters as in the following equation,

$$v = \alpha v - \frac{\gamma}{M'} \sum_{i=1}^{M'} \nabla_\theta L(h(x^{(i)}, \theta + \alpha v), y^{(i)}). \tag{2.38}$$

Through the intermediate parameters, the algorithm can predict where the point is heading in the parameters space, so as to slow down before climbs. In particular, if a momentum term αv is used to update the parameters θ, then $\theta + \alpha v$ gives an approximation of their future position in the parameters space. Thus, Nesterov momentum is able to look ahead, by calculating the gradient not with respect to the current parameters, but with respect to an approximation of their future value, [117]. Nesterov momentum is presented in Algorithm 9.

Adaptive Gradient Algorithm (AdaGrad)

The high-dimensional non-convex nature of neural networks can lead the objective function to be sensitive to some directions in the parameters space with strong curvature, and less sensitive to other directions with minor curvature. Therefore, the learning rate can be too small in some dimensions and too large in others.
A series of minibatch methods have been introduced, which automatically adapt the learning rate for individual parameters, during learning. In this way, the various parameters are updated to make longer or shorter steps depending on their importance.

Algorithm 9 Stochastic gradient descent (SGD) with Nesterov momentum.

```
Input: Training set $\mathcal{S} = \{(x^{(i)}, y^{(i)}) \mid x^{(i)} \in X,\ y^{(i)} \in Y,\ i = 1, \dots, M\}$
  $M' < M$, size of the minibatches
  Learning rate schedule $\gamma_1, \gamma_2, \dots$
  $\theta_0$ initial set of parameters
  $\alpha$ momentum parameter, $v$ initial velocity

  for $k = 0, 1, \dots$ do
    Sample a minibatch $S_k = \{(x^{(1)}, y^{(1)}), \dots, (x^{(M')}, y^{(M')})\}$ from the training set
    Apply intermediate update: $\tilde{\theta}_k = \theta_k + \alpha v$
    Compute gradient (at intemediate point): $g_k = \frac{1}{M'} \nabla_{\tilde{\theta}} \sum_i L(h(x^{(i)}; \tilde{\theta}_k), y^{(i)})$
    Compute velocity update: $v = \alpha v - \gamma_k g_k$
    Apply update: $\theta_{k+1} = \theta_k + v$
  end for
```

The Adaptive Gradient algorithm, AdaGrad [47], is one of these methods. At each iteration, AdaGrad modifies the general learning rate γ_k for every parameter θ_{k_i} , by using the gradients of previous steps, calculated with respect to θ_i. In particular, AdaGrad scales the step size of a factor, which is inversely proportional to the square root of the sum of the squared previous gradients [47],

$$\theta_{k+1} = \theta_k - \frac{\gamma_k}{\delta I + \sqrt{\operatorname{diag}(G_k)}} \odot g_k. \tag{2.39}$$

Here we denote the Hadamard product with the symbol $\odot$, δ is a smoothing term that avoids division by zero, and G_k is the outer product matrix

$$G_k = \sum_t^k g_t g_t^T, \tag{2.40}$$

whose diagonal elements are the sum of the squared gradients w.r.t. θ_i up to iteration k. In this way, parameters with greater partial derivatives have faster decreasing learning rates, while parameters with smaller partial derivatives have learning rates which decrease slowly.
A slightly easier-to-understand implementation is reported in Algorithm 10.

AdaGrad's main weakness is its accumulation of the squared gradients in the denominator: since every added term is positive, the accumulated sum keeps growing during training. This can lead to an excessive and premature decrease of the learning rate. In this way the algorithm may not reach a minimum.

RMSProp is an unpublished, adaptive learning rate method proposed by Hinton in a lecture of his Coursera Class, which seeks to reduce the strong decrease in the learning rate obtained with AdaGrad. To do this it changes the accumulation of the squares of the previous gradients

Algorithm 10 AdaGrad algorithm, from [56]

```
Input: Training set $\mathcal{S} = \{(x^{(i)}, y^{(i)}) \mid x^{(i)} \in X,\ y^{(i)} \in Y,\ i = 1, \dots, M\}$
  $M' < M$, size of the minibatches
  $\gamma$, global learning rate
  $\theta_0$, initial set of parameters
  Small constant $\delta \sim 10^{-7}$, for numerical stability

  Initialize gradient accumulation variable $G_{diag} = 0$
  for $k = 0, 1, \dots$ do
    Sample a minibatch $S_k = \{(x^{(1)}, y^{(1)}), \dots, (x^{(M')}, y^{(M')})\}$ from the training set
    Compute gradient estimate: $g_k = \frac{1}{M'} \nabla_\theta \sum_{i \in S_k} L(h(x^{(i)}, \theta_k), y^{(i)})$
    Accumulate squared gradients: $G_{diag} = G_{diag} + g_k \odot g_k$
    Compute update: $\Delta\theta_k = -\frac{\gamma}{\delta I + \sqrt{G_{diag}}} \odot g_k$    (operations applied element-wise)
    Apply update: $\theta_{k+1} = \theta_k + \Delta\theta_k$
  end for
```

into a moving average, subject to an exponential decay. In this way older contributions are discarded and the algorithm can converge quickly after arriving near a minimum. The RMSProp method is presented in Algorithm 11.
Empirical results show that the RMSProp algorithm is an efficient learning algorithm, and one of the most widely used in neural networks training, [56].

Adaptive Moment Estimation Algorithm (Adam)

Adam algorithm, from [83], is a variant of the RMSProp algorithm with momentum, described in Algorithm 12.
As we can see in the algorithm, in addition to accumulating an exponential moving average v_k of the squared previous gradients, like RMSProp does, an average of the gradients m_k with exponential decay is also accumulated, similar to the momentum algorithm,

$$m_k = \rho_1 m_{k-1} + (1 - \rho_1) g_k \tag{2.41}$$

$$v_k = \rho_2 v_{k-1} + (1 - \rho_2) g_k \odot g_k \tag{2.42}$$

where the hyper-parameters $\rho_1, \rho_2 \in [0, 1)$ control the exponential decay rates, and where m_k and v_k are respectively the estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients, with $m_0 = 0$, $v_0 = 0$.
Now, the authors observed that a bias correction is necessary, because these moving averages are initialized as null vectors, and this makes them biased towards zero, especially in the early

Algorithm 11 RMSProp algorithm, from [56].

```
Input: Training set $\mathcal{S} = \{(x^{(i)}, y^{(i)}) \mid x^{(i)} \in X,\ y^{(i)} \in Y,\ i = 1, \dots, M\}$
  $M' < M$, size of the minibatches
  $\gamma$, global learning rate (suggested default: 0.001)
  $\theta_0$, initial set of parameters
  $\rho$, decay rate (suggested default: 0.9)
  Small constant $\delta \sim 10^{-6}$, for numerical stability

  Initialize accumulation variable $G_{diag} = 0$
  for $k = 0, 1, \dots$ do
    Sample a minibatch $S_k = \{(x^{(1)}, y^{(1)}), \dots, (x^{(M')}, y^{(M')})\}$ from the training set
    Compute gradient estimate: $g_k = \frac{1}{M'} \nabla_\theta \sum_{i \in S_k} L(h(x^{(i)}, \theta_k), y^{(i)})$
    Accumulate squared gradients: $G_{diag} = \rho\, G_{diag} + (1 - \rho) g_k \odot g_k$
    Compute update:$\Delta\theta_k = -\frac{\gamma}{\delta I + \sqrt{G_{diag}}} \odot g_k$    (operations applied element-wise)
    Apply update: $\theta_{k+1} = \theta_k + \Delta\theta_k$
  end for
```

stages of training. Because of this, Adam applies bias corrections to first and second moment estimates,

$$\hat{m_k} = \frac{m_k}{1 - \rho_1^k} \tag{2.43}$$

$$\hat{v_k} = \frac{v_k}{1 - \rho_2^k} \tag{2.44}$$

and from these estimates, the method derives individual learning rates for the different parameters, following the update rule

$$\theta_k = \theta_{k-1} - \gamma \frac{\hat{m_k}}{\sqrt{\hat{v_k}} + \delta}. \tag{2.45}$$

Let us see from [83], how the unbiased term for the second moment estimate (2.44) is derived, knowing that the derivation for the first moment estimate is similar.
At iteration k, given the gradient of the loss function on a selected minibatch g_k, its second raw moment is estimated using (2.42), with decay rate ρ_2. In particular, initializing $v_0 = 0$, this update can be written as a function of the gradients at the previous iterations, such that

$$v_k = (1 - \rho_2) \sum_{i=1}^{k} \rho_2^{k-i} \cdot g_i^2. \tag{2.46}$$

Algorithm 12 Adam algorithm, from [83].

```
Input: Training set $\mathcal{S} = \{(x^{(i)}, y^{(i)}) \mid x^{(i)} \in X,\ y^{(i)} \in Y,\ i = 1, \dots, M\}$
  $M' < M$, size of the minibatches
  $\gamma$, global learning rate (suggested default: 0.001)
  $\theta_0$, initial set of parameters
  $\rho_1, \rho_2 \in [0, 1)$, exponential decay rates for moments estimates (suggested defaults: 0.9 and 0.999 respectively)
  $\delta \sim 10^{-8}$, for numerical stability

  Initialize 1$^{st}$ and 2$^{nd}$ moment variables $m_0 = 0$, $v_0 = 0$
  for $k = 0, 1, \dots$ do
    Sample a minibatch $S_k = \{(x^{(1)}, y^{(1)}), \dots, (x^{(M')}, y^{(M')})\}$ from the training set
    Compute gradient estimate: $g_k = \frac{1}{M'} \nabla_\theta \sum_{i \in S_k} L(h(x^{(i)}, \theta_{k-1}), y^{(i)})$
    Update biased first moment estimate: $m_k = \rho_1 m_{k-1} + (1 - \rho_1) g_k$
    Update biased second moment estimates: $v_k = \rho_2 v_{k-1} + (1 - \rho_2) g_k \odot g_k$
    Compute bias-corrected moments:
    $\hat{m_k} = \frac{m_k}{1-\rho_1^k}$
    $\hat{v_k} = \frac{v_k}{1-\rho_2^k}$
    Compute update: $\Delta\theta_k = -\gamma \frac{\hat{m}_k}{\sqrt{\hat{v}_k}+\delta}$    (operations applied element-wise)
    Apply update: $\theta_k = \theta_{k-1} + \Delta\theta_k$
  end for
```

where we use g_i^2 to indicate the element-wise square $g_i \odot g_i$.
From the bias definition in equation (1.4), we want to know how the expected value of the exponential moving average, $\mathbb{E}[v_k]$, is related to the true second moment $\mathbb{E}[g_k^2]$, so to correct the discrepancy between them. Taking expectations of both sides of (2.46), it results

$$\mathbb{E}[v_k] = \mathbb{E}\left[(1-\rho_2)\sum_{i=1}^{k}\rho_2^{k-i}\cdot g_i^2\right] \tag{2.47}$$

$$= \mathbb{E}[g_k^2]\cdot(1-\rho_2)\sum_{i=1}^{k}\rho_2^{k-i} + \zeta \tag{2.48}$$

$$= \mathbb{E}[g_k^2]\cdot(1-\rho_2^k) + \zeta \tag{2.49}$$

where $\zeta = 0$, if $\mathbb{E}[g_i^2]$ is stationary; otherwise ζ can be kept small since the exponential decay rate ρ_2 is chosen such that the exponential moving average assigns small weights to gradients too far in the past. In Algorithm 12 we therefore divide by $(1-\rho_2^t)$ to correct the initialization bias, derived from initializing the running average with zeros.

Chapter 3

Inverse Problems and Imaging

The topics addressed in this chapter use some notions of convex analysis and functional analysis. A brief introduction to these concepts can be found by the reader in Appendix A.

3.1 Inverse Problems and Ill-Posedness

Inverse problems theory has existed for many years in many branches of physics, engineering and mathematics. Especially in recent decades, it has been extensively developed thanks to applications of interest, the availability of more powerful computers and reliable numerical methods.

In 1976 Professor J.B. Keller gave a definition of inverse problem, which showed how the concept has a certain degree of ambiguity. From [81]:

> We call two problems inverse of one another if the formulation of each involves all or part of the solution of the other. Often, for historical reasons, one of the two problems has been studied extensively for some time, while the other has never been studied and is not so well understood. In such cases, the former is called the direct problem, while the latter is the inverse problem.

As stated, one problem can be obtained from the other by exchanging the roles of the data and the unknowns, so that it may seem arbitrary to decide which is the direct problem and which is the inverse. What is usually done is to call direct problem the simpler one or the one which was studied earlier.

The choice is less arbitrary dealing with problems of applied mathematics and physics. In this case, there is a fairly natural distinction between direct and inverse problems. A direct problem is a problem which consists of calculating the consequences of certain causes. Then the corresponding inverse problem is associated with the inversion of the cause–effect sequence, i.e., it regards the study of the causes for a specific effect [9]. For example, knowing the current state of a physical system and the physical laws that rule it, a direct problem is to predict the

future behavior of the system. Conversely, if we want to identify some physical parameters from observations of the system evolution we are dealing with an inverse problem. In this way, inverse problems use the results of actual observations to infer the model or the values of the parameters characterizing the system under investigation. Observations can be indirect measurements; for instance, in medical tomography, one wishes to imagine the structures within the body from X–ray measurements that have passed through the body.
Furthermore, external factors affecting empirical measurement cause it to return approximate values, so that observations are usually noisy data. This often leads to an instability phenomenon called ill–posedness, which implies that even small perturbations in the data may give rise to significant errors in the sought estimates. The basic concept of a well–posed problem was introduced by the French mathematician Hadamard in [61].

Definition 3.1. *A problem is well–posed in the sense of Hadamard if it satisfies the following conditions:*

1. *for all admissible data, a solution exists;*
2. *for all admissible data, the solution is unique;*
3. *the solution depends continuously on the data.*

If at least one of the three properties is not satisfied, the problem is said to be ill–posed.

3.1.1 Linear Inverse Problems

In our work we consider a particular linear inverse problem. In this case, we assume that X and Y are Hilbert spaces, and we denote with $\mathcal{L}(X,Y)$ the space of linear and continuous operators between X and Y. A linear inverse problem can be formulated according to a general scheme. First, the direct problem is defined, and its solution introduces a linear continuous operator A, whose domain is in the space X of the ground-truths, while its range is in the space Y of the functions representing the measured data. Then, the inverse problem consists in determining $f \in X$ from the knowledge of $g \in Y$ when g and f are related by the relation

$$g = Af. \tag{3.1}$$

In the case of linear inverse problems, i.e., when $A \in L(X,Y)$, the well–posedness conditions can be synthetically written as:

1. $D(A^{-1}) = Y$;
2. $N(A) = 0$;
3. A^{-1} is continuous.

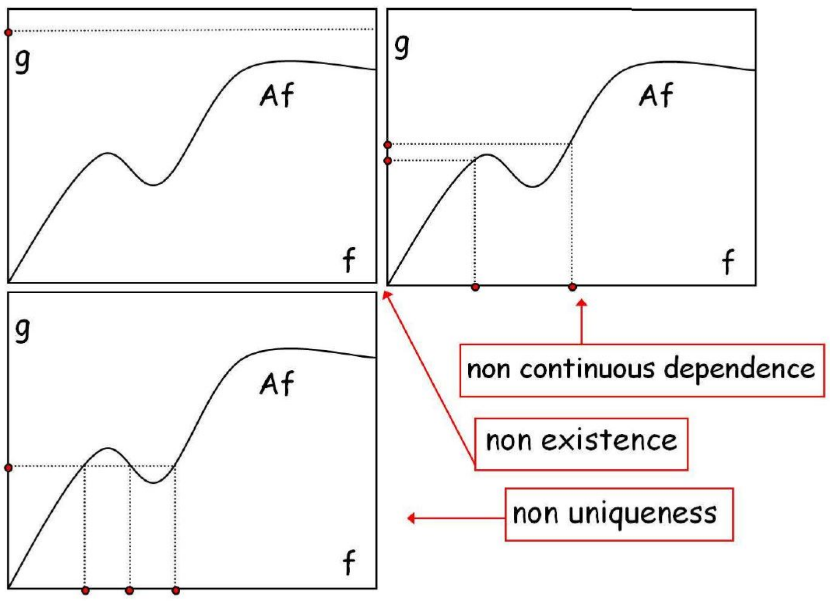


Figure 3.1: Examples of the violation of the well–posedness conditions.

where $D(A^{-1})$ denotes the domain of A^{-1} in Y.

Let us consider a linear ill-posed problem. Since the observations are obtained by measurements, we work with approximate values. If we have a noisy data, which is not in the range of the operator A, then the solution of the inverse problem does not exist. In this case, the existence condition can usually be repaired by relaxing the notion of solution.
Conversely, if the condition of uniqueness is not fulfilled, a criterion is required to decide which one among all the solutions is of interest, typically through the assumption of additional information about the solution itself. In particular, when working with real data, non-uniqueness is usually introduced by the need to discretize the problem.
Finally, a solution which does not depend continuously on the data can cause serious numerical issues. In particular, if the continuity condition is not satisfied, the numerical solution becomes unstable, meaning that arbitrarily small perturbations on data can produce arbitrarily large perturbations on the solution. A stability requirement is essential for meaningful problems in applied sciences, claiming that a solution which varies considerably for a small variation of the data is not really a solution in a physical sense.
See Figure 3.1 for different examples of ill–posed inverse problems.

It is to be said that well–posedness is not a sufficient condition for the stability of the problem. In fact, let us assume to have a well–posed linear inverse problem, so that A^{-1} is well–defined and continuous. Then, if in equation (3.1) δg is a small variation of the datum and δf is the corresponding variation on the solution, the continuity of A^{-1} allows to apply inequality (A.3), so that

$$||\delta f||_X \leq ||A^{-1}|| \; ||\delta g||_Y. \tag{3.2}$$

Likewise, the continuity of A implies

$$||f||_X \geq \frac{||g||_Y}{||A||} \tag{3.3}$$

so that

$$\frac{||\delta f||_X}{||f||_X} \leq ||A|| \; ||A^{-1}|| \frac{||\delta g||_Y}{||g||_Y}. \tag{3.4}$$

Definition 3.2. *The real positive number*

$$C(A) = ||A|| \; ||A^{-1}|| \tag{3.5}$$

is said "condition number" and provides an estimate of the instability of the problem. A problem with a low condition number is said to be well–conditioned, while a problem with a high condition number is said to be ill–conditioned.

From inequality (3.4) we see that the condition number can be used to measure how sensitive a solution is to changes or errors in the observed data. If the condition number $C(A)$ is much greater than 1, then a small relative variation on the data can produce very large oscillations on the solution. It follows that the presence of even a small error on the data of an ill–conditioned problem can make its solution extremely unstable.

In summary, whenever we solve a linear inverse problem on a computer, we always face difficulties similar to those above, because the associated computational problem is ill–conditioned, meaning that even if the solution exists and is unique, it may be completely corrupted by a small error on the data. We can partially repair these problems with the use of mathematical techniques known as regularization methods. The problem is regularized, such that the solution becomes more stable, i.e., less sensitive to perturbations, and this is done by including additional hypotheses about the solution. In fact, by supplying proper additional information, regularization methods can compute approximate solutions, which are the best compromise between accuracy and stability.
Therefore, to suppress the ill–posedness, designing the proper inversion model and developing proper regularization and optimization algorithms play a vital role [138]. Before moving on to these topics, let us introduce the specific linear inverse problem we faced in the experimental part of this thesis.

3.2 Image Reconstruction Problem

We focus our study on the problem of image reconstruction. This is a well–known example of ill–posed inverse problem, where the goal is to recover an estimate of the unknown object $\bar{x}$, of which we have a degraded image y [9, 62]. An accurate mathematical modeling is necessary to obtain a proper formulation of the problem.

3.2.1 Mathematical Modeling of Data Acquisition

An image can be described as a signal that carries information about a physical object which is not directly observable. This information is generally a degraded representation of the original object obtained through an imaging system composed of two parts: the image formation process, and the image recording process.

- Image formation is performed by a physical apparatus (composed by components such as sources, mirrors, lenses etc.), able to transform the radiation (photons, X–rays, microwaves, ultrasounds etc.) emitted by the object into a detectable radiation containing useful information on the spatial properties of the object. The degradation due to the image formation process is denoted by blurring and is a sort of bandlimiting of the object. For instance, in aerial photograph two typical sources of blurring are the relative motion between the camera and the ground, and the atmospheric turbulence [9].

- Image recording is performed by a detector, which provides measured values of the incoming radiation on an array y of disjoint pixels. The detector introduces a degradation in the recorded digital image, called noise, due to measurement or counting errors. This noise is a statistical process, so that, each component y_i of the recorded image vector can be seen as a realization of a random variable Y_i.

Unlike noise, blurring is due to a deterministic process and, in most cases, it can be described with a sufficiently accurate mathematical model. In particular, we want to model the map that transforms the spatial distribution of the object into the radiation captured by the detector. In the following, we only consider the case where this map is linear, thanks to the use of suitable physical approximations.

If we denote with $\bar{x} = \bar{x}(s)$ ($s = \{s_1, s_2\}$ in 2-D imaging) a function describing the unknown object in a suitable object space X, and with $\bar{y} = \bar{y}(s)$ the acquired noise–free image in the image space Y, then the optical image formation is frequently modeled by the following continuous linear model [9],

$$\bar{y}(s) = \int \mathcal{H}(s, s')\bar{x}(s')ds' \tag{3.6}$$

which is a Fredholm integral equation of the first kind [63], with kernel function $\mathcal{H}(s, s')$ called the Point Spread Function (PSF).

Equation (3.6) establishes a linear relationship between the two functions $\bar{x}$ and $\bar{y}$. If the object $\bar{x}$ and the PSF $\mathcal{H}$ are known, then we can compute the blurred image $\bar{y}$ by evaluating the integral using standard numerical quadrature; this is the direct problem. Equation (3.6) can be used, for instance, to model the diffraction of light from the source, as it propagates through the atmosphere. The same equation can also model distortion due to imperfections in optical devices, like telescopes [138]. In particular, the PSF $\mathcal{H}(s, s')$ characterizes the blurring effects that occur during image formation. The term "Point Spread Function" comes from the fact that $\mathcal{H}(\cdot, s')$ is the image of a point source located in s', which is spread over a number of

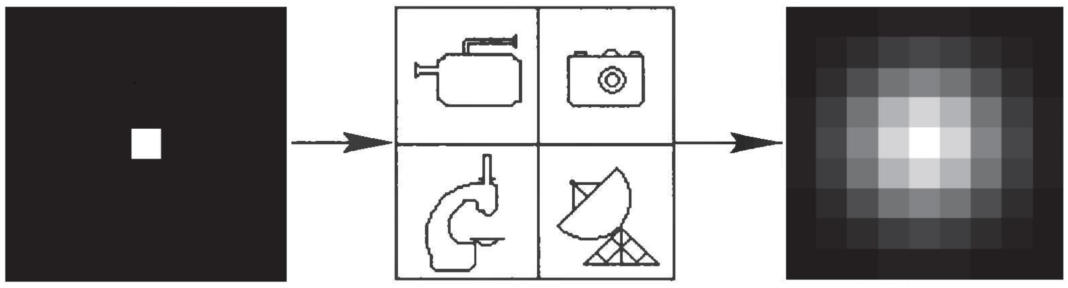


Figure 3.2: Schematic representation of the PSF.

pixels in the blurred image. Indeed, if the object is a point source given by $\bar{x}(s'') = \delta(s''-s')$ in equation (3.6), with integration variable denoted now by s'', and where $\delta(\cdot)$ indicates the Dirac delta distribution, then according to (3.6), one obtains $\bar{y}(s) = \mathcal{H}(s,s')$. A schematic representation of the PSF is given in Figure 3.2.

In several acquisition systems, the PSF is assumed to be space–invariant, i.e., invariant with respect to translations. In this case, the kernel $\mathcal{H}(s,s')$ depends only on the difference between s and s', i.e., $\mathcal{H}(s,s') = \mathcal{H}(s-s')$, and model (3.6) reduces to a convolution product

$$\bar{y}(s) = \int \mathcal{H}(s-s')\,\bar{x}(s')\,ds' = (\mathcal{H} * \bar{x})\,(s) \tag{3.7}$$

as defined in (2.13). This representation greatly simplifies computations. As observed in section 2.1.4, since the integral in (3.7) has a convolution form, given an object $\bar{x}$ and the PSF $\mathcal{H}$, the blurred image $\bar{y}$ can be computed using the convolution theorem [138],

$$\bar{y} = \mathcal{F}^{-1}\{\mathcal{F}\{\mathcal{H}\}\mathcal{F}\{\bar{x}\}\} \tag{3.8}$$

where $\mathcal{F}$ denotes the Fourier transform operator, so that $\mathcal{F}\{\bar{x}\}$ and $\mathcal{F}\{\mathcal{H}\}$ are the Fourier transforms of $\bar{x}$ and $\mathcal{H}$, respectively. Here the Fourier transform of a function f, defined on the set of square-integrable functions $L^2(\mathbb{R}^2)$, is given by

$$\mathcal{F}\{f\}(\omega) = \int_{\mathbb{R}^2} f(s)e^{-i2\pi s^T\omega}ds, \qquad \omega \in \mathbb{R}^2, \tag{3.9}$$

while the inverse continuous Fourier transform is given by

$$\mathcal{F}^{-1}\{g\}(s) = \int_{\mathbb{R}^2} g(\omega)e^{i2\pi s^T\omega}d\omega, \qquad s \in \mathbb{R}^2. \tag{3.10}$$

Furthermore, we have already seen in section 2.1.4 that, when images are treated as digital signals, a discrete version of model (3.7) is required. In this case, the object and the PSF are

treated as two vectors $\bar{x}, h \in \mathbb{R}^n$, and the convolution product can be seen as the matrix–vector product $H\bar{x}$, where $H \in \mathbb{R}^{m\times n}$ is the convolution matrix obtained by imposing some specific boundary conditions on the discretized PSF h [64].
Standard assumptions require H to have non–negative elements, and each row and column must contain at least one positive entry. In other words, we assume:

$$H_{i,j} \geq 0, \ \forall\ i,j; \quad \sum_{i=1}^{m} H_{i,j} > 0, \ \forall j; \quad \sum_{j=1}^{n} H_{i,j} > 0, \ \forall i.$$

Let us also remark that, if periodic boundary conditions are employed, i.e., if the two–dimensional PSF $h = \{h_{i,j}\}_{i=1,\dots,m}^{j=1,\dots,n}$ is such that

$$h_{m+1,j} = h_{1,j}, \quad h_{i,n+1} = h_{i,1}, \quad \forall\ i,j \tag{3.11}$$

then H is block circulant with circulant blocks and the matrix–vector products $H\bar{x}$ and $H^T\bar{x}$ can be efficiently computed by making use of the Discrete Fourier Transform (DFT) and its inverse (IDFT) [9, 64]. Indeed, by means of the convolution theorem, we have

$$\begin{aligned} H\bar{x} &= \mathrm{IDFT}\left(\mathrm{DFT}(h)\cdot \mathrm{DFT}(\bar{x})\right) \\ H^T\bar{x} &= \mathrm{IDFT}\left(\overline{\mathrm{DFT}(h)}\cdot \mathrm{DFT}(\bar{x})\right), \end{aligned}$$

where $\overline{\alpha}$ denotes the complex conjugate of $\alpha \in \mathbb{C}$. Hence, the above matrix–vector products may be performed with a $\mathcal{O}(mn\log(mn))$ complexity by means of the Fast Fourier Transform (FFT) algorithm implementation [64, 138]. This efficient computation is guaranteed also with other boundary conditions, such as zero or reflexive conditions, which imply the use of other discrete transforms apart from the DFT [64].

It is to be noted that the set of noise–free images, which is the range of the operator H (in matrix form), does not coincide with the image space Y. In fact, Y also contains the noisy images, corresponding to the detected images corrupted by the noise affecting the recording process. Taking into account the introduction of noise during detection, digital image acquisition can be modeled by

$$y = \mathcal{D}(H\bar{x}), \tag{3.12}$$

where $y \in \mathbb{R}^m$ is the blurred and noisy image (observed data), $\bar{x} \in \mathbb{R}^n$ is the sought image, $H \in \mathbb{R}^{m\times n}$ is the blurring operator, which is assumed linear, and $\mathcal{D}$ is the noise perturbation operator. The noise is in general a realization of a stochastic process, of which we may know statistical properties, such as mean value, variance, etc. These properties, when known, should be used in the treatment of the inverse problem. As a consequence of the noise introduction in model (3.12), each pixel y_i of the detected image can be seen as a realization of a random variable Y_i. By setting $Y = (Y_1, \dots, Y_m)$, the modeling of the system is then related to the probability density of the vector-valued random variable Y, corresponding to the recorded image y. This density depends on the object $\bar{x}$ and therefore we denote it as $p_Y(y;\bar{x})$. The following assumptions on Y_i and Y are usually accepted as reasonable ones [9, 12]:

- the random variables Y_i are statistically independent, so that

$$p_Y(y;\bar{x}) = \prod_{i=1}^{m} p_{Y_i}(y_i;\bar{x}); \tag{3.13}$$

- the expected value of Y_i is given by the i–th pixel of the noise–free image, hence

$$E(Y) = \int y \; p_Y(y;\bar{x}) \; dy = H\bar{x}. \tag{3.14}$$

In our study, we consider two classical examples of noise models, used in imaging to mimic the effect of different kind of processes that occur in nature: the additive white Gaussian noise and the Poisson noise.
In the case of additive white Gaussian noise, the noise is added to the blurred noise–free image, so that the detected y is given by

$$y = H\bar{x} + w. \tag{3.15}$$

Here each component w_i of the noise vector w is a realization of a random variable W_i having a Gaussian distribution with zero mean and standard deviation $\sigma > 0$. Then the noise w is a realization of a vector-valued random variable W, with statistically independent components, whose probability density is

$$p_W(w) = \frac{1}{(2\pi\sigma^2)^{m/2}} \exp\left(-\frac{1}{2\sigma^2}\|w\|^2\right). \tag{3.16}$$

Therefore, the statistical model for the detected image is given by

$$p_Y(y;\bar{x}) = \frac{1}{(2\pi\sigma^2)^{m/2}} \exp\left(-\frac{1}{2\sigma^2}\|y - H\bar{x}\|^2\right). \tag{3.17}$$

Poisson noise is used in general to describe the noise affecting counting processes. For example, light intensity is proportional to the number of photons hitting the measurement device. In this case, each Y_i is a Poisson random variable with expected value given by the blurred noise–free image $\bar{y}_i = (H\bar{x})_i$,

$$Y_i \sim \text{Poisson}\{(H\bar{x})_i\}.$$

By invoking the statistical independence of the random variables Y_i, we have that the probability density of Y is a distribution with support the set of non-negative integers, as each y_i is a non-negative integer. In particular, its formula is given by

$$p_Y(y;\bar{x}) = \prod_{i=1}^{m} \frac{e^{-(H\bar{x})_i}(H\bar{x})_i^{y_i}}{y_i!}. \tag{3.18}$$

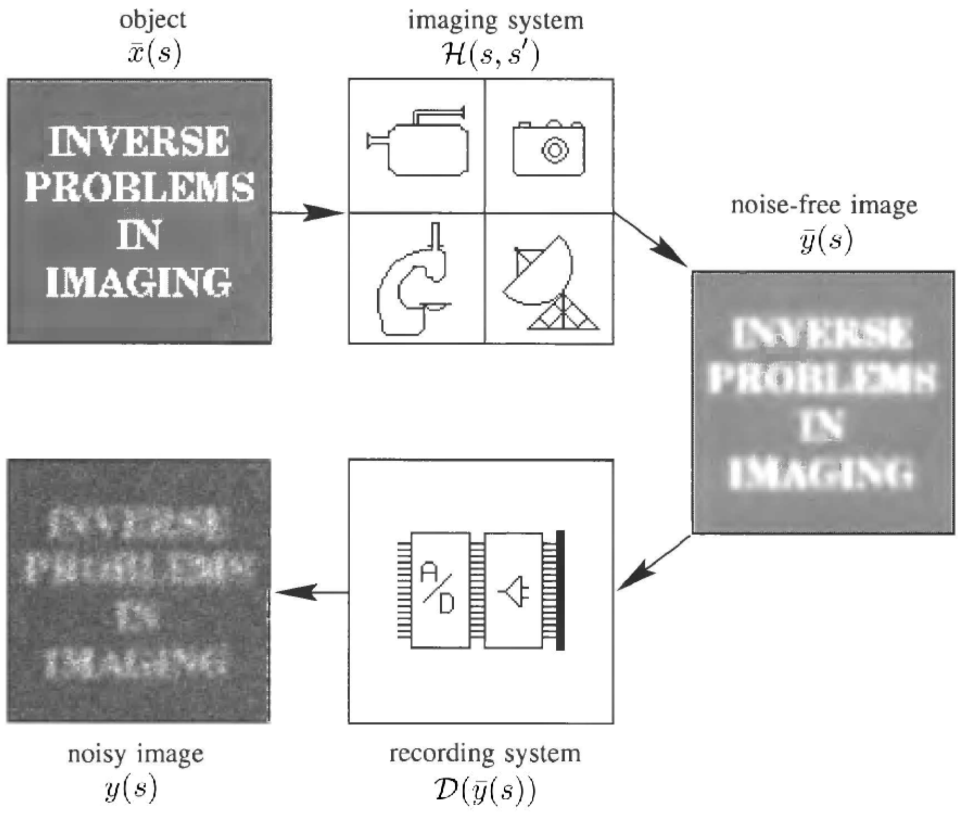


Figure 3.3: Schematic representation of the image acquisition system, from [9].

In conclusion, we have a complete model of the process of data acquisition when we know the imaging matrix H and the probability density $p_Y(y;\bar{x})$. In Figure 3.3 a schematic representation of image acquisition is reported where the two processes, imaging and detection, are clearly separated.

3.3 Statistical Formulation of Image Reconstruction Problem

The inverse problem related to the process of image acquisition, called image reconstruction, or image deblurring, can be formulated as the problem of estimating the unknown object $\bar{x}$, given the detected image y [9, 62]. Assuming to know the probability density $p_Y(y;\bar{x})$, image reconstruction is said to be a non-blind deconvolution problem or a blind deconvolution problem, depending on whether the blurring kernel H is known or unknown, respectively. In our study, we assume to have a complete model in the sense previously defined, so that we

approach the non-blind version of the problem.
As we can see in Figure 3.3, a loss of information is typical in the solution of the direct problem; in particular, the matrix–vector product $H\bar{x}$ has a smoothing effect on $\bar{y}$. This has two important consequences. First, it may be possible for two ore more objects to have the same image. Secondly, two very distant objects may have images which are very close [9]. This make the inverse problem ill–posed, so that, a trivial approach that computes the solution $\bar{x} = H^{-1}(y)$ of the linear system $H\bar{x} = y$ is in general not successful.
Indeed, if we have a noisy image, which is not in the range of the operator H, then the solution of the inverse problem does not exist; and generally noisy images are the only ones available. If an image corresponds to two distinct objects, the solution is not unique. If we have two neighbouring images such that the corresponding objects are very distant, then the solution of the inverse problem does not depend continuously on the data.
Finally, even if the matrix H is non–singular, if it is ill–conditioned, errors in y may be greatly amplified. Certain errors, like the noise in the image recording device, cannot be controlled in a practical setting. Consequently, a direct inversion is likely to fail. It often happens that the object reconstruction is completely deprived of physical meaning as a consequence of error propagation and amplification from the noisy data to the solution. There are also cases in which direct inversion of the linear model (3.12) is practicable, such as computed tomography, but these are exceptions [12].
As information on statistical properties of the data is available, statistical approaches for image reconstruction problem arise quite naturally. These approaches provide a rigorous, effective way to manage noisy data.

3.3.1 Maximum Likelihood Approach

Given the above considerations, we look for a statistical formulation of the image reconstruction problem. Since we assume to know the probability density $p_Y(y; \bar{x})$, where the detected image y is given, and where the object $\bar{x}$ can be seen as vector of unknown parameters, the problem appears as a problem of parameter estimation. In statistics, the standard method to approach it is the so–called maximum likelihood (ML) estimation [126]. This method estimates the parameters of a probability distribution by maximizing a likelihood function, so that, according to the assumed statistical model, the observed data becomes the most probable.

Definition 3.3. *A ML–estimate for the unknown object $\bar{x}$, given y, is any x^* that maximizes the likelihood function*

$$L_y^Y(\bar{x}) = p_Y(y; \bar{x}), \tag{3.19}$$

so that

$$x^* = \underset{x \in \mathbb{R}^n}{\operatorname{argmax}}\, L_y^Y(x). \tag{3.20}$$

Since, from (3.13), the likelihood function is the product of a very large number of factors, the introduction of a logarithm can simplify the ML–estimate computation. Moreover, if we consider the negative logarithm (neglog), which is strictly convex, the maximization problem (3.20) can be reformulated as a minimization one,

$$x^* = \operatorname*{argmin}_{x \in \mathbb{R}^n} \; f(x, y) \tag{3.21}$$

with

$$f(x, y) \equiv -A \ln L_y^Y(x) + B. \tag{3.22}$$

Here A and B are suitable constants introduced to simplify the expression of $f = f(x, y)$. We refer to (3.21) as the variational formulation of the problem provided by the ML–approach, where the functional f is called data–fidelity term, since it measures the distance between the observed data and the data predicted by the linear model.

Since in this process we are maximizing the probability density $p_Y(y; \bar{x})$, different noise models lead to different functionals f. This can be seen for the two noise models introduced in the previous section: the additive white Gaussian noise and the Poisson noise.

Example 3.1 (additive white Gaussian noise)**.** By setting $A = \sigma^2$, $B = A/(2\pi\sigma^2)^{m/2}$, and knowing $p_Y(y; \bar{x})$ from (3.17), in equation (3.22) we obtain the following functional, which is the squared norm of the residual,

$$f(x, y) = \frac{1}{2}\|Hx - y\|^2. \tag{3.23}$$

Therefore the ML approach leads to the classical least–squares (LS) minimization problem. We remark that, given the detected data y, the partial derivative of the LS–functional, with respect to its first variable, can be written as

$$\nabla_x f(x, y) = H^T H x - H^T y. \tag{3.24}$$

Taking into account the assumptions on H, the functional in (3.23) is non–negative, convex, and strictly convex if and only if the equation $Hx = 0$ has only the solution $x = 0$. This means that it always has a global minimum, i.e., the LS–problem always has a solution [111]. Nevertheless, we already said that in image reconstruction the problem is ill–conditioned, since the condition number of the matrix H, that comes from the discretization of an integral operator, can be very large.

Example 3.2 (Poisson noise)**.** Let us introduce the Kullback–Leibler (KL) divergence of a vector x_2 from a vector x_1, defined as

$$\mathrm{KL}(x_1, x_2) = \sum_{i=1}^{m} \left(x_{1i} \ln\left(\frac{x_{1i}}{x_{2i}}\right) + x_{2i} - x_{1i} \right),$$

where we assume that $0 \ln 0 = 0$, and $x_{2i} > 0$ for $i = 1, ..., m$. Taking the probability density $p_Y(y; \bar{x})$ defined in equation (3.18) for the Poisson noise case, if we apply Stirling's formula

$\ln(n!) \approx n\ln(n) - n$, then the expression for the functional $f(x,y)$, with $A = 1$ and $B = 0$, becomes

$$\begin{aligned} f(x,y) &= \mathrm{KL}(y, Hx) \\ &= \sum_{i=1}^{m} \left(y_i \ln\left(\frac{y_i}{(Hx)_i}\right) + (Hx)_i - y_i \right). \end{aligned} \tag{3.25}$$

In particular, the partial derivative of the KL–functional, with respect to x, is given by

$$\nabla_x \mathrm{KL}(y, Hx) = H^T\, e - H^T \frac{y}{Hx}, \tag{3.26}$$

where $e \in \mathbb{R}^m$ denotes the array whose components are all equal to one. In this case, the domain of f is the non–negative orthant[1]; moreover, the function is convex and strictly convex if the additional condition $y_i > 0$ for $i = 1, ..., m$ is satisfied, and the equation $Hx = 0$ has only the solution $x = 0$ [111]. The functional is also non–negative and locally bounded. Thus, it has global minima. Accurate analysis of the properties of this functional can be found in [104, 105]. Neverthless, noise is expected to strongly affect the minima of the discrete problem; this is confirmed by a typical effect of the noise introduction, which is known as *checkerboard effect*, due to the fact that many components of the minima are zero.

For both types of noise, we highlighted that the ML–problem (3.21) is still affected by ill–posedness, or by the ill–conditioning of the matrix H [9]. In the framework of inverse problems, we already said that, if no additional information on the object is employed, the resulting problem is ill–posed. This is the case with ML–approach, where only information about the noise is used, with possibly the introduction of non–negativity constraints when formulating the minimization problem. For this reason we are not interested in computing the exact minimum points of the functional f, since they do not provide sensible estimates of the unknown object $\bar{x}$.
In this sense, we must be very careful in applying methods derived from optimization theory to ML–problems. In particular, second–order methods, pointing directly to the minima, can be dangerous. On the other hand, iterative first–order methods can provide acceptable solutions by arresting the algorithm before convergence through some early stopping criteria, as the Morozov's discrepancy principle in the case of Gaussian noise [111]. In the framework of regularization theory, iterative methods of this type are widely investigated; like for example Landweber, steepest descent and conjugate gradient methods.

3.3.2 Maximum A Posteriori Approach

A more complete statistical framework is provided by the Bayesian approach to estimation [53], in which the additional information can be incorporated in the form of statistical properties

[1]In geometry, an orthant is the analogue in $\mathbb{R}^n$ of a quadrant in $\mathbb{R}^2$. The nonnegative orthant is the generalization of the first quadrant to n dimensions.

of the object. In particular, in this approach we assume that also the unknown object $\bar{x}$ is a realization of a vector-valued random variable X, so that the probability density of X, denoted by $p_X(\bar{x})$, can be defined. This is the so–called a priori probability density, or simply prior, because it is used to express one's beliefs about the object before some evidence is taken into account. For example, some properties of the object can be incorporated in the prior $p_X(\bar{x})$, such as smoothness, sharp edges, etc.
Priors of the Gibbs type are the most used; they can be written as

$$p_X(x) = \frac{1}{Z}\exp\left(-\lambda\Omega(x)\right) \tag{3.27}$$

where Z is a normalization constant, $\lambda \in \mathbb{R}_{>0}$ is a positive parameter, and $\Omega(x)$ is a functional which is generally convex.
Now, the probability density $p_Y(y;\bar{x})$ is considered as the conditional probability density of Y, when $\bar{x}$ is the value assumed by the random variable X,

$$p_Y(y|X=\bar{x}) = p_Y(y;\bar{x}),$$

which is denoted by $p_Y(y|\bar{x})$, for simplicity.
Then, introducing the marginal probability density $p_Y(y)$ of Y, we can apply the Bayes theorem [7], to compute the conditional probability density of X, with respect to the given value y of Y:

$$p_X(\bar{x}|y) = \frac{p_Y(y|\bar{x})p_X(\bar{x})}{p_Y(y)}. \tag{3.28}$$

Finally, by inserting the value of detected image y in (3.28), we obtain the a posteriori probability density of X

$$P_y^X(\bar{x}) = p_X(\bar{x}|y) = L_y^Y(\bar{x})\frac{p_X(\bar{x})}{p_Y(y)}. \tag{3.29}$$

Definition 3.4. *For a given detected image y, a maximum a posteriori (MAP) estimate of the unknown object $\bar{x}$ is any x^* that maximizes the a posteriori probability $P_y^X(\bar{x})$, so that*

$$x^* = \underset{x\in\mathbb{R}^n}{\operatorname{argmax}}\; P_y^X(x). \tag{3.30}$$

As we did for the likelihood case, to simplify the computations it is convenient to consider the equivalent formulation, where we apply the negative logarithm of $P_y^X(x)$. Then, assuming to have a Gibbs prior $p_X(x)$ as in equation (3.27), the MAP estimate of $\bar{x}$ is given by

$$x^* = \underset{x\in\mathbb{R}^n}{\operatorname{argmin}}\, J(x,y) \tag{3.31}$$

where, recalling equation (3.22), the following functional is introduced

$$\begin{aligned} J(x,y) &= -A\ln P_y^X(x) + B - A\ln Z - A\ln p_Y(y) \\ &= -A\ln L_y^Y(x) + B + \lambda A\Omega(x) \\ &= f(x,y) + \lambda\mathcal{R}(x). \end{aligned}$$

In the above variational formulation of the problem provided by the MAP–approach, the sought object can be approximated by the minimizer of a regularized cost functional, which is expressed as the sum of a data-fidelity term $f(x, y)$, and a regularization term $\mathcal{R}(x) = A\Omega(x)$. In particular, the data–fitting functional $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ is used to control the fidelity of the solution to the degradation model (3.12). On the other hand, the functional $\mathcal{R} : \mathbb{R}^n \to \mathbb{R}$, that comes from the Gibbs prior, is used to induce stability and to encode some prior information on the desired solution. In fact, we have already mentioned in Chapter 1 that, by including the regularization term in the functional to be minimized, we are giving preference to a particular solution with desired properties. The influence of $\mathcal{R}$ with respect to f, and so the importance for the solution to have the requested properties, is weighted by a scalar parameter $\lambda \in]0, +\infty[$, called regularization parameter.
We remark that, contrary to the maximum likelihood problem, where we do not want to reach a minimum of the functional f, we now aim to find the minimizer x^* of $J(x, y)$. In this case, the quality of the reconstruction x^* obtained with the MAP–approach will highly depend on how the regularization parameter λ is selected. The problem of the optimal choice for λ is not trivial, and it has been extensively studied in regularization theory [50]. This problem is addressed in our study too, and, in particular, it is one of the foundations in the construction of our method, as we will see in Chapter 4.

3.3.3 Regularization Functionals

In the following, D indicates a discrete gradient operator, i.e., $D = (D_1^T, \ldots, D_n^T)^T$ is a matrix where $D_i \in \mathbb{R}^{2\times n}$ computes the forward finite differences at the i–th pixel of an image $x \in \mathbb{R}^n$. We can write

$$(Dx)_i = \begin{pmatrix} (D_\mathrm{v}x)_i \\ (D_\mathrm{h}x)_i \end{pmatrix} = \begin{pmatrix} x_{i+1} - x_i \\ x_{i+m} - x_i \end{pmatrix}, \tag{3.32}$$

where we assume that x is the vectorized version of a 2-D image of size $(m \times m)$. In the above definition, $D_\mathrm{v} \in \mathbb{R}^{n\times n}$ and $D_\mathrm{h} \in \mathbb{R}^{n\times n}$ are called vertical and horizontal gradient operators, respectively.

Let us recall now some classical regularization functionals, which are the most used in image deblurring.

Tikhonov Regularization

Given a linear operator $L \in \mathbb{R}^{n\times n}$, the choice of giving preference to solutions with smaller norms, by taking the regularization term as

$$\mathcal{R}(x) = \frac{1}{2}\|Lx\|^2 \tag{3.33}$$

is known as generalized Tikhonov regularization [132, 133]. Often, L represents a gradient operator. In particular, according to the choice of L, we distinguish between zero–order, and first–order Tikhonov regularization:

- $L = I$ (zero–order);
- $L = D$ (first–order).

This type of regularization is used to enforce smoothness, if the underlying vector is believed to be mostly continuous. It is to be noted that the theory behind this regularization is based on the assumption that the noise affecting the data is additive and Gaussian [12], therefore this kind of regularization is generally associated to least–squares data–fidelity functional.
The Tikhonov regularization functional is continuously differentiable and convex. We recall its gradient, which has a simple formula

$$\nabla \mathcal{R}(x) = L^T L\, x. \tag{3.34}$$

Edge–Preserving Regularization

In many applications, objects to be restored manifest sharp features. For example, natural images usually contain multiple areas of different colors, enclosed by marked edges. Therefore, a method used to recover clear photographs from misfocused noisy snapshots should be edge-preserving.
As we said, using Tikhonov regularizers generally results in smooth reconstructions, especially with derivative penalty. An alternative method, the so-called Total Variation (TV) functional, was introduced in [118] to preserve discontinuities and edges. The basic idea is to replace, in a continuous environment, the ℓ^2-norm of the generalized Tikhonov regularization by ℓ^1-norm, thus leading to the discrete version [138]

$$\begin{aligned} TV(x) &= \sum_{i=1}^{n} \|(Dx)_i\| \\ &= \sum_{i=1}^{n} \sqrt{(D_\mathrm{v}x)_i^2 + (D_\mathrm{h}x)_i^2}, \end{aligned}$$

where D is the discretization of the gradient operator, as defined in (3.32), so that $D_\mathrm{v} \in \mathbb{R}^{n\times n}$ and $D_\mathrm{h} \in \mathbb{R}^{n\times n}$ are the vertical and horizontal gradient operators, respectively.

In 2-D image deblurring, TV regularization is proved to be successful in the reconstruction of qualitatively correct blocky images [46], where by blocky we intend that images are nearly piecewise constant with jump discontinuities.

The functional $TV(x)$ is convex; however, it is not suitable for the implementation of gradient-based methods, because it is nondifferentiable at any point x, such that $(Dx)_i = 0$, for some $i \in \{1, \dots, n\}$. Then, in order to remove the singularity at the origin and recover differentiability, one can introduce an approximation $\sqrt{\|.\|^2 + \delta^2}$ of the Euclidean norm $\|.\|$, by inserting a small positive parameter $\delta \in \mathbb{R}_{>0}$. This yields to a frequently used regularization functional, which is a smooth approximation of total variation,

$$\mathrm{HS}(x) = \sum_{i=1}^{n} \sqrt{\|(Dx)_i\|^2 + \delta^2} = \sum_{i=1}^{n} \sqrt{(D_v x)_i^2 + (D_h x)_i^2 + \delta^2}. \tag{3.35}$$

Formulation (3.35) is known as smoothed total variation, or also Hypersurface Potential (HS) functional [27, 138], and δ is the so–called smoothing parameter. Again, this functional is non–negative and strictly convex [111]. In particular, this is a generalization of the classical TV, which can be obtained by choosing $\delta = 0$.

We recall the gradient of the smoothed total variation functional, which is given by

$$\nabla \operatorname{HS}(x) = \sum_{i=1}^{n} \frac{D_v^T (D_v x)_i + D_h^T (D_h x)_i}{\sqrt{(D_v x)_i^2 + (D_h x)_i^2 + \delta^2}}. \tag{3.36}$$

3.3.4 Hard Constraints

In order to find an appropriate solution to an ill-posed inverse problem, prior information on the sought object $\bar{x}$ can be incorporated not only through regularization functions, but also through the introduction of hard constraints in the variational formulation of the problem. They are usually introduced when the information consists in prescribed bounds on the solution, and/or on its derivatives up to a certain order. For example, it may be known that the object must be non-negative or that it must be different from zero only inside a specific region. Working with images, range constraints can be imposed on the pixel intensities, in order to avoid meaningless solutions, outside the brightness range.

For all these examples, we can restrict the search for the solution to a nonempty, closed and convex set $\mathcal{C} \subset \mathbb{R}^n$, which is the feasible set of all the candidates that satisfy the constraints. This set is defined by equations and inequalities,

$$\mathcal{C} = \{x \in \mathbb{R}^n \mid \forall l \in \{1, \dots, p'\} \; g_l(x) = 0, \; \forall j \in \{1, \dots, p\} \; c_j(x) \geq 0\}, \tag{3.37}$$

where $g_l(x) = 0$, for $l = 1, \dots, p'$, and $c_j(x) \geq 0$, $j = 1, \dots, p$, are equality and inequality constraints, respectively, which determine the properties to be satisfied a priori by the image.

Among the most typical examples of constraints, we recall:

- conservation of the flux: $\mathcal{C} = \{x \in \mathbb{R}^n \mid \sum_{i=1}^n x_i = c\}$, where $c \in \mathbb{R}$ is a flux constant;
- affine constraints: $\mathcal{C} = \{x \in \mathbb{R}^n \mid a^T x \leq b\}$, where $a \in \mathbb{R}^n \setminus \{0\}$ is a coefficient vector and $b \in \mathbb{R}$; as a special case, it is a very common choice to require non-negative values, so that $\mathcal{C}$ is chosen as the nonnegative orthant: $\mathcal{C} = \mathbb{R}^n_{\geq 0}$;
- hyperslab constraints: $\mathcal{C} = \{x \in \mathbb{R}^n \mid b_{\mathrm{m}} \leq a^T x \leq b_{\mathrm{M}}\}$, where $a \in \mathbb{R}^n \setminus \{0\}$, $b_{\mathrm{m}} \in \mathbb{R}$ and $b_{\mathrm{M}} \in \mathbb{R}$ with $b_{\mathrm{m}} < b_{\mathrm{M}}$; a particular case is given by box constraints, where some physical bounds $a, b \in \mathbb{R}^n$ are imposed on the object, so that $\mathcal{C} = \{x \in \mathbb{R}^n \mid a_i \leq x_i \leq b_i, \; i = 1, \dots, n\}$;
- Bounded ℓ^2-norm constraints: we search the solution inside the Euclidean ball $\mathcal{C} = B(c, \alpha) = \{x \in \mathbb{R}^n \mid \|x - c\| \leq \alpha\}$, of center $c \in \mathbb{R}^n$ and radius $\alpha \in \mathbb{R}_{>0}$.

3.4 Interior Point Methods

3.4.1 Problem Formulation

We have now introduced all the elements to approach the image reconstruction problem, which is related to the discrete model defined in equation (3.12),

$$y = \mathcal{D}(H\bar{x}), \tag{3.38}$$

where $y \in \mathbb{R}^m$ is the observed data, $\bar{x} \in \mathbb{R}^n$ is the sought image, $H \in \mathbb{R}^{m\times n}$ is the observation operator, which is assumed linear, and $\mathcal{D}$ is the noise perturbation operator. The linear operator H is assumed to be known from a physical model or prior identification step [89, 143]. In order to find an appropriate reconstruction, we follow a MAP–approach and we restrict the search for the solution to a given feasible set. This leads to the following constrained minimization problem,

$$\min_{x\in\mathcal{C}} \; f(Hx, y) + \lambda\mathcal{R}(x) \tag{3.39}$$

where $f : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ is the data–fidelity function, $\mathcal{R} : \mathbb{R}^n \to \mathbb{R}$ is the regularization function, with regularization parameter $\lambda \in \mathbb{R}_{>0}$, and $\mathcal{C} \in \mathbb{R}^n$ is the feasible set.

For every $q \in \mathbb{N}$, we denote as $\Gamma_0(\mathbb{R}^q)$ the set of functions which are convex, proper, lower semicontinuous on $\mathbb{R}^q$, and which take values in $\mathbb{R} \cup \{+\infty\}$. Hereafter, for every noisy image $y \in \mathbb{R}^m$, we assume that $f(\cdot, y) \in \Gamma_0(\mathbb{R}^m)$ is a twice-differentiable data-fidelity function, and $\mathcal{R} \in \Gamma_0(\mathbb{R}^n)$ is a twice-differentiable regularization term. It is to be noted that the twice-differentiability condition is necessary to apply our method, as presented in Chapter 4. In particular, it is required when defining the derivative steps involved in the backpropagation procedure for the training of the proposed network.

Moreover, in the rest of the thesis we will use the following notation: for all $(x, y, \lambda) \in \mathbb{R}^n \times \mathbb{R}^m \times\ ]0, +\infty[$, the objective function is written as

$$h(x, y, \lambda) = f(Hx, y) + \lambda\mathcal{R}(x), \tag{3.40}$$

while its partial gradient with respect to x is given by

$$\nabla_1 h(x, y, \lambda) = H^T \nabla_1 f(Hx, y) + \lambda\nabla\mathcal{R}(x), \tag{3.41}$$

where $\nabla_1 f$ denotes the partial gradient of f with respect to its first variable.

Regarding the feasible set $\mathcal{C}$, it is defined by p inequality constraints, which determine the properties to be satisfied a priori by the image:

$$\mathcal{C} = \{x \in \mathbb{R}^n \mid c_j(x) \geq 0,\ \forall j = 1, \ldots, p\}, \tag{3.42}$$

where, $-c_j \in \Gamma_0(\mathbb{R}^n)$ for every $j = 1, \ldots, p$. Also, the strict interior of the feasible domain, denoted int$\mathcal{C}$, is equal to

$$\text{int}\mathcal{C} = \{x \in \mathbb{R}^n \mid c_j(x) > 0,\ \forall j = 1, \ldots, p\}, \tag{3.43}$$

and it is assumed to be nonempty.

Finally, we assume that either the function $h(\cdot, y, \lambda) = f(H\cdot, y) + \lambda\mathcal{R}$ is coercive, or $\mathcal{C}$ is bounded. In this way, the existence of solutions to problem (3.39) is guaranteed. It is worth emphasizing that, in the literature regarding image reconstruction, a large class of regularized formulations meets the above requirements, see e.g. [49] and references therein.

3.4.2 Interior Point Approaches

In general, without additional conditions, the variational formulation of the image reconstruction problem provided in (3.39), does not have a closed-form solution on account of the inequality constraints, even for simple regularizations. Hence an iterative solver is required, able to provide a meaningful solution in a reasonable time. Over the past decades, iterative reconstruction methods have achieved remarkable results to solving inverse problems in imaging, including deconvolution [26, 87]. Thanks to robust regularizers such as total variation, practical algorithms have appeared with excellent image quality and reasonable computational complexity.

Among all possible choices, first-order methods are particularly suited to deal with this kind of problems for several reasons. First, due to the large size of the images, to handle the Hessian matrix is an impractical task. Then, by using second-order methods with an excessively fast convergence, it can happen that a difference of few iterations from the one providing the optimal reconstruction can lead to substantial differences in the final images.
Several approaches are available, either based on projected gradient strategies [16, 75], ADMM [24], primal-dual schemes [86], or interior point techniques [17]. When feasibility at intermediate points is essential, as in practical problems such as ours, where points are meaningless outside the boundary constraints, it is desirable for iterates to approach the solution from the interior of the feasible region. Interior point methods constitute a well known class of methods with this property. They have been useful for the resolution of small to medium size constrained optimization problems [77]. However, in their standard formulation, interior point methods require to invert several $n \times n$ linear systems, which leads to a high computational complexity for large scale problems. Nonetheless, it has recently been shown that combining the interior point framework with a proximal forward–backward strategy [34, 36] leads to very competitive solvers for inverse problems [32, 38, 39].
Based on these considerations, we decided to start from a particular proximal interior point algorithm for the development of a novel neural network, able to face problem (3.39).

The idea behind interior point methods (IPMs) is to replace the initial constrained optimization problem (3.39) with a sequence of unconstrained subproblems, easier to handle. To do this, we replace the inequality constraints with an additional penalizing term $\mathcal{B}$, parametrized by a sequence of parameters $\{\mu_k\}_{k\in\mathbb{N}}$, with $0 < \mu_{k+1} < \mu_k$, so that the problem becomes

$$\min_{x\in\mathbb{R}^n} \; f(Hx, y) + \lambda\mathcal{R}(x) + \mu_k\mathcal{B}(x). \tag{3.44}$$

The new function $\mathcal{B}$, called the barrier function, has unbounded derivative at the boundary of the feasible domain. In this way it acts as a barrier, so that its minimization prevents the iterates from leaving the feasible region. It is to be noted that the solution of (3.44) does not coincide with that of (3.39) for $\mu > 0$. The unconstrained minimization of this augmented objective function is then carried out for a sequence of positive barrier parameters μ_k that converges to zero. In this way, the barrier term $\mu_k \mathcal{B}(x)$ becomes increasingly irrelevant for all interior points $x \in \mathcal{C}$, while progressively allowing x_k to get closer to the boundary of $\mathcal{C}$.

In IPMs, we consider $\mathcal{B} : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ to be the logarithmic barrier function,

$$(\forall x \in \mathbb{R}^n) \quad \mathcal{B}(x) = \begin{cases} -\displaystyle\sum_{i=1}^{p} \ln(c_i(x)) & \text{if } x \in \text{int}\mathcal{C} \\ +\infty & \text{otherwise.} \end{cases} \tag{3.45}$$

As we can see, this is a continuous function whose value increases to infinity as it approaches the boundary of $\mathcal{C}$ (recall $-\log t \to +\infty$, as $t \to 0^+$).
We assumed that either $h(\cdot, y, \lambda) = f(H\cdot, y) + \lambda\mathcal{R}$ is coercive, or $\mathcal{C}$ is bounded, then the set of solutions to (3.39) is bounded. Since int$\mathcal{C}$ is not empty, it can be shown that the existence of solutions to the unconstrained formulation (3.44), containing a logarithmic barrier function, is guaranteed, see [142, Theorem 5(ii)].

As we said, one advantage of IPMs is that only feasible iterates are produced, which can be beneficial from the application viewpoint and can also boost convergence. On the other hand, these methods have high computational costs for large scale problems. The main idea behind our algorithm is to combine the logarithmic barrier method, which ensures the feasibility of the iterates, with a proximal algorithm for solving convex optimization problems.

Proximity Operator

The notion of proximity operator was first introduced by Moreau in [103].

Definition 3.5. *Let $f \in \Gamma_0(\mathbb{R}^n)$. For every $x \in \mathbb{R}^n$, the minimization problem*

$$\min_{u \in \mathbb{R}^n} \frac{1}{2}\|x - u\|^2 + f(u) \tag{3.46}$$

admits a unique solution, which is denoted by $\text{prox}_f(x)$. *The operator* $\text{prox}_f : \mathbb{R}^n \to \mathbb{R}^n$ *thus defined is called the proximity operator of f.*

In particular, in this thesis, given $f \in \Gamma_0(\mathbb{R}^n)$ and $\gamma \in \mathbb{R}_{>0}$, for every $x \in \mathbb{R}^n$, we consider the proximity operator of a scaled function γf at x:

$$\text{prox}_{\gamma f}(x) = \underset{u \in \mathbb{R}^n}{\text{argmin}} \frac{1}{2}\|x - u\|^2 + \gamma f(u). \tag{3.47}$$

The above definition indicates that $\text{prox}_f(x)$ is an operator which compromises between minimizing f and being near x. Considering the scaled version $\text{prox}_{\gamma f}(x)$, the parameter γ can be interpreted as a trade-off parameter between these two terms.

Remark 3.1. Let us consider the indicator function of a non empty, closed and convex subset $\mathcal{C} \subseteq \mathbb{R}^n$, i.e.,

$$\iota_{\mathcal{C}}(x) = \begin{cases} 0, & \text{if } x \in \mathcal{C} \\ +\infty, & \text{if } x \notin \mathcal{C}. \end{cases}$$

The proximity operator of the indicator function $\iota_{\mathcal{C}}$ coincides with the projection operator [34],

$$\operatorname{prox}_{\iota_{\mathcal{C}}}(x) = \underset{u \in \mathbb{R}^n}{\operatorname{argmin}} \frac{1}{2}\|x-u\|^2 + \iota_{\mathcal{C}}(u) = P_{\mathcal{C}}(x),$$

where the standard Euclidean projection operator onto $\mathcal{C}$, $P_{\mathcal{C}} : \mathbb{R}^n \to \mathcal{C}$ is defined as

$$P_{\mathcal{C}}(x) = \underset{u \in \mathcal{C}}{\operatorname{argmin}} \|x-u\|. \tag{3.48}$$

Proximity operators are therefore an extension of projection operators, whereby the indicator function $\iota_{\mathcal{C}}$ is replaced by an arbitrary function $f \in \Gamma_0(\mathbb{R}^n)$.

3.4.3 Proposed Iterative Schemes

Thanks to the introduction of a proximity operator in the proposed IPM (3.44), the method does not require any matrix inversion. In particular, when the proximity operator is computed in an exact manner, i.e., by calculating $\operatorname{prox}_{\gamma_k(h(\cdot,y,\lambda)+\mu_k\mathcal{B})}(x)$, the proposed IPM can be rewritten as a proximal point algorithm [79]. This method is reported in Algorithm 13, and convergence results can be found in [79, Theorem 4.1], under particular assumptions.

Algorithm 13 Exact version of the proximal IPM in [79], applied to problem (3.39).

```
Let $x_0 \in \operatorname{int}\mathcal{C}$, $\underline{\gamma} > 0$ and $(\gamma_k)_{k\in\mathbb{N}}$ be a sequence such that $(\forall k \in \mathbb{N})\ \underline{\gamma} \le \gamma_k$;
for $k = 0, 1, \ldots$ do
    $x_{k+1} = \operatorname{prox}_{\gamma_k(h(\cdot,y,\lambda)+\mu_k\mathcal{B})}(x_k)$
end for
```

Evaluating the proximity operator of a function involves solving a convex optimization problem. For this reason proximal algorithms are most useful when all the relevant proximity operators have closed-form expressions, and can be evaluated in a reasonable computational time.

In our case, Algorithm 13 requires computing the proximity operator of the sum of the regularized cost function $h(\cdot, y, \lambda)$ and the barrier term $\mu_k\mathcal{B}$. This can be problematic since, in most cases, this operator does not have a closed-form expression. For this reason, we modify Algorithm 13 by introducing a gradient step at each iteration, which leads to a forward-backward method.

Among the several numerical strategies designed to address (3.44), forward-backward methods

[34, 36] have gained great popularity in the last years, because they can handle the nondifferentiable barrier term arising in the objective function. Such algorithms deal with the functions $h(x, y, \lambda)$ and $\mu\mathcal{B}(x)$ separately, by alternating, at each iteration, a *forward* gradient step on the differentiable term $h(x, y, \lambda)$ with a *backward* proximal step onto the nondifferentiable logarithmic barrier. To be more specific, the backward step evaluates the proximity operator of $\mu\mathcal{B}(x)$, at the point obtained by applying to $h(x, y, \lambda)$ a standard gradient descent step, as defined in (2.17).
The proposed method is reported in Algorithm 14.

Algorithm 14 Proposed forward–backward proximal IPM.

```
Let $x_0 \in \mathrm{int}\mathcal{C}$, $\underline{\gamma} > 0$ and $(\gamma_k)_{k\in\mathbb{N}}$ be a sequence such that $(\forall k \in \mathbb{N})\ \underline{\gamma} \le \gamma_k$;
for $k = 0, 1, \dots$ do
    $x_{k+1} = \mathrm{prox}_{\gamma_k \mu_k \mathcal{B}} \left(x_k - \gamma_k \nabla_1 h\left(x_k, y, \lambda\right)\right)$
end for
```

In the algorithm we consider the gradient $\nabla_1 h\,(., y, \lambda)$ as provided in (3.41), while $\{\gamma_k\}_{k\in\mathbb{N}}$ is the sequence of stepsizes used along the iterations.
As far as we know, among the literature on interior-point methods there is no convergence study available for Algorithm 14. There are links between this algorithm and the diagonal or penalization method introduced in [42]. Indeed, taking $A \equiv 0$ and $\Psi_1 \equiv 0$ in [42] leads to Algorithm 14, so that convergence is proven. However, there are some important differences between the two approaches, that are: *i)* the algorithm in [42] solves a hierarchical minimization problem instead of the constrained optimization problem (3.39), and *ii)* in [42] the barrier parameter tends to infinity while in our case it tends to zero.

Again, let us remark that Algorithm 14 involves, at each iteration, the solution of the convex subproblem related to the evaluation of the proximity operator at the gradient point. Therefore, its solution must be known in closed-form or, at least, within a certain accuracy, in order for the method to be effective. In Section 3.5, we will provide the expression for the proximity operator of the logarithmic barrier, for three different types of constraints, which are examples of interest.

3.4.4 Limitations

In IPMs, the sequences of the barrier parameter and stepsize, $\{\mu_k\}_{k\in\mathbb{N}}$ and $\{\gamma_k\}_{k\in\mathbb{N}}$, are generally set by following some heuristic rules, so that the convergence of the method to a minimizer of the objective function is guaranteed. However, heuristic approaches may not find the best values for the parameters, with consequent loss in efficiency and versatility of the resulting reconstruction scheme. For example, by using handcrafted variational formulations, we may not achieve a satisfactory visual quality. This is even more true if we consider the selection of the regularization parameter λ. As already mentioned in Section 3.3.2, an accurate setting of the regularization parameter is particularly critical, because the quality of the reconstruction obtained with regularized approaches highly depends on it. Existing approaches for the selection of λ, which are based on statistical considerations, generally lead to a substantial increase in computational cost.

To overcome these limitations, our strategy consists in exploiting neural networks in order to learn the stepsize, the barrier and the regularization parameters for every iteration of Algorithm 14, in a supervised fashion. Backpropagation algorithm will be used for the training step in our deep learning method. In particular, to apply backpropagation, the derivatives of the proximity operator in Algorithm 14 are required, with respect to its input and to the aforementioned parameters that must be learned. Therefore, we conclude this chapter by deriving the expression of the proximity operator of the barrier and of its derivatives, for three common types of constraints, one of which is the one considered in the experiments.

3.5 Proximity Operator of the Barrier

Although computing the proximity operator in numerical applications is a challenging task, in certain cases closed-form expressions are available [34, 36].

In our case, let $\mathcal{B}$ be defined as in (3.45) and for all $\mu > 0$, $\gamma > 0$ and $x \in \mathbb{R}^n$, let φ be defined as follows:

$$\varphi(x, \mu, \gamma) = \operatorname{prox}_{\gamma\mu\mathcal{B}}(x). \tag{3.49}$$

We provide in this section expressions of φ and of its derivatives with respect to its input variable x and the involved barrier and stepsize parameters (μ, γ), for three different types of constraints. The latter will be necessary for training the proposed neural network using a gradient backpropagation scheme.

3.5.1 Affine Constraints

Let us first consider the following half-space constraint, introduced in Section 3.3.4:

$$\mathcal{C} = \{x \in \mathbb{R}^n \mid a^T x \leq b\}, \tag{3.50}$$

with $a \in \mathbb{R}^n \setminus \{0\}$ and $b \in \mathbb{R}$.

Proposition 3.1. *Let $\gamma > 0$, $\mu > 0$, and let $\mathcal{B}$ be the function associated to* (3.50)*, defined as*

$$(\forall u \in \mathbb{R}^n) \quad \mathcal{B}(u) = \begin{cases} -\ln(b - a^T u) & \text{if } a^T u < b, \\ +\infty & \text{otherwise.} \end{cases} \tag{3.51}$$

Then, for every $x \in \mathbb{R}^n$, the proximity operator of $\gamma\mu\mathcal{B}$ at x is given by

$$\varphi(x, \mu, \gamma) = x + \frac{b - a^T x - \sqrt{(b - a^T x)^2 + 4\gamma\mu\|a\|^2}}{2\|a\|^2} a. \tag{3.52}$$

In addition, the Jacobian matrix of φ with respect to x and the gradients of φ with respect to μ and γ are given by

$$J_{\varphi}^{(x)}(x,\mu,\gamma) = I_n - \frac{1}{2\|a\|^2}\left(1 + \frac{a^Tx - b}{\sqrt{(b-a^Tx)^2 + 4\gamma\mu\|a\|^2}}\right)aa^T, \tag{3.53}$$

$$\nabla_{\varphi}^{(\mu)}(x,\mu,\gamma) = \frac{-\gamma}{\sqrt{(b-a^Tx)^2 + 4\gamma\mu\|a\|^2}}a, \tag{3.54}$$

and

$$\nabla_{\varphi}^{(\gamma)}(x,\mu,\gamma) = \frac{-\mu}{\sqrt{(b-a^Tx)^2 + 4\gamma\mu\|a\|^2}}a, \tag{3.55}$$

where $I_n \in \mathbb{R}^{n\times n}$ denotes the identity matrix.

Proof. Given the function $\mathcal{B}$, as defined in equation (3.51), the expression of the proximity operator (3.52) can be obtained by taking into consideration the following statements.

Proposition 3.2. *[6, Example 24.40] Let $\gamma \in \mathbb{R}_{>0}$, and consider the function defined as*

$$(\forall \xi \in \mathbb{R}) \quad \phi(\xi) = \begin{cases} -\ln \xi & \text{if } \xi > 0, \\ +\infty & \text{otherwise.} \end{cases} \tag{3.56}$$

Then $\operatorname{prox}_{\gamma\phi}(\xi) = \xi + \sqrt{\xi^2 + 4\gamma}/2$, $\forall \xi \in \mathbb{R}$. (see proof in [6, Proposition 24.1]).

Proposition 3.3. *[6, Proposition 24.8 (vi)] Let $f \in \Gamma_0(\mathbb{R}^n)$, and $x \in \mathbb{R}^n$.
If $g \in \Gamma_0(\mathbb{R}^n)$ is the reversal of f, i.e., $g(x) = f(-x)$, then $\operatorname{prox}_g(x) = -\operatorname{prox}_f(-x)$.*

Corollary 3.1. *[6, Corollary 24.15] Suppose that $a \in \mathbb{R}^n \setminus \{0\}$, $\phi \in \Gamma_0(\mathbb{R}^n)$, and let $\mathcal{B} = \phi((.,a))$. Then, given $x \in \mathbb{R}^n$,*

$$\operatorname{prox}_{\mathcal{B}}(x) = x + \frac{\operatorname{prox}_{\|a\|^2\phi}(x,a) - (x,a)}{\|a\|^2}a.$$

By combining the two propositions with the corollary, we are able to compute expression (3.52), from $\mathcal{B}$. Finally, taking the derivative of (3.52) with respect to x, μ and γ leads to (3.53)–(3.55). □

3.5.2 Hyperslab Constraints

We now consider the following hyperslab set:

$$\mathcal{C} = \{x \in \mathbb{R}^n \mid b_{\mathrm{m}} \leq a^T x \leq b_{\mathrm{M}}\}, \tag{3.57}$$

where $a \in \mathbb{R}^n \setminus \{0\}$, $b_{\mathrm{m}} \in \mathbb{R}$ and $b_{\mathrm{M}} \in \mathbb{R}$ with $b_{\mathrm{m}} < b_{\mathrm{M}}$.

Proposition 3.4. *Let $\gamma > 0$, $\mu > 0$, and let $\mathcal{B}$ be the barrier function associated to* (3.57)*, defined as*

$$(\forall u \in \mathbb{R}^n) \quad \mathcal{B}(u) = \begin{cases} -\ln(b_{\mathrm{M}} - a^T u) - \ln(a^T u - b_{\mathrm{m}}) & \text{if } b_{\mathrm{m}} < a^T u < b_{\mathrm{M}}, \\ +\infty & \text{otherwise.} \end{cases} \tag{3.58}$$

Then, for every $x \in \mathbb{R}^n$, the proximity operator of $\gamma\mu\mathcal{B}$ at x is given by

$$\varphi(x, \mu, \gamma) = x + \frac{\kappa(x, \mu, \gamma) - a^T x}{\|a\|^2} a, \tag{3.59}$$

where $\kappa(x, \mu, \gamma)$ is the unique solution in $]b_{\mathrm{m}}, b_{\mathrm{M}}[$, of the following cubic equation:

$$\begin{aligned} 0 = {} & z^3 - (b_{\mathrm{m}} + b_{\mathrm{M}} + a^T x) z^2 + (b_{\mathrm{m}} b_{\mathrm{M}} + a^T x(b_{\mathrm{m}} + b_{\mathrm{M}}) - 2\gamma\mu\|a\|^2) z \\ & - b_{\mathrm{m}} b_{\mathrm{M}} a^T x + \gamma\mu(b_{\mathrm{m}} + b_{\mathrm{M}})\|a\|^2. \end{aligned} \tag{3.60}$$

In addition, the Jacobian matrix of φ with respect to x and the gradients of φ with respect to μ and γ are given by

$$J_{\varphi}^{(x)}(x, \mu, \gamma) = I_n + \frac{1}{\|a\|^2} \left(\frac{(b_{\mathrm{M}} - \kappa(x, \mu, \gamma))(b_{\mathrm{m}} - \kappa(x, \mu, \gamma))}{\eta(x, \mu, \gamma)} - 1 \right) a a^T, \tag{3.61}$$

$$\nabla_{\varphi}^{(\mu)}(x, \mu, \gamma) = \frac{-\gamma(b_{\mathrm{m}} + b_{\mathrm{M}} - 2\kappa(x, \mu, \gamma))}{\eta(x, \mu, \gamma)} a, \tag{3.62}$$

and

$$\nabla_{\varphi}^{(\gamma)}(x, \mu, \gamma) = \frac{-\mu(b_{\mathrm{m}} + b_{\mathrm{M}} - 2\kappa(x, \mu, \gamma))}{\eta(x, \mu, \gamma)} a, \tag{3.63}$$

where

$$\begin{aligned} \eta(x, \mu, \gamma) = {} & (b_{\mathrm{M}} - \kappa(x, \mu, \gamma))(b_{\mathrm{m}} - \kappa(x, \mu, \gamma)) \\ & - (b_{\mathrm{m}} + b_{\mathrm{M}} - 2\kappa(x, \mu, \gamma))(\kappa(x, \mu, \gamma) - a^T x) - 2\gamma\mu\|a\|^2. \end{aligned} \tag{3.64}$$

Proof. Let $x \in \mathbb{R}^n$, $\gamma > 0$, and $\mu > 0$. The expression for the proximity operator (3.59) follows from [29, Example 4.15], reported below, and Corollary 3.1.

Proposition 3.5. *[29, Example 4.15] Let* $\alpha_1, \alpha_2 \in \mathbb{R}_{>0}$, $b_{\mathrm{m}}, b_{\mathrm{M}} \in \mathbb{R}$, *with* $b_{\mathrm{m}} < b_{\mathrm{M}}$, *and consider the function defined as*

$$(\forall \xi \in \mathbb{R}) \quad \phi(\xi) = \begin{cases} -\alpha_1 \ln(\xi - b_{\mathrm{m}}) - \alpha_2 \ln(b_{\mathrm{M}} - \xi) & \text{if } b_{\mathrm{m}} < \xi < b_{\mathrm{M}}, \\ +\infty & \text{otherwise.} \end{cases} \tag{3.65}$$

Then, $\forall \xi \in \mathbb{R}$, $z = \operatorname{prox}_{\phi}(\xi)$ *is the unique solution in* $]b_{\mathrm{m}}, b_{\mathrm{M}}[$ *of*

$$z^3 - (b_{\mathrm{m}} + b_{\mathrm{M}} + \xi)z^2 + (b_{\mathrm{m}} b_{\mathrm{M}} - \alpha_1 - \alpha_2 + (b_{\mathrm{m}} + b_{\mathrm{M}})\xi)z - b_{\mathrm{m}} b_{\mathrm{M}} \xi + \alpha_2 b_{\mathrm{m}} + \alpha_1 b_{\mathrm{M}} = 0.$$

To compute the derivatives, let F be defined as follows:

$$F(x, \mu, \gamma, z) = (b_{\mathrm{M}} - z)(b_{\mathrm{m}} - z)(z - a^T x) + \gamma\mu(b_{\mathrm{M}} + b_{\mathrm{m}} - 2z)\|a\|^2, \tag{3.66}$$

for $z \in]b_{\mathrm{m}}, b_{\mathrm{M}}[$. Expanding (3.66) gives the following:

$$\begin{aligned} F(x, \mu, \gamma, z) = {} & z^3 - (a^T x + b_{\mathrm{m}} + b_{\mathrm{M}})z^2 + (b_{\mathrm{m}} b_{\mathrm{M}} + a^T x(b_{\mathrm{m}} + b_{\mathrm{M}}) - 2\gamma\mu\|a\|^2)z \\ & - b_{\mathrm{m}} b_{\mathrm{M}} a^T x + \gamma\mu(b_{\mathrm{m}} + b_{\mathrm{M}})\|a\|^2. \end{aligned} \tag{3.67}$$

Hence, by definition of $\kappa(x, \mu, \gamma)$, we have $F(x, \mu, \gamma, \kappa(x, \mu, \gamma)) = 0$. In addition, the derivative of F with respect to its last variable is equal to

$$\nabla F^{(z)}(x, \mu, \gamma, z) = (b_{\mathrm{M}} - z)(b_{\mathrm{m}} - z) - (b_{\mathrm{m}} + b_{\mathrm{M}} - 2z)(z - a^T x) - 2\gamma\mu\|a\|^2. \tag{3.68}$$

By construction, $(b_{\mathrm{M}} - \kappa(x, \mu, \gamma))(b_{\mathrm{m}} - \kappa(x, \mu, \gamma)) < 0$. Moreover, $-2\gamma\mu\|a\|^2 < 0$ and, since $F(x, \mu, \gamma, \kappa(x, \mu, \gamma)) = 0$, it follows that $(b_{\mathrm{m}} + b_{\mathrm{M}} - 2\kappa(x, \mu, \gamma))$ and $\kappa(x, \mu, \gamma) - a^T x$ share the same sign. Hence,

$$\eta(x, \mu, \gamma) = \nabla F^{(z)}(x, \mu, \gamma, \kappa(x, \mu, \gamma)) \neq 0. \tag{3.69}$$

From the Implicit Function Theorem (A.2), we deduce that the gradient of κ with respect to x and the partial derivatives of κ with respect to μ and γ exist and are equal to

$$\nabla \kappa^{(x)}(x, \mu, \gamma) = \frac{(b_{\mathrm{M}} - \kappa(x, \mu, \gamma))(b_{\mathrm{m}} - \kappa(x, \mu, \gamma))}{\eta(x, \mu, \gamma)} a, \tag{3.70}$$

$$\nabla \kappa^{(\mu)}(x, \mu, \gamma) = \frac{-\gamma\|a\|^2(b_{\mathrm{m}} + b_{\mathrm{M}} - 2\kappa(x, \mu, \gamma))}{\eta(x, \mu, \gamma)}, \tag{3.71}$$

and

$$\nabla \kappa^{(\gamma)}(x, \mu, \gamma) = \frac{-\mu\|a\|^2(b_{\mathrm{m}} + b_{\mathrm{M}} - 2\kappa(x, \mu, \gamma))}{\eta(x, \mu, \gamma)}. \tag{3.72}$$

Differentiating (3.59) with respect to x, μ and γ and using (3.70)–(3.72) yields (3.61)–(3.63). $\square$

Note that the three roots of (3.60) can easily be computed using the Cardano formula. The graph of the resulting proximity operator is plotted on Figure 3.4 for $n = 1$, $a = 1$, $b_{\mathrm{m}} = 0$, $b_{\mathrm{M}} = 1$, and various values for $\gamma\mu$.

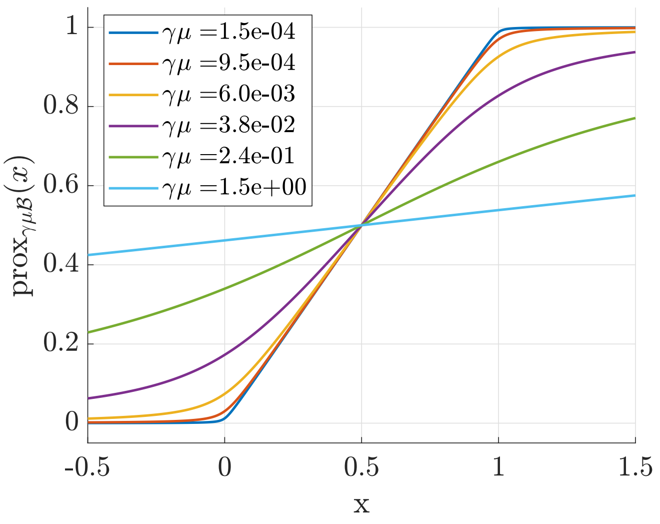


Figure 3.4: Proximity operator of the logarithmic barrier, $\mathrm{prox}_{\gamma\mu\mathcal{B}}(x)$, for hyperslab constraint with $b_{\mathrm{m}} = 0$ and $b_{\mathrm{M}} = 1$.

3.5.3 Bounded ℓ^2-norm

We now consider the case when the feasible set in (3.39) is a Euclidean ball

$$\mathcal{C} = \{x \in \mathbb{R}^n \mid \|x - c\|^2 \leq \alpha\}, \tag{3.73}$$

with $\alpha > 0$ and $c \in \mathbb{R}^n$.

Proposition 3.6. *Let $\gamma > 0$ and let $\mu > 0$. Let $\mathcal{B}$ be the barrier function associated to* (3.73)*, defined as*

$$(\forall u \in \mathbb{R}^n) \quad \mathcal{B}(u) = \begin{cases} -\ln(\alpha - \|u - c\|^2) & \text{if } \|u - c\|^2 < \alpha, \\ +\infty & \text{otherwise.} \end{cases} \tag{3.74}$$

Then, for every $x \in \mathbb{R}^n$, the proximity operator of $\gamma\mu\mathcal{B}$ at x is given by

$$\varphi(x, \mu, \gamma) = c + \frac{\alpha - \kappa(x, \mu, \gamma)^2}{\alpha - \kappa(x, \mu, \gamma)^2 + 2\gamma\mu}(x - c), \tag{3.75}$$

where $\kappa(x, \mu, \gamma)$ is the unique solution in $[0, \sqrt{\alpha}[$ of the cubic equation:

$$0 = z^3 - \|x - c\|z^2 - (\alpha + 2\gamma\mu)z + \alpha\|x - c\|. \tag{3.76}$$

In addition, the Jacobian matrix of φ with respect to x and the gradients of φ with respect to μ and γ are given by

$$J_{\varphi}^{(x)}(x, \mu, \gamma) = \frac{\alpha - \|\varphi(x, \mu, \gamma) - c\|^2}{\alpha - \|\varphi(x, \mu, \gamma) - c\|^2 + 2\gamma\mu} M(x, \mu, \gamma), \tag{3.77}$$

$$\nabla_{\varphi}^{(\mu)}(x,\mu,\gamma) = \frac{-2\gamma}{\alpha - \|\varphi(x,\mu,\gamma)-c\|^2 + 2\gamma\mu} M(x,\mu,\gamma)(\varphi(x,\mu,\gamma)-c), \tag{3.78}$$

and

$$\nabla_{\varphi}^{(\gamma)}(x,\mu,\gamma) = \frac{-2\mu}{\alpha - \|\varphi(x,\mu,\gamma)-c\|^2 + 2\gamma\mu} M(x,\mu,\gamma)(\varphi(x,\mu,\gamma)-c), \tag{3.79}$$

where

$$M(x,\mu,\gamma) = I_n - \frac{2(x-\varphi(x,\mu,\gamma))(\varphi(x,\mu,\gamma)-c)^T}{\alpha - 3\|\varphi(x,\mu,\gamma)-c\|^2 + 2\gamma\mu + 2(\varphi(x,\mu,\gamma)-c)^T(x-c)}. \tag{3.80}$$

Proof. Let $x \in \mathbb{R}^n$, $\gamma > 0$, $\mu > 0$. Let us first consider the case when $c = 0$. We denote with φ_0 the following proximity operator:

$$\varphi_0(x,\mu,\gamma) = \operatorname*{argmin}_{u \in \operatorname{int}\mathcal{C}} \frac{1}{2}\|x-u\|^2 - \gamma\mu\ln(\alpha - \|u\|^2). \tag{3.81}$$

Hence, $\|\varphi_0(x,\mu,\gamma)\|^2 < \alpha$ and $\varphi_0(x,\mu,\gamma)$ is a solution to the following equation:

$$0 = \varphi_0(x,\mu,\gamma) - x + \frac{2\gamma\mu}{\alpha - \|\varphi_0(x,\mu,\gamma)\|^2}\varphi_0(x,\mu,\gamma). \tag{3.82}$$

Since $\alpha - \|\varphi_0(x,\mu,\gamma)\|^2 + 2\gamma\mu > 0$, (3.82) becomes

$$\varphi_0(x,\mu,\gamma) = \frac{\alpha - \|\varphi_0(x,\mu,\gamma)\|^2}{\alpha - \|\varphi_0(x,\mu,\gamma)\|^2 + 2\gamma\mu} x. \tag{3.83}$$

By taking the norm in both sides of (3.83), we deduce that $\|\varphi_0(x,\mu,\gamma)\| = \kappa(x,\mu,\gamma)$ is a solution to the cubic equation (3.76). Since the proximity operator at a given x is uniquely defined, there exists only one real solution to (3.76) which belongs to $[0,\sqrt{\alpha}[$. Inserting the latter into (3.83) leads to (3.75). The analysis when $c \neq 0$ is deduced from the case $c = 0$ by using the following statement:

Proposition 3.7. *[6, Proposition 24.8 (v)] Let $f \in \Gamma_0(\mathbb{R}^n)$, $x, z \in \mathbb{R}^n$, and let $\gamma \in \mathbb{R}_{>0}$. If $g \in \Gamma_0(\mathbb{R}^n)$ is such that $g = f(\beta \cdot -c)$ with $\beta \in \mathbb{R} \setminus \{0\}$, then*

$$\operatorname{prox}_{\gamma g}(x) = \frac{1}{\beta}\left(c + \operatorname{prox}_{\gamma\beta^2 f}(\beta x - c)\right).$$

Then, the proximity operator of $\gamma\mu\mathcal{B}$ at x is given by

$$\varphi(x,\mu,\gamma) = c + \varphi_0(x-c,\mu,\gamma). \tag{3.84}$$

Let us study the derivatives of φ_0. For every $v \in \mathbb{R}^n$, let F be defined as

$$F(x,\mu,\gamma,v) = (\alpha - \|v\|^2)(v-x) + 2\gamma\mu v. \tag{3.85}$$

The Jacobian of F with respect to its last variable is equal to

$$J_F^{(v)}(x,\mu,\gamma,v) = (\alpha - \|v\|^2 + 2\gamma\mu)I_n + 2(x-v)v^T. \tag{3.86}$$

Since $\alpha - \|\varphi_0(x,\mu,\gamma)\|^2 > 0$, according to the Sherman–Morrison Lemma [5], $J_F^{(v)}(x,\mu,\gamma,\varphi_0(x,\mu,\gamma))$ is invertible if and only if

$$\alpha - \|\varphi_0(x,\mu,\gamma)\|^2 + 2\gamma\mu + 2\varphi_0(x,\mu,\gamma)^T(x - \varphi_0(x,\mu,\gamma)) \neq 0. \tag{3.87}$$

Furthermore, it follows from (3.82) that

$$F(x,\mu,\gamma,\varphi_0(x,\mu,\gamma)) = 0. \tag{3.88}$$

Applying $\varphi_0(x,\mu,\gamma)^T$ on (3.88) leads to $\varphi_0(x,\mu,\gamma)^T(x - \varphi_0(x,\mu,\gamma)) \geq 0$. In addition, $\alpha - \|\varphi_0(x,\mu,\gamma)\|^2 + 2\gamma\mu > 0$. Hence, $J_F^{(v)}(x,\mu,\gamma,\varphi_0(x,\mu,\gamma))$ is invertible and its inverse is given by the Sherman–Morrison formula:

$$\begin{aligned} J_F^{(v)}(x,\mu,\gamma,\varphi_0(x,\mu,\gamma))^{-1} = & \frac{1}{\alpha - \|\varphi_0(x,\mu,\gamma)\|^2 + 2\gamma\mu} \times \\ & \left[I_n - \frac{2(x - \varphi_0(x,\mu,\gamma))\varphi_0(x,\mu,\gamma)^T}{\alpha - 3\|\varphi_0(x,\mu,\gamma)\|^2 + 2\gamma\mu + 2\varphi_0(x,\mu,\gamma)^T x} \right]. \end{aligned} \tag{3.89}$$

From the Implicit Function Theorem (A.2) we deduce that the Jacobian of φ_0 with respect to x and the gradients of φ_0 with respect to μ and γ exist and are equal to

$$J_{\varphi_0}^{(x)}(x,\mu,\gamma) = -J_F^{(v)}(x,\mu,\gamma,\varphi_0(x,\mu,\gamma))^{-1} J_F^{(x)}(x,\mu,\gamma,\varphi_0(x,\mu,\gamma)), \tag{3.90}$$

$$\nabla_{\varphi_0}^{(\mu)}(x,\mu,\gamma) = -J_F^{(v)}(x,\mu,\gamma,\varphi_0(x,\mu,\gamma))^{-1} \nabla_F^{(\mu)}(x,\mu,\gamma,\varphi_0(x,\mu,\gamma)), \tag{3.91}$$

and

$$\nabla_{\varphi_0}^{(\gamma)}(x,\mu,\gamma) = -J_F^{(v)}(x,\mu,\gamma,\varphi_0(x,\mu,\gamma))^{-1} \nabla_F^{(\gamma)}(x,\mu,\gamma,\varphi_0(x,\mu,\gamma)). \tag{3.92}$$

When $c \neq 0$, the derivatives of φ are deduced from those of φ_0 using (3.84):

$$J_{\varphi}^{(x)}(x,\mu,\gamma) = -J_F^{(v)}(x-c,\mu,\gamma,\varphi(x,\mu,\gamma)-c)^{-1} J_F^{(x)}(x-c,\mu,\gamma,\varphi(x,\mu,\gamma)-c), \tag{3.93}$$

$$\nabla_{\varphi}^{(\mu)}(x,\mu,\gamma) = -J_F^{(v)}(x-c,\mu,\gamma,\varphi(x,\mu,\gamma)-c)^{-1} \nabla_F^{(\mu)}(x-c,\mu,\gamma,\varphi(x,\mu,\gamma)-c), \tag{3.94}$$

and

$$\nabla_{\varphi}^{(\gamma)}(x,\mu,\gamma) = -J_F^{(v)}(x-c,\mu,\gamma,\varphi(x,\mu,\gamma)-c)^{-1} \nabla_F^{(\gamma)}(x-c,\mu,\gamma,\varphi(x,\mu,\gamma)-c), \tag{3.95}$$

which lead to (3.77)-(3.79). □

Similarly to the previous case, the three solutions to (3.76) can be obtained by using the Cardano formula. The form of the resulting proximity operator for $n = 2$ is plotted on Figure 3.5 (right) for $\alpha = 0.7$, $c = 0$, and several values of $\gamma\mu$ and x; for symmetry reasons, only the first component $\left(\text{prox}_{\gamma\mu\mathcal{B}}(x)\right)_1$ is represented.

As shown in this section, the proximity operator of the barrier is easily computable and differentiable for several classic types of constraints. In particular, some of these results are useful for dealing with the specific constraints we considered in Chapter 4.

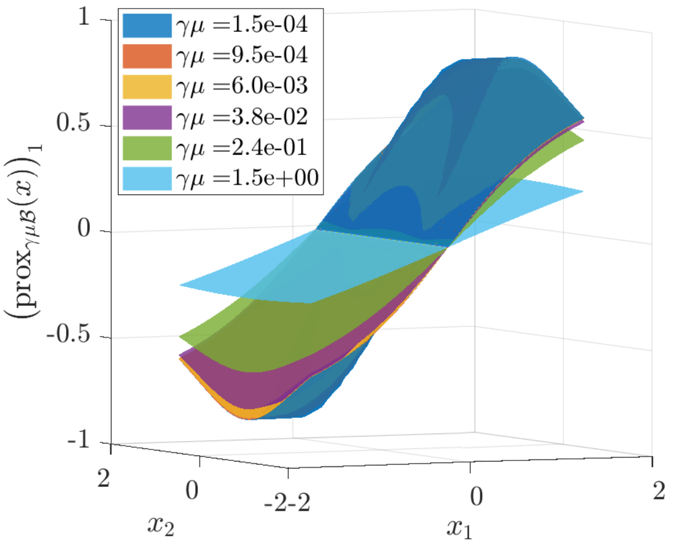


Figure 3.5: First component of the proximity operator of the logarithmic barrier $\left(\mathrm{prox}_{\gamma\mu\mathcal{B}}(x)\right)_1$ for a constraint on the ℓ^2-norm with $\alpha = 0.7$ and $c = 0$.

Chapter 4

A Neural Network Based on a Proximal IPM for Image Restoration

4.1 Unfolding Methods

As detailed in Section 3.4.1, we focus on the particular image reconstruction problem arising from the discrete model (3.12). In this context, both variational and Deep Learning approaches provide efficient methods for delivering an estimate of $\bar{x}$, while offering different benefits and drawbacks, which are discussed hereafter.
In order to find an appropriate reconstruction, we have seen in the previous chapter that variational methods incorporate prior information on the sought image $\bar{x}$, through constraints or regularization functions, such as the total variation and its various extensions [4]. This leads to the constrained minimization problem (3.39), where $\bar{x}$ is approximated by the minimizer of a penalized cost function expressed as the sum of a data-fitting term f, and a regularization term $\mathcal{R}$.
Although useful, this approach is sometimes limited by its complexity: solving (3.39) requires iterative algorithms, like the one proposed in Section 3.4, that may be too slow for real-time applications, especially if we consider the time required for parameters estimation. For example, $\mathcal{R}$ is usually parametrized by one or several parameters, like λ, whose optimal choice may strongly depend on the data at hand. In Section 3.4.4 we mentioned these limitations. The parameters are often tuned manually or computed using, for instance, cross validation, the discrepancy principle [121], or methods based on Stein unbiased risk estimates (SURE) [44]. However, these methods are often time-consuming and their success is not always guaranteed. Furthermore, despite numerous efforts in designing sophisticated models, the solution to (3.39) could be further away from $\bar{x}$, than an intermediate iterate of the iterative algorithm. Therefore, a reliable early stopping procedure is needed to stop the method at the right iterate.

Finding the optimal stopping time depends on the algorithm and usually requires the use of an oracle such as SURE, which may explain why these techniques are currently restricted to relatively simple cost functions.

A more recent trend in Imaging is Deep Learning. In the last few years, Deep Neural Networks (DNNs), and in particular Convolutional Neural Networks (CNNs), provided good performance for various applications related to inverse problems, such as denoising [149], non-blind and blind deblurring [124, 125, 144], super-resolution [92], or CT reconstruction [25, 76]. As detailed in [98], DNNs for inverse problems are very often preceded by a pre-processing step. Indeed, a rough estimation of $\bar{x}$ can be found by using the inverse or pseudoinverse of H. The latter tends, however, to strongly amplify noise. Hence, in this context, DNNs are used as denoisers and artifact-removers. However, since prior knowledge about its output can hardly be incorporated into a DNN, which in most of the cases is seen as a black-box, the explainability and reliability of such methods could be questioned [131]. Furthermore, the pre-processing step, in itself, can include a penalty, thus amounting to solving a problem of the form (3.39), where the regularization weight strongly depends on the noise level, see, *e.g.* [22, 124].

One straightforward way to combine the benefits of both variational-based methods and DNNs is to unfold an iterative method, turning each iteration into a layer of a network, and to untie the parameters of both the model and the algorithm across the network layers [68]. The underlying idea is that, nearly all state-of-the-art iterative reconstruction algorithms alternate between linear steps and pointwise nonlinear steps. So it follows that neural networks should be able to perform similarly well, given appropriate training [98]. Interestingly, the fact that this approach makes use of a limited number of layers can be viewed as an analogue of stopping procedure. It is however worth mentioning that, in unfolded algorithms, the number of iterations, i.e., layers, is tuned during the off-line training step and is then fixed for all test images. This differs from stopping strategies where the iteration number usually differs for each processed image.
We therefore propose a novel neural network architecture called iRestNet [13, 37], which is obtained by unfolding the proximal interior point algorithm presented in Algorithm 14, over a finite number of iterations. One key feature of this algorithm is that it produces only feasible iterates thanks to the logarithmic barrier. This barrier enables prior knowledge to be directly incorporated into iRestNet and, as opposed to a projection onto $\mathcal{C}$, it allows differentiation and gradient backpropagation throughout the network. Hence, gradient descent can be used for training. The stepsize, barrier parameter, and regularization weight are untied across the network and learned for each layer. Thus, once the network has been trained, its application on test images requires only a short execution time per image without any parameter search, as opposed to traditional variational methods.
Several recent works consider replacing handcrafted algorithms by learned iterative methods [3, 94]. However, only a few works so far have considered combining interior point methods

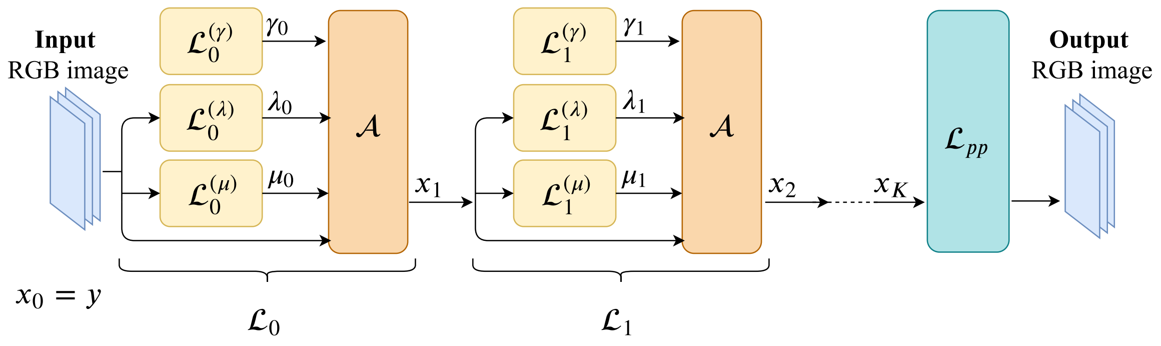


Figure 4.1: iRestNet global architecture.

(IPMs) with Deep Learning [2, 134]. In these approaches, the goal is to find the minimizer of the given objective function. Conversely, in our case we consider two distinct objective functions. The first one, $h(x, y, \lambda)$, leads to the constrained problem from which the proposed architecture is inspired. However, a better indicator of perceptual quality (SSIM) is optimized during the training step, as we will see in Section 4.4.4. It is worth noticing that the output of the trained network is not necessarily a minimizer of the first objective. Moreover, the second objective could not be substituted to the first one since it requires the knowledge of the ground-truth, which is available for training time but not in testing conditions. In addition, iRestNet appears to have more flexibility since the regularization weight can vary among layers.

To the best of our knowledge, iRestNet is the first architecture corresponding to a deep unfolded version of an interior point algorithm with untied stepsize and regularization parameter. As opposed to other unfolding methods like [45, 97], the proximity operator and the regularization term are kept explicit, which establishes a direct relation between the original algorithm and the network.

4.2 iRestNet Architecture

Our proposal is to adopt a supervised learning strategy in order to determine, from a training set of images, an optimal set of parameters for Algorithm 14. This should lead to an optimal image reconstruction quality also on test images. To this aim, Algorithm 14 is unfolded over K iterations and the regularization parameter λ is untied across the network, so as to provide more flexibility to the approach [68]. The update rule at a given iteration $k \in \{0, \ldots, K-1\}$ reads

$$x_{k+1} = \mathcal{A}\left(x_k, \mu_k, \gamma_k, \lambda_k\right) \tag{4.1}$$

with

$$\mathcal{A}\left(x_k, \mu_k, \gamma_k, \lambda_k\right) = \operatorname{prox}_{\gamma_k \mu_k \mathcal{B}}\left(x_k - \gamma_k \nabla_1 h\left(x_k, y, \lambda_k\right)\right). \tag{4.2}$$

For every $k \in \{0, \ldots, K-1\}$, we build the k-th layer $\mathcal{L}_k$ as the association of three hidden

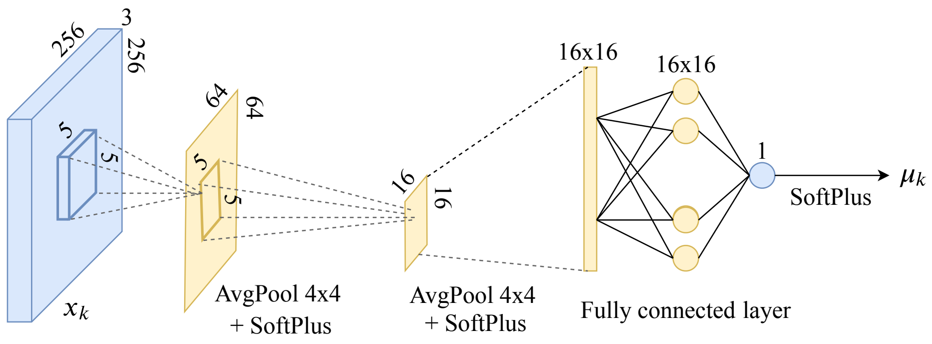


Figure 4.2: Architecture of $\mathcal{L}_k^{(\mu)}$.

structures, $\mathcal{L}_k^{(\mu)}$, $\mathcal{L}_k^{(\gamma)}$ and $\mathcal{L}_k^{(\lambda)}$, followed by the update $\mathcal{A}$. Structures $\mathcal{L}_k^{(\mu)}$, $\mathcal{L}_k^{(\gamma)}$, and $\mathcal{L}_k^{(\lambda)}$ aim at inferring the barrier parameter μ_k, the stepsize γ_k and the regularization weight λ_k, respectively. Since a finite number K of layers, i.e., updates, is used, the convergence of the resulting scheme is not an issue. Note that we also allow in our framework the use of a post–processing step after going through the K layers. This will be denoted as $\mathcal{L}_{\text{pp}}$. The resulting architecture is depicted in Figure 4.1.

4.2.1 Hidden Structures

Let us now provide more details about the hidden structures. For every $k \in \{0, \dots, K-1\}$, the outputs $(\mu_k, \gamma_k, \lambda_k)$ of the structures $\mathcal{L}_k^{(\mu)}$, $\mathcal{L}_k^{(\gamma)}$, and $\mathcal{L}_k^{(\lambda)}$ must be positive. To enforce such constraint, we use Softplus activation functions, defined in (2.12) as

$$(\forall z \in \mathbb{R}) \quad \text{Softplus}(z) = \ln(1 + \exp(z)). \tag{4.3}$$

This is a smooth approximation of the ReLU activation function. However, unlike ReLU, the gradient of Softplus is never strictly equal to zero, which, given our architecture, helps to propagate the gradient through the network.
In particular, the stepsize is estimated as follows,

$$\gamma_k = \mathcal{L}_k^{(\gamma)} = \text{Softplus}\left(a_k\right), \tag{4.4}$$

where a_k is an internal scalar parameter of the network, learned during training. The barrier parameter μ_k is obtained using a sequence of two convolutional and average pooling layers followed by a fully connected layer. The detailed architecture $\mathcal{L}_k^{(\mu)}$ is depicted in Figure 4.2.
Finally, traditional methods for estimating the regularization parameter generally depend on the signal-to-noise ratio and on the image statistics [138]. For most applications the noise level is unknown and can be estimated, for instance, by applying a median filter over the wavelet diagonal coefficients of the image [112]. This is the strategy we use in numerical experiments,

in Section 4.4. The advantage is to yield a network which can handle datasets for which the signal-to-noise ratio is unknown and can vary within a reasonable range. The expression of $\mathcal{L}_k^{(\lambda)}$ is then problem–dependent since its expression depends on the regularization function $\mathcal{R}$. A specific example is given in Section 4.4, for the total variation regularization function.
Regarding the post-processing step $\mathcal{L}_{\text{pp}}$, its detailed architecture also depends on the task to be performed. An example is provided in Section 4.4 for the deblurring case: the purpose of $\mathcal{L}_{\text{pp}}$ is then to remove remaining artifacts, by using a particular convolutional neural network.

4.2.2 Differential Calculus

To train the neural network presented in Figure 4.1 using gradient descent, one needs to compute the gradient of x_K with respect to the different internal parameters of the network. The chain rule can be applied since most of the steps in the network correspond to operators having straightforward derivatives. However, particular care should be taken when differentiating $\mathcal{A}$. Since f and $\mathcal{R}$ are assumed to be twice differentiable, the only area of concern is related to $\text{prox}_{\gamma\mu\mathcal{B}}$. If $\text{prox}_{\gamma\mu\mathcal{B}}$ is simple enough, automatic differentiation can be used. Otherwise, as shown in Section 3.5, the differential of this term is well-defined for common examples of barrier functions. In particular, we have seen in Propositions 3.1, 3.4, 3.6, the corresponding expressions for the derivatives of three different barrier functions.

4.3 Network Stability

One critical issue concerning neural networks is to guarantee their stability with respect to small perturbations to their inputs, so that their performance remains acceptable when the input is perturbed. For example, the authors of [131] show that the class prediction made by AlexNet can be arbitrarily changed by using small nonrandom perturbations on the test image. For some applications involving high risk and legal responsibility, for instance in medical image processing, the lack of theoretical guarantees is a significant curb on the utilization of deep learning approaches. A recent work [35] provides a theoretical framework which enables to evaluate the robustness of a network. We recall that, in order for a neural network to be considered robust to noise, performance must be stable after adding some noise to the dataset. In this section, we will focus on a subclass of problem (3.39), where both $f(\cdot, y)$ and $\mathcal{R}$ are quadratic functions, while $\mathcal{C}$ is defined as in (3.42). After highlighting the similarities between the proposed architecture and generic feedforward networks in that case, we will give explicit conditions under which the robustness of the proposed architecture is ensured.

4.3.1 Relation to Generic Deep Neural Networks

Although the proposed architecture may seem specific to Algorithm 14, it is actually very similar to generic feedforward neural networks. Classical feedforward (acyclic) architectures

[122] can be expressed as $R_{K-1} \circ (W_{K-1} \cdot + b_{K-1}) \circ \cdots \circ R_0 \circ (W_0 \cdot + b_0)$, where $(R_k)_{0 \le k \le K-1}$ are nonlinear activation functions, $(W_k)_{0 \le k \le K-1}$ are weight operators and $(b_k)_{0 \le k \le K-1}$ are bias parameters. Let us show that iRestNet actually shares a similar structure. For the sake of simplicity, we will consider the variational problem,

$$\underset{x \in \mathcal{C}}{\text{minimize}} \; \frac{1}{2}\|Hx - y\|^2 + \frac{\lambda}{2}\|Dx\|^2, \tag{4.5}$$

where $y \in \mathbb{R}^n$, $H \in \mathbb{R}^{n \times n}$, $D \in \mathbb{R}^{n \times n}$, and $\mathcal{C} = \{x \in \mathbb{R}^n \mid c_j(x) \ge 0, \ \forall j = 1, \ldots, p\}$ is defined by p inequality constraints as in (3.42). Moreover, we assume that no post–processing layer $\mathcal{L}_{\text{pp}}$ is used. Following the notation of Section 4.2, $(\forall k \in \{0, \ldots, K-1\})$ $(\mu_k, \gamma_k, \lambda_k)$ are given positive real numbers, K being the number of layers of the network. Then, for every $k \in \{0, \ldots, K-1\}$, layer $\mathcal{L}_k$ corresponds to the following update,

$$\begin{aligned} x_{k+1} &= \operatorname{prox}_{\gamma_k \mu_k \mathcal{B}} \left(x_k - \gamma_k \left(H^T (Hx_k - y) + \lambda_k D^T D x_k \right) \right) \\ &= \operatorname{prox}_{\gamma_k \mu_k \mathcal{B}} \left(\left[I_n - \gamma_k \left(H^T H + \lambda_k D^T D \right) \right] x_k + \gamma_k H^T y \right), \end{aligned} \tag{4.6}$$

where $\mathcal{B} = -\sum_{i=1}^{p} \ln(c_i(x))$ if $x \in \operatorname{int}\mathcal{C}$, $+\infty$ otherwise, as defined in (3.45).
For every $k \in \{0, \ldots, K-1\}$, we set

$$W_k = I_n - \gamma_k \left(H^T H + \lambda_k D^T D \right), \quad b_k = \gamma_k H^T y, \quad \text{and} \quad R_k = \operatorname{prox}_{\gamma_k \mu_k \mathcal{B}}. \tag{4.7}$$

Then, the K-layer network $\mathcal{L}_{K-1} \circ \cdots \circ \mathcal{L}_0$ is equivalent to $R_{K-1} \circ (W_{K-1} \cdot + b_{K-1}) \circ \cdots \circ R_0 \circ (W_0 \cdot + b_0)$, where $(W_k)_{0 \le k \le K-1}$ and $(b_k)_{0 \le k \le K-1}$ are interpreted as weight operators and bias parameters, respectively. It is worth noticing that the operators $(R_k)_{0 \le k \le K-1}$ defined in (4.7) can be viewed as specific activation functions since, as shown in [35], every standard activation function can be derived from a proximity operator. In addition, using [6, Proposition 24.8(iii)], for every $k \in \{0, \ldots, K-1\}$, R_k can be re-written as the sum of a *proximal activation operator* [35, Definition 2.20] and a bias.

4.3.2 Preliminary Results

Before stating our main stability theorem, we recall the result from [35, Lemma 3.3] in Proposition 4.1 below. We then derive Proposition 4.2, which will appear useful when addressing the robustness of the global network. In the following, $\mathcal{S}_n$ denotes the set of symmetric matrices in $\mathbb{R}^{n \times n}$ and, for every $W \in \mathbb{R}^{n \times n}$, $\|W\|$ denotes its spectral norm.

Proposition 4.1. [35] *Let $K \ge 1$ be an integer and set $\theta_{-1} = 1$. For every $k \in \{0, \ldots, K-1\}$, let $W_k \in \mathbb{R}^{n \times n}$ and let θ_k be defined by*

$$\begin{aligned} \theta_k = \|W_k \cdot \cdots \cdot W_0\| + \sum_{\ell=0}^{k-1} \Big\{ \sum_{0 \le j_0 < \cdots < j_\ell \le k-1} & \|W_k \cdot \cdots \cdot W_{j_\ell + 1}\| \cdot \\ & \|W_{j_\ell} \cdot \cdots \cdot W_{j_{\ell-1}+1}\| \cdots \|W_{j_0} \cdot \cdots \cdot W_0\| \Big\}. \end{aligned} \tag{4.8}$$

Then, for every $k \in \{0, \ldots, K-1\}$, $\theta_k = \sum_{\ell=0}^{k} \theta_{\ell-1} \|W_k \cdot \cdots \cdot W_\ell\|$.

Proposition 4.2. *Let $K \geq 1$, $\theta > 0$, and $\alpha \in [1/2, 1]$. Let $W \in \mathcal{S}_n$ and let β_- and β_+ denote the smallest and largest eigenvalues of W, respectively. Then, the condition*

$$\|W - 2^K(1-\alpha)I_n\| - \|W\| + 2\theta \leq 2^K\alpha \tag{4.9}$$

is satisfied if and only if one of the following conditions holds:

1. $\beta_+ + \beta_- \leq 0$ *and* $\theta \leq 2^{K-1}(2\alpha - 1)$*;*

2. $0 \leq \beta_+ + \beta_- \leq 2^{K+1}(1-\alpha)$ *and* $2\theta \leq \beta_+ + \beta_- + 2^K(2\alpha - 1)$*;*

3. $2^{K+1}(1-\alpha) \leq \beta_+ + \beta_-$ *and* $\theta \leq 2^{K-1}$.

Proof. Let $\alpha \in [1/2, 1]$. Since $W \in \mathcal{S}_n$, we have,

$$\|W\| = \max\{\beta_+, -\beta_-\}, \tag{4.10}$$

and

$$\|W - 2^K(1-\alpha)I_n\| = \max\left\{\beta_+ - 2^K(1-\alpha), -\beta_- + 2^K(1-\alpha)\right\}. \tag{4.11}$$

Three different cases arise that we review below.

(i) If $\beta_+ + \beta_- \leq 0$ then $\|W\| = -\beta_-$ and

$$\beta_+ - 2^K(1-\alpha) \leq -\beta_- + 2^K(1-\alpha). \tag{4.12}$$

From (4.11) and (4.12), we deduce that $\|W - 2^K(1-\alpha)I_n\| = -\beta_- + 2^K(1-\alpha)$. Replacing $\|W\|$ and $\|W - 2^K(1-\alpha)I_n\|$ by their value in (4.9) leads to Proposition 4.2(1).

(ii) If $0 \leq \beta_+ + \beta_- \leq 2^{K+1}(1-\alpha)$ then $\|W\| = \beta_+$ and (4.12) is satisfied. Hence, $\|W - 2^K(1-\alpha)I_n\| = -\beta_- + 2^K(1-\alpha)$. Replacing $\|W\|$ and $\|W - 2^K(1-\alpha)I_n\|$ by their value in (4.9) leads to Proposition 4.2(2).

(iii) If $2^{K+1}(1-\alpha) \leq \beta_+ + \beta_-$ then $\|W\| = \beta_+$ and

$$\beta_+ - 2^K(1-\alpha) \geq -\beta_- + 2^K(1-\alpha). \tag{4.13}$$

From (4.11) and (4.13), we deduce that $\|W - 2^K(1-\alpha)I_n\| = \beta_+ - 2^K(1-\alpha)$. Replacing $\|W\|$ and $\|W - 2^K(1-\alpha)I_n\|$ by their value in (4.9) leads to Proposition 4.2(3), which completes the proof.

□

4.3.3 Averaged Operator

The notion of nonexpansiveness, whose definition is recalled below, plays a central role in the analysis of the robustness of nonlinear operators. Indeed, it indicates that a perturbation on the output of a given operator is bounded by the amplitude of the input perturbation.

Definition 4.1 (Nonexpansiveness). *Let $T : \mathbb{R}^n \to \mathbb{R}^n$. Then, T is nonexpansive if it is Lipschitz continuous with constant 1, i.e.,*

$$(\forall x \in \mathbb{R}^n)(\forall y \in \mathbb{R}^n) \quad \|T(x) - T(y)\| \leq \|x - y\|. \tag{4.14}$$

In the present study we make use of the notion of averaged operator [6], which is stronger than nonexpansiveness.

Definition 4.2 (α-averaged operator). *Let $T : \mathbb{R}^n \to \mathbb{R}^n$ be nonexpansive, and let $\alpha \in [0, 1]$. Then T is averaged with constant α, or α–averaged, if there exists a nonexpansive operator $R : \mathbb{R}^n \to \mathbb{R}^n$ such that $T = (1 - \alpha)\,\mathrm{Id}_n + \alpha R$, where Id_n denotes the identity operator of $\mathbb{R}^n$.*

The following property provides an upper bound of the effect of an input perturbation, which depends on the averageness constant α. In particular, the smaller the α, the more stable the operator.

Proposition 4.3. [6, Remark 4.34, Proposition 4.35] *Let $T : \mathbb{R}^n \to \mathbb{R}^n$.*

(i) If T is averaged, then it is nonexpansive.

(ii) Let $\alpha \in \,]0, 1]$. T is α–averaged if and only if for every $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^n$,

$$\|T(x) - T(y)\|^2 \leq \|x - y\|^2 - \frac{1 - \alpha}{\alpha} \|(\mathrm{Id}_n - T)(x) - (\mathrm{Id}_n - T)(y)\|^2. \tag{4.15}$$

4.3.4 Robustness of iRestNet to an Input Perturbation

Let us consider problem (4.5), where we assume additionally that $H^T H$ and $D^T D$ are diagonalizable in a same basis denoted $\mathcal{P}$. The latter is satisfied for instance if H and D are the results of cyclic convolutive operators. Theorem 4.1 below gives sufficient conditions under which the proposed network applied to problem (4.5) is an averaged operator.

Theorem 4.1. *Let $\alpha \in [1/2, 1]$, $(W_k, b_k, R_k)_{0 \leq k \leq K-1}$ be defined by (4.7), and $(\theta_k)_{-1 \leq k \leq K-1}$ be defined as in Proposition 4.1. Let β_- and β_+ be the smallest and largest eigenvalues of $W = W_{K-1} \cdot \cdots \cdot W_0$, respectively. For every $p \in \{1, \ldots, n\}$ and every $k \in \{0, \ldots, K-1\}$, let $\beta_k^{(p)} = 1 - \gamma_k \left(\beta_H^{(p)} + \lambda_k \beta_D^{(p)}\right)$, where $\beta_H^{(p)}$ and $\beta_D^{(p)}$ denote the p^{th} eigenvalue of $H^T H$ and $D^T D$ in $\mathcal{P}$, respectively. Then, β_-, β_+, and $(\forall k \in \{0, \ldots, K-1\})$ θ_k can be computed as follows:*

$$\beta_- = \min_{1\le p\le n} \prod_{k=0}^{K-1} \beta_k^{(p)}, \ \beta_+ = \max_{1\le p\le n} \prod_{k=0}^{K-1} \beta_k^{(p)} \ \textit{and} \ \theta_k = \sum_{\ell=0}^{k} \theta_{\ell-1} \max_{1\le q_\ell\le n} \left|\beta_k^{(q_\ell)} \cdots \beta_\ell^{(q_\ell)}\right|, \quad (4.16)$$

with $q_\ell \in \{1,\ldots,n\}$*. In addition, if one of the following conditions is satisfied*

(i) $\beta_+ + \beta_- \le 0$ *and* $\theta_{K-1} \le 2^{K-1}(2\alpha-1)$*;*

(ii) $0 \le \beta_+ + \beta_- \le 2^{K+1}(1-\alpha)$ *and* $2\theta_{K-1} \le \beta_+ + \beta_- + 2^K(2\alpha-1)$*;*

(iii) $2^{K+1}(1-\alpha) \le \beta_+ + \beta_-$ *and* $\theta_{K-1} \le 2^{K-1}$*,*

then the operator $R_{K-1} \circ (W_{K-1}\cdot + b_{K-1}) \circ \cdots \circ R_0 \circ (W_0 \cdot + b_0)$ *is* α*–averaged.*

Proof. If $H^T H$ and $D^T D$ are diagonalizable in the same basis then $W \in \mathcal{S}_n$, which, combined with Proposition 4.1, leads to (4.16). If one of the conditions (i)–(iii) is satisfied, then we deduce from Proposition 4.2 that W satisfies [35, Proposition 3.6(iii)], and [35, Condition 3.1]. In addition, for every $k \in \{0,\ldots,K-1\}$, $R_k(\cdot + b_k)$ is firmly nonexpansive [6, Proposition 12.28]. Finally, [35, Theorem 3.8] completes the proof. □

The conditions provided by Theorem 4.1 can be easily checked using (4.16). Theorem 4.1 provides a framework under which iRestNet is robust to a perturbation of its input: the upper bound of the output perturbation can then be derived from Proposition 4.3.

4.4 Numerical Experiments

In this section, we present numerical experiments on a set of problems of image restoration, demonstrating that in many cases the proposed approach yields a better reconstruction quality than standard variational and machine learning methods.

4.4.1 Problem Formulation for Gaussian Noise

We treat the non-blind image deblurring problem, where the images to be reconstructed are natural color images, and where additive white Gaussian noise is added. In this case, we consider the degradation model defined in (3.15),

$$y = H\bar{x} + \omega, \quad (4.17)$$

where n is the number of pixels, $y = (y^{(j)})_{1\le j\le 3} \in \mathbb{R}^{3n}$ is the blurred RGB image, $\bar{x} = (\bar{x}^{(j)})_{1\le j\le 3} \in \mathbb{R}^{3n}$ is the ground-truth, $H \in \mathbb{R}^{3n\times 3n}$ is a linear operator that models the circular convolution of a known blur kernel with each channel of the color image, and $\omega \in \mathbb{R}^{3n}$ is a realization of an additive white Gaussian noise with standard deviation σ.

As we saw in Section 3.3, an estimate of $\bar{x}$ can be derived from the following penalized formulation,

$$\underset{x\in\mathcal{C}}{\text{minimize}} \quad \frac{1}{2}\|Hx-y\|^2 + \lambda \sum_{i=1}^{3n} \sqrt{\frac{(D_{\mathrm{v}}x)_i^2 + (D_{\mathrm{h}}x)_i^2}{\delta^2} + 1}, \tag{4.18}$$

which includes a least–squares data–fidelity term, from (3.23), and a smoothed total variation regularization, as defined in (3.35). In particular, the data–fidelity function directly derives from applying a MAP–approach in the case of Gaussian noise. Also, we saw in Section 3.3.3, that in contrast with the Tikhonov regularizers, TV functions preserve discontinuities and edges, that are naturally present in digital natural images. In the above formulation (4.18), the feasible set $\mathcal{C}$ is the hypercube $[x_{\min}, x_{\max}]^{3n}$, where $x_{\min}$ and $x_{\max}$ are a lower and an upper bound on the pixel intensity, respectively, $D_{\mathrm{v}} \in \mathbb{R}^{3n\times 3n}$ and $D_{\mathrm{h}} \in \mathbb{R}^{3n\times 3n}$ are the vertical and horizontal gradient operators, respectively, $\delta > 0$ is a smoothing parameter and $\lambda > 0$ is the regularization parameter. Here, $x_{\min} = 0$, $x_{\max} = 1$ and we set $\delta = 0.01$ in all experiments. To find this δ, we solved Problem (4.18) for a small set of images of the database and used the simplex method to find the values for δ and λ that lead to the best image quality. It is worth noticing that, from these values we deduced 0.01 as an appropriate order of magnitude for δ, but we did not perform fine-tuning on this parameter. Nevertheless, the proposed architecture can be easily modified to include the inference of δ too. The update $\mathcal{A}$, defined in (4.2), is derived from (4.18), and is unfolded over K iterations, as it is described in Section 4.2. The bound constraints in problem (4.18) fall under the framework studied in Section 3.5.2, which provides us with the expression for the proximity operator of the barrier and its gradient.

4.4.2 Network Characteristics

The tuning of the number of unfolded iterations K must achieve a compromise between training time, memory requirement, and performance. In order to determine a suitable K, we followed a structural stabilization strategy, introduced in Section 2.2. So, we trained networks with different numbers of layers, gradually increasing K until the performance of the network did not improve significantly. Using this procedure, the depth of iRestNet is taken equal to $K = 40$. Regarding the hidden structures $(\mathcal{L}_k^{(\lambda)})_{0\le k\le K-1}$, which estimate the regularization parameter, they are chosen in view of the regularization function used in problem (4.18) and have the following expression,

$$(\forall k \in \{0,\dots,K-1\}) \ \ \lambda_k = \mathcal{L}_k^{(\lambda)}(x_k) = \frac{\mathrm{Softplus}\,(b_k)\,\widehat{\sigma}(y)}{\eta(x_k) + \mathrm{Softplus}\,(c_k)}. \tag{4.19}$$

Here (b_k, c_k) is a pair of scalars learned by the network, $\eta(x_k)$ is the standard deviation of the concatenated spatial gradients of x_k, $[(D_{\mathrm{v}}x_k)^T (D_{\mathrm{h}}x_k)^T]$, and $\widehat{\sigma}(y)$ is an approximation of the noise level in the observed image, estimated as in [96, Section 11.3.1]

$$\widehat{\sigma}(y) = \mathrm{median}(|W_{\mathrm{H}}y|)/0.6745, \tag{4.20}$$

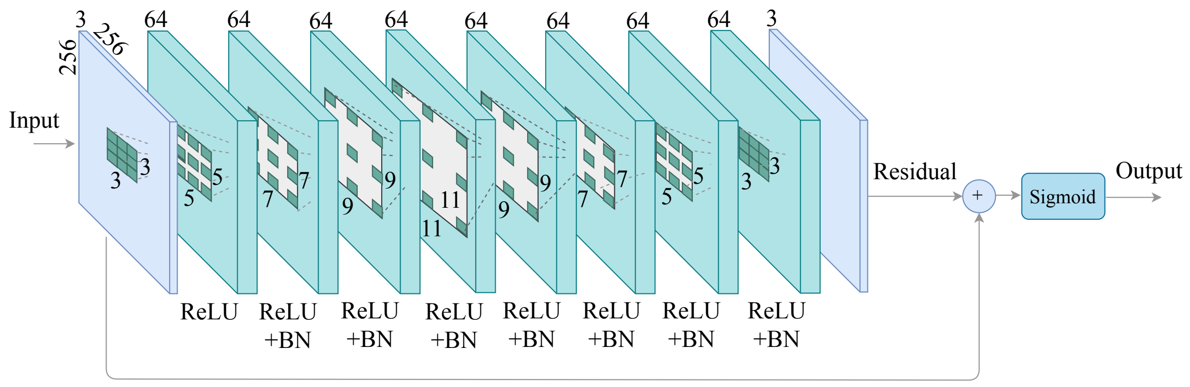


Figure 4.3: Architecture of $\mathcal{L}_{\mathrm{pp}}$. BN: batch normalization.

where $|W_{\mathrm{H}}y|$ is the vector gathering the absolute value of the diagonal coefficients of the first level Haar wavelet decomposition of y. It is worth noticing that, by computing this approximation, the proposed architecture does not require any prior knowledge about the noise level of the input image. In particular, the noise standard deviation does not have to be the same for all input images, so that iRestNet can be applied to a dataset of images corrupted with different levels of noise.

The architecture of the post-processing layer $\mathcal{L}_{\mathrm{pp}}$, which can be found in Figure 4.3, is inspired from [150]. This structure is made of 9 convolutional layers with filters of spatial size 3×3. A ReLU activation function is used after each convolution, and the final activation function is chosen as the Sigmoid function, in order to preserve the feasibility of the final output. It is to be noted that there is no pooling layer, since we want the final output to have the same size as the input. As we can see in Figure 4.3, dilation factors are used in the convolutional layers, so as to widen the receptive field without creating memory issues. In particular, increasing the receptive field is useful because, in this way, we use the context information of a larger region, and it has been widely acknowledged that the context information facilitates the reconstruction of a corrupted pixel [150]. In our case, a dilated filter of size 3×3, with dilation factor s can be interpreted as a sparse filter of size $(2s+1) \times (2s+1)$ where only 9 entries of fixed positions can be non-zeros. In this way, dilated convolutions are able to expand the receptive field while keeping the merits of traditional 3×3 convolutions. Another characteristic of the post-processing layer $\mathcal{L}_{\mathrm{pp}}$ derives from its purpose of removing remaining artifacts. As illustrated in Figure 4.4.2, the artifacts that remain in the image after going through the K blocks of iRestNet, are identified by the residual, i.e., by the difference between the ground-truth image and the deconvolved image x_K, in output to the K^{th}–layer of iRestNet. When the mapping is similar to an identity mapping (in our case x_K is similar to the groundtruth), it is easier for the network to learn the residual mapping instead of the original one. In fact, it is easier to train the network to push the residual to zero, then to fit an identity mapping by a stack of nonlinear layers [67, 149, 150]. Therefore, we add a skip

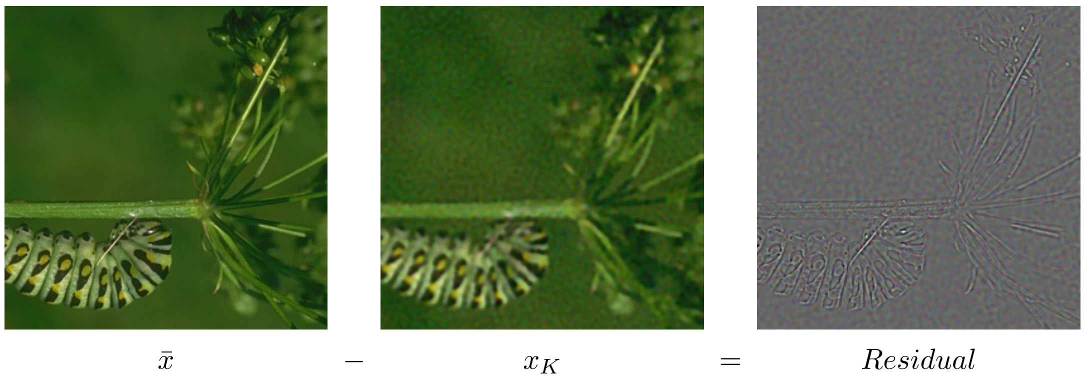


Figure 4.4: Remaining artifacts given by the difference between a ground-truth image and x_K obtained with the network trained for the experimental GaussianB configuration.

connection between the input of $\mathcal{L}_{\text{pp}}$ and its output, to apply a residual learning strategy. This approach is also useful to mitigate the vanishing gradient problem during training, because skip connections make it easier for the gradient to flow from the output layer to layers nearer the input, yielding to a noticeable increase in performance [76]. Finally, residual learning is combined with batch normalization (BN), a technique proposed in [74], which is widely used in Deep Learning to accelerate and stabilize the training process. In neural networks, batch normalization is achieved through a normalization step that fixes the means and variances of each layer's inputs. This normalization is conducted over the images of each mini-batch in the training process. In this way, the range in which the hidden unit values shift around is reduced, i.e., it alleviates the internal covariate shift [74]. The combination of residual learning and batch normalization leads to fast training, improved performance, and low sensitivity to initialization [150].

4.4.3 Dataset and Experimental Settings

The training set is made of 1200 RGB images: 200 images stem from the Berkeley segmentation (BSD500) training set[1], while the remaining 1000 images are taken from the COCO training set[2]. We use the BSD500 validation set, which is made of 100 images, to monitor the training and check if there is overfitting. The performance of the proposed method is evaluated on two different test sets: the BSD500 test set, which is made of 200 RGB images, and the Flickr30 test set used in [144], which is made of 30 RGB images. The test images have been center-cropped using a window of size 256×256. Blurry images are produced using the following 25×25 blur kernels and noise levels:

- A Gaussian kernel, which models the long-time average effects of atmospheric turbulence [9, 138], with a standard deviation of 1.6 pixels, and a Gaussian noise standard deviation

[1] `https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/`
[2] `http://cocodataset.org/`

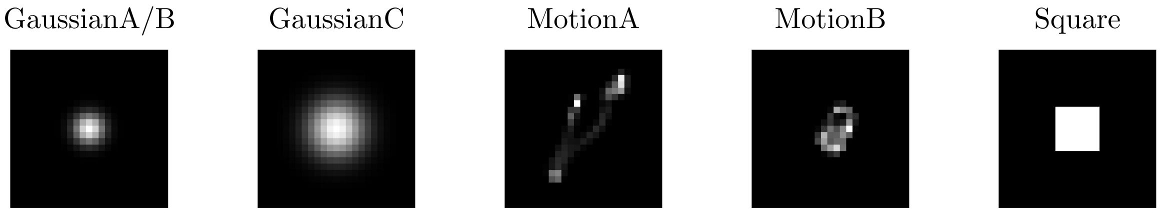


Figure 4.5: Blur kernels used in experiments, to produce the blurred images.

of $\sigma = 0.008$. This configuration is denoted as GaussianA. To evaluate the robustness of the proposed method with respect to the noise level, the same kernel is used with a Gaussian noise whose standard deviation is uniformly distributed between 0.01 and 0.05. The latter is denoted as GaussianB.

- The Gaussian kernel with a standard deviation of 3 pixels, and a Gaussian noise standard deviation of $\sigma = 0.04$, denoted as GaussianC.
- The eighth and third motion test kernels from [93], which are real-world camera shake kernels, with a Gaussian noise standard deviation of $\sigma = 0.01$. These settings are denoted as MotionA and MotionB, respectively.
- The square uniform kernel of size 7×7, with a Gaussian noise standard deviation of $\sigma = 0.01$. This configuration is referred to as Square.

The aforementioned blur kernels are reported in Figure 4.5.

4.4.4 Training

For each degradation model, one iRestNet network of 40 layers is trained. The first 30 layers are trained with a greedy approach, that consists in training individually each layer [56]. In our case, for $\mathcal{L}_0$, a minibatch of 10 images is selected at every iteration, randomly cropped using a window of size 256×256, blurred with the given kernel, and degraded with Gaussian noise; the training of $\mathcal{L}_0$ stops after a fixed number of epochs. Then, for each image of the training set, a random crop of size 256×256 is selected, blurred, corrupted with noise and passed through $\mathcal{L}_0$. The output is saved and used as an input to train $\mathcal{L}_1$. When the training of $\mathcal{L}_1$ is complete, its output is used to train the next layer, etc... This training strategy is chosen with regards to its low memory requirement: the number of layers is not limited by the hardware. The rest of the network, $\mathcal{L}_{\text{pp}} \circ \mathcal{L}_{39} \circ \ldots \circ \mathcal{L}_{30}$, is trained as one block for a variable number of epochs, depending on the progress on the validation set. With the validation set we monitor this last step of the training. In particular, by following the early stopping approach described in Chapter 2, the configuration of network parameters that gives the best performance on the validation set during the training is the one saved and used for the tests. To accelerate the training, for every

$k \in \{1, \dots, K-1\}$, the internal parameters of $\mathcal{L}_k$ are initialized with those of $\mathcal{L}_{k-1}$, while the internal filters of the CNN structure $\mathcal{L}_{\mathrm{pp}}$ are initialized with the suggested default values.
The learning algorithm used to train iRestNet is Adam optimizer, reported in Algorithm 12. Detailed information about the setting of this algorithm can be found in Table 4.1 below. In particular, when training $\mathcal{L}_{\mathrm{pp}} \circ \mathcal{L}_{39} \circ \dots \circ \mathcal{L}_{30}$, the initial learning rates, reported in the table, are scaled by a factor of 0.9 every 50 epochs.

	GaussianA	GaussianB	GaussianC	MotionA	MotionB	Square
Rates	(0.01,0.001)	(0.01,0.001)	(0.001,0.001)	(0.01,0.002)	(0.01,0.001)	(0.01,0.005)
Epochs	(40,393)	(40,340)	(40,300)	(40,1200)	(40,1250)	(40,740)

Table 4.1: Algorithm 12 setting. *First row*: global learning rates. *Second row*: number of epochs. For every pair, the first and second numbers correspond to the training of $(\mathcal{L}_k)_{0 \le k \le 29}$ and $\mathcal{L}_{\mathrm{pp}} \circ \mathcal{L}_{39} \circ \dots \circ \mathcal{L}_{30}$, respectively. For the exponential decay rates and the constant δ the default values are used, i.e., $\rho_1 = 0.9, \rho_2 = 0.999$ and $\delta = 10^{-8}$.

Note that for the first 30 layers, after each layer the quality of the restored training images should improve. This property comes from the training strategy, it is not encoded in the network: if memory was not an issue, then iRestNet should be trained in an end-to-end fashion, to be consistent with the iterative method.
In all training phases, Adam optimizer is used to minimize a training loss function, which is taken as the negative of the structural similarity measure (SSIM) [140] defined below

$$\mathrm{SSIM}(x, \bar{x}) = \frac{(2\mu_x \mu_{\bar{x}} + c_1)(2\sigma_x \sigma_{\bar{x}} + c_2)(2\mathrm{cov}_{x\bar{x}} + c_3)}{(\mu_x^2 + \mu_{\bar{x}}^2 + c_1)(\sigma_x^2 + \sigma_{\bar{x}}^2 + c_2)(\sigma_x \sigma_{\bar{x}} + c_3)}, \tag{4.21}$$

where $\bar{x}$ is the ground truth, x is the restored image, (μ_x, σ_x) and $(\mu_{\bar{x}}, \sigma_{\bar{x}})$ are mean and standard deviation of x and $\bar{x}$, respectively, $\mathrm{cov}_{x\bar{x}}$ is the cross–covariance of x and $\bar{x}$, and c_1, c_2 and c_3 are constants. As explained in [140], the SSIM is a good measure of perceived visual quality, since it is based on how the human eye extracts structural information from an image. Hence, it is more discriminative with regards to artifacts than the mean square error for instance.
The gradient of the SSIM loss with respect to the trainable parameters of the network is computed by combining the code available online [3] based on [140], with the implementation of the chain rule, in which we exploit automatic differentiation, presented in Section 2.3.3, and the expression given in Section 3.5.2 for the derivatives of the barrier proximity operator.
Codes are implemented in Pytorch. Some hidden layers in the post-processing part make use of ReLU, which is not differentiable everywhere. Since this nondifferentiability happens only

[3]`https://github.com/Po-Hsun-Su/pytorch-ssim`

at specific points for which the left and right derivatives are well–defined, Pytorch can handle it, as explained in Section 2.1.3. All trainings are conducted using a GeForce GTX 1080 GPU, or a Tesla V100 GPU. The training, which can be performed off-line, takes approximately 3 to 4 days for each blur kernel. Once a network is trained, it can be used for deblurring. Then, the time taken per test image is only about 1.4 sec on a GeForce GTX 1080 GPU.

4.4.5 Evaluation Metrics and Competitors

The performance of iRestNet is evaluated on the test set, in terms of the SSIM metric. The reconstruction given by the proposed approach is compared with a solution to problem (4.18) obtained using the projected gradient algorithm [75]. For every blurred test image, the pair (λ, δ) which leads to the best SSIM is selected using the simplex method. The solution given by this variational approach is referred to as VAR. The latter is an unrealistic scenario since it assumes that there is a perfect estimator of the error, but it gives an upper bound on the image quality that one can expect by solving (4.18).
We also use the following Deep Learning image reconstruction methods for comparison: EPLL [152], and MLP [124].
Finally, we include comparisons with three unfolded-based methods:

- IRCNN [150], where an empirical algorithm derived from an augmented Lagrangian is unfolded over 30 iterations and a CNN is used as a denoiser to update the splitting variable;

- FCNN [148], where the authors unfold a half-quadratic splitting algorithm and use a network to learn an effective regularization function;

- The method from [100], which is referred to as PDHG, where the authors perform a maximum of 30 iterations of a primal-dual hybrid gradient algorithm and the proximity operator of the second regularization function is replaced by a neural network.

For FCNN, we use the code that is available online, in which the authors provide a model that has only been trained for motion blurs. Hence, for a fair comparison, we only provide the results of FCNN on MotionA and MotionB, and we specify that this method is not applicable (n/a) to the other configurations. Similarly, for MLP and PDHG, the authors do not provide models that were trained specifically for MotionB and Square, so we do not test these methods on these two configurations.
Since MLP, EPLL and IRCNN require the knowledge of the noise level, for the GaussianB degradation model, we make use of the estimation of the noise standard deviation given by the method in [112]. In addition, since some comparison methods, like EPLL for instance, do not estimate well the borders of the images, the SSIM index is computed excluding a 6-pixel-wide frame for all images and all tested methods.

4.4.6 Results and Discussion

The average SSIM obtained with the different methods for the various blur kernels and noise levels on the BSD500 test set can be found in Table 4.2. The mean SSIM achieved with iRestNet on this test set is greater than those obtained with the other methods for all degradation models, except MotionA. For this kernel, the average SSIM achieved with iRestNet is the second highest value after IRCNN, which appears as the most competitive method. IRCNN involves two steps: first, a Wiener filter is applied to the blurred image, then, a neural network is used to predict the residual and denoise the image. These two steps are repeated 30 times, for 30 different manually tuned regularization parameters. In contrast, iRestNet does not require any tuning from the user regarding the regularization parameters during training.
For completeness, the SSIM of all images of the BSD500 test set are plotted in Figure 4.6 for the 6 different degradation models. As one can see, iRestNet performs well in terms of SSIM on most of the images.

	GaussianA	GaussianB	GaussianC	MotionA	MotionB	Square
Blurred	0.676	0.526	0.326	0.383	0.549	0.544
VAR	0.804	0.723	0.587	0.819	0.829	0.756
EPLL [152]	0.800	0.708	0.565	0.816	0.839	0.755
MLP [124]	0.821	0.734	0.608	0.854	n/a	n/a
PDHG [100]	0.796	0.716	0.563	0.801	n/a	n/a
IRCNN [150]	0.841	0.768	0.619	**0.902**	0.907	0.834
FCNN [148]	n/a	n/a	n/a	0.794	0.847	n/a
iRestNet	**0.853**	**0.787**	**0.641**	0.898	**0.910**	**0.840**

Table 4.2: SSIM results on the BSD500 test set.

Since no image was taken from Flickr for training iRestNet, the results on the Flickr30 test set show how well the performance of the trained networks are transferable on test sets with statistics that are different from those of the training set. Table 4.3 contains the average SSIM obtained with the different methods on the Flickr30 test set. Similarly to the BSD500 test set, iRestNet compares favorably with the other approaches on the Flickr30 test set.

Examples of visual results obtained with the different methods can be found in Figures 4.7 and 4.8, for two images from the BSD500 test set corrupted with the GaussianB and Square degradation models, respectively. We also provide in Figure 4.9 the results obtained for one image from the Flickr30 test set, that has been degraded with MotionB.
As one can see from inspecting these pictures, details from the snake's and caterpillar's skin patterns are better retrieved with iRestNet, which provides more visually-satisfactory results than competitors. Similarly, on Figure 4.9, competitors tend to smooth too much the details on the leaves, as it can be seen in the top left-hand corner. Regarding Figure 4.7, which belongs

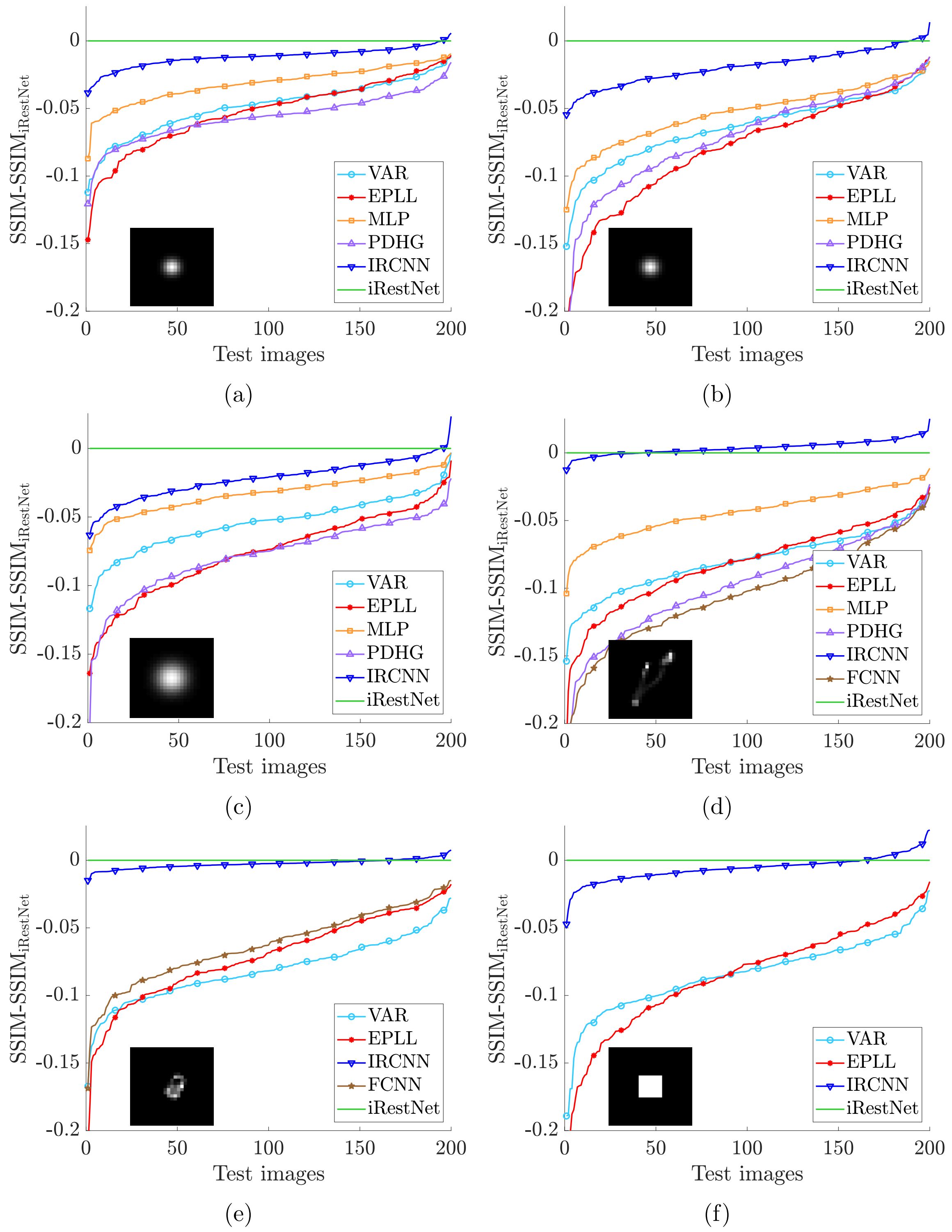


Figure 4.6: Sorted improvement of iRestNet with regards to other methods on the BSD500 test set using the SSIM metric: a negative value indicates a better performance of iRestNet. (a): GaussianA, (b): GaussianB, (c): GaussianC, (d): MotionA, (e): MotionB, (f): Square.

	GaussianA	GaussianB	GaussianC	MotionA	MotionB	Square
Blurred	0.723	0.545	0.355	0.376	0.590	0.579
VAR	0.857	0.776	0.639	0.856	0.869	0.818
EPLL [152]	0.860	0.770	0.616	0.857	0.887	0.827
MLP [124]	0.874	0.798	0.668	0.891	n/a	n/a
PDHG [100]	0.853	0.781	0.623	0.855	n/a	n/a
IRCNN [150]	0.885	0.819	0.676	**0.927**	**0.930**	**0.886**
FCNN [148]	n/a	n/a	n/a	0.801	0.890	n/a
iRestNet	**0.892**	**0.833**	**0.696**	0.919	**0.930**	**0.886**

Table 4.3: SSIM results on the Flickr30 test set.

to the test set with a level-varying noise, it is worth noticing that the green background is free of artifacts on the result obtained with the proposed method. This is not the case for the other methods, in particular for PDHG and IRCNN. This suggests that those two competitors are not robust to a small change in the noise level.
Finally, it is interesting to see in Figure 4.10 the sequences of stepsizes, barrier parameters and regularization weights obtained by passing the image from Figure 4.7 through the 40 layers of iRestNet.

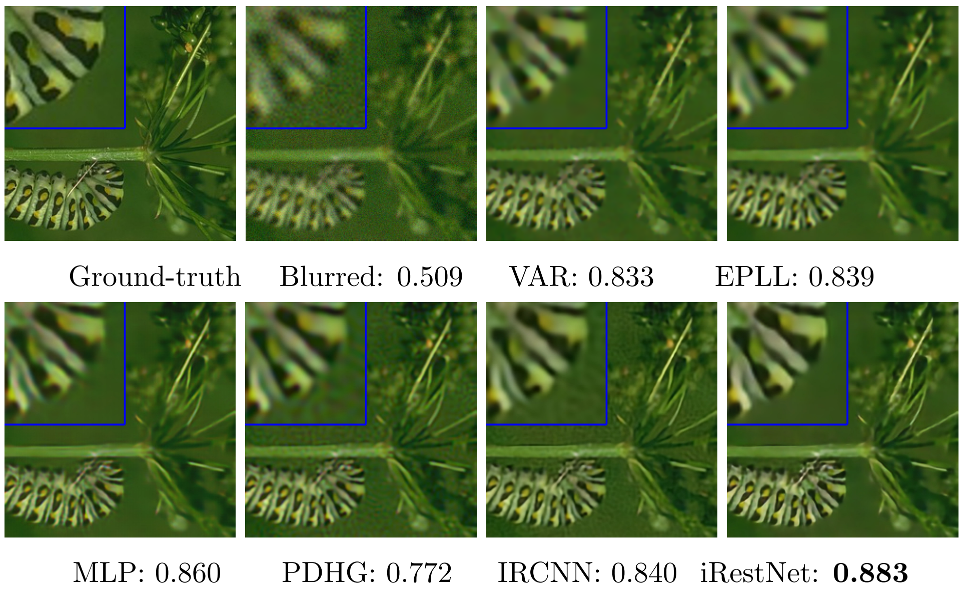


Figure 4.7: Visual results and SSIM obtained with the different methods on one image from the BSD500 test set degraded with GaussianB.

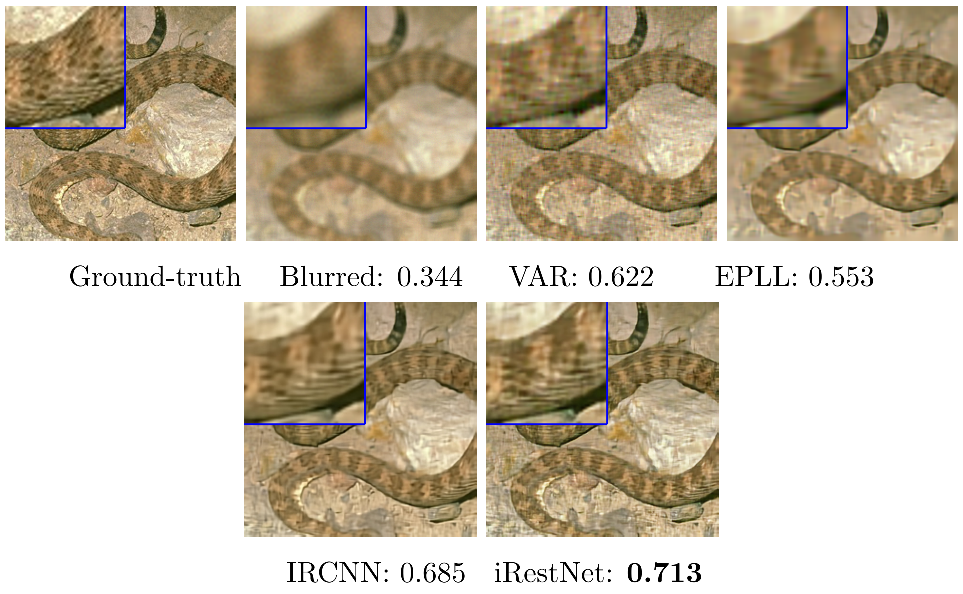


Figure 4.8: Visual results and SSIM obtained with the different methods on one image from the BSD500 test set degraded with Square.

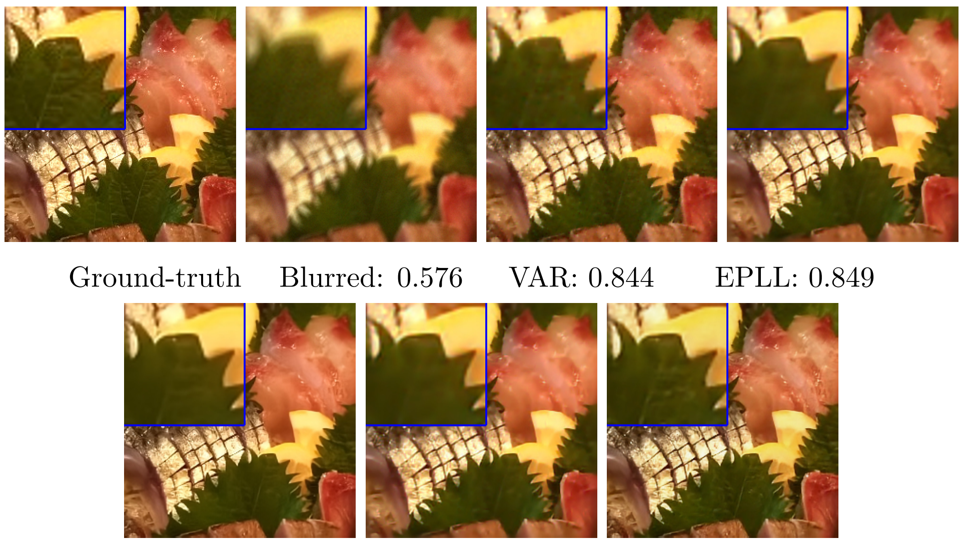


Figure 4.9: Visual results and SSIM obtained with the different methods on one image from the Flickr30 test set degraded with MotionB.

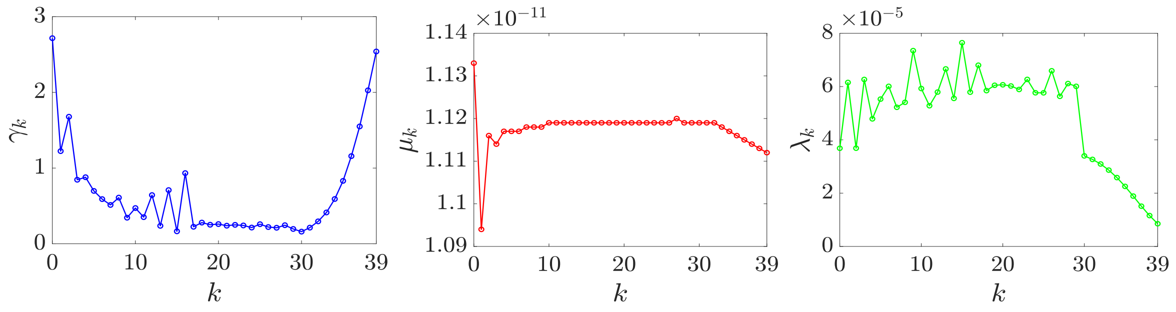


Figure 4.10: *Left to right*: estimated stepsize $(\gamma_k)_{0 \le k \le K-1}$, barrier parameter $(\mu_k)_{0 \le k \le K-1}$ and regularization weight $(\lambda_k)_{0 \le k \le K-1}$, for the image from Figure 4.7 passed through the network layers.

4.5 Extending iRestNet to Poisson Noise

What we want to do in future, is to extend our approach to a wider variety of applications. As a first step in this direction, in this last section we adapt iRestNet to address the image reconstruction problem on images corrupted with Poisson noise. This may be useful, because in imaging applications such as emission tomography, microscopy and astronomy, the main source of noise corrupting the images is photon counting [145]. In such a case, the statistics of the noisy images are well described by a Poisson process. We recall that the main properties of Poisson random variables are reported in section 3.2.1.

4.5.1 Problem Formulation for Poisson Noise

We treat the non-blind color image deblurring problem, where we take into consideration natural images corrupted with Poisson noise. Again, the degradation model is taken from equation (3.12),

$$y = \mathcal{D}(H\overline{x}). \tag{4.22}$$

where $y = (y^{(j)})_{1\leq j\leq 3} \in \mathbb{R}^{3n}$ is the noisy and blurred RGB image, $\overline{x} = (\overline{x}^{(j)})_{1\leq j\leq 3} \in \mathbb{R}^{3n}$ is the ground-truth, and $H \in R^{3n\times 3n}$ is a linear operator that models the circular convolution of a known blur kernel with each channel of the color image. In this case, $\mathcal{D}$ is the noise perturbation operator, which models the application of Poisson noise to the blurred image.

From the theory in Section 3.3, applying a MAP–approach for the deconvolution of images corrupted by Poisson noise, we obtain the following penalized formulation,

$$\underset{x\in\mathcal{C}}{\text{minimize}}\, \mathrm{KL}(y, Hx) + \lambda \sum_{i=1}^{3n} \sqrt{\frac{(D_{\mathrm{v}}x)_i^2 + (D_{\mathrm{h}}x)_i^2}{\delta^2} + 1}. \tag{4.23}$$

The data–fidelity function, which directly derives from Poisson noise, is given by the KL–divergence defined in (3.25), with $y_i \ln \frac{y_i}{(Hx)_i} = 0$ when $y_i = 0$, and with $(Hx)_i > 0$. Note that the last condition ensures the twice-differentiability of the KL, which is required to apply the backpropagation procedure in the network training. Instead the smoothed TV regularization is maintained for its edge-preserving properties, as we keep the feasible set $\mathcal{C} = [0, 1]^{3n}$, and $\delta = 0.01$ in all experiments.

4.5.2 Network Characteristics

As before, to approach Problem (4.23), we consider a neural network obtained by unfolding the forward–backward proximal IPM, proposed in Algorithm 14. The global structure is the same that was previously implemented for $K = 40$ layers, see Figure 4.1.
We only need to pay attention to the hidden structures $(\mathcal{L}_k^{(\lambda)})_{0\leq k\leq K-1}$, which estimate the regularization parameter λ. They are chosen in view of the objective function in problem (4.23)

and have the following expression,

$$(\forall k \in \{0,\dots,K-1\}) \;\; \lambda_k = \mathcal{L}_k^{(\lambda)}(x_k) = \frac{\text{Softplus}(b_k)}{\sqrt{2}\,(\eta(x_k) + \text{Softplus}(c_k))}, \tag{4.24}$$

where (b_k, c_k) is a pair of scalars learned by the network and $\eta(x_k)$ is the standard deviation of the concatenated spatial gradients of x_k, $[(D_\text{v}x_k)^T(D_\text{h}x_k)^T]$.
We recall that in Gaussian noise case, the regularization parameter was inferred from the image statistics and from the noise level in the observed image. In fact, in equation (4.19), the factor $\widehat{\sigma}(y)$ derived from [96], is an approximation of the variance of the additive Gaussian noise in the image. Switching to Poisson data, the noise variance is not uniform over the image spatial domain, but is pixel dependent. Also we can't use the approximation from (4.19), which is specific for additive Gaussian noise. So we have to modify equation (4.19) accordingly. In particular, we replace $\widehat{\sigma}(y)$ with a factor $1/\sqrt{2}$, that derives from the following discrepancy principle for Poisson data.

Discrepancy Principle

In order to estimate the regularization parameter for the Poisson noise, a discrepancy principle, which is a counterpart of the Morozov principle [11], has been proposed in [10, 145, 146], based on the KL data-fidelity function (3.25). As shown in [145], if $\bar{x}$ is the ground-truth corresponding to the observed image y, and the values of $\bar{x}$ are sufficiently large, then the expected value of $\text{KL}(y, H\bar{x})$ is approximately given by $n/2$, being n the number of pixels in the image spatial domain. Motivated by the above argument, it is proposed to select the value of λ such that the normalized discrepancy function

$$D_H(x,y) = \frac{2}{n}\text{KL}(y, Hx) \tag{4.25}$$

obeys

$$D_H(x_\lambda^*, y) = 1. \tag{4.26}$$

Here we denote as x_λ^* a nonnegative minimizer of (4.23). In this way, we are requiring that the discrepancy corresponding to the selected minimizer is close to that of the ground-truth [146].

4.5.3 Experimental Settings and Network Training

In this section, we present numerical experiments and the preliminary results we obtained on Poisson data, demonstrating that the proposed approach yields a good reconstruction quality, and it is competitive with standard variational methods.

We test iRestNet modified for Poisson data, with the same experimental settings used for the Gaussian noise case, seen in Section 4.4. In particular we use the same datasets of natural images, divided in training, validation and test sets, as explained in Section 4.4.3. The images have been center-cropped using a window of size 256×256. Blurry images are produced using the 25×25 blur kernels depicted in Figure 4.5:

- The Gaussian kernel with a standard deviation of 1.6 pixels (GaussianA configuration).

- The Gaussian kernel with a standard deviation of 3 pixels (GaussianC configuration).

- The eighth and third motion test kernels from [93] (MotionA and MotionB configuration, respectively).

- The square uniform kernel of size 7×7 (Square configuration).

The noise-free images, obtained from the blurring step, are corrupted with Poisson noise using a Python code available online [4]. Through this operation each pixel value of the noise-free image is replaced with the value generated by a Poisson distribution with mean given by the previous pixel value.

For each degradation model, one iRestNet network is trained using the approach explained in Section 4.4.4, with minibatches of 10 images. Again, we used the Adam optimizer [83] to minimize the training loss. Detailed information about the setting of the learning algorithm can be found in Table 4.4 below.

	GaussianA	GaussianC	MotionA	MotionB	Square
Rates	(0.01,0.001)	(0.01,0.001)	(0.01,0.001)	(0.01,0.001)	(0.005,0.001)
Epochs	(40,1000)	(40,1000)	(40,1200)	(40,1000)	(40,1000)

Table 4.4: Training information. *First row*: global learning rates. *Second row*: number of epochs. For every pair, the first and second numbers correspond to the training of $(\mathcal{L}_k)_{0\leq k\leq 29}$ and $\mathcal{L}_{\text{pp}} \circ \mathcal{L}_{39} \circ \ldots \circ \mathcal{L}_{30}$, respectively. In the second case, the global learning rates are scaled by a factor of 0.9 every 50 epochs. Exponential decay rates and δ are set as $\rho_1 = 0.9, \rho_2 = 0.999$ and $\delta = 10^{-8}$.

The training loss function is taken as the negative of the structural similarity measure (SSIM) [140] defined in equation (4.21). We tried to use the mean squared error (MSE) training loss function too, (1.7), but it did not improve the results.

All trainings are conducted using a GeForce GTX 1080 GPU or a GeForce GTX 1070 GPU. The training, which can be performed off-line, takes approximately 4 to 5 days for each blur kernel, while the time taken per test image is only about 1.2 sec on a GeForce GTX 1080 GPU.

[4] `https://github.com/scikit-image/scikit-image/blob/master/skimage/util/noise.py`

4.5.4 Results and Discussion

As for the Gaussian case, the reconstruction quality is evaluated in terms of the SSIM metric. The reconstructed image given by the proposed approach is compared with a solution to problem (4.23) obtained using a scaled gradient projection (SGP) algorithm from [16, 18]. Again, for every image of the test set, a simplex method selects the optimal pair (λ, δ) to be used in SGP for obtaining the best SSIM. To the best of our knowledge, for the problem of image reconstruction applied to Poisson data, there are no Deep Learning or unfolding methods, whose codes are available online. Because of this, SGP is the only method with which we confronted ourself.

In Table 4.5 it is reported the average SSIM of the reconstructed images of the BSD500 test set, for the different kernel configurations. As we can see, the proposed approach yields a good reconstruction quality. The mean SSIM achieved with iRestNet on this test set is competitive to that obtained with SGP, with some differences in the results that depend on the considered blurring configuration. These are preliminary results, obtained while still working on the iterative part of iRestNet, to further improve the efficiency of the first 30 layers. For completeness, the SSIM of all images of the BSD500 test set are plotted in Figure 4.11 for the 5 different degradation models. As one can see, iRestNet performs well in terms of SSIM.

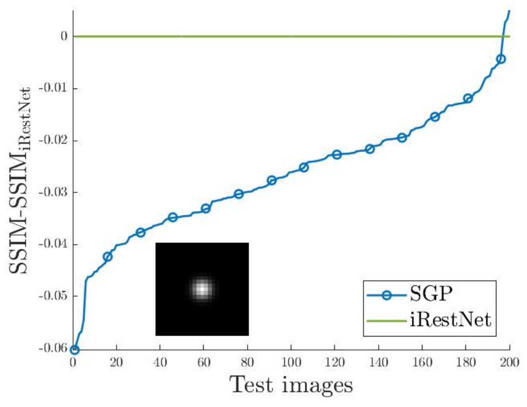
	GaussianA	GaussianB	MotionA	MotionB	Square
Blurred	0.697	0.549	0.408	0.579	0.573
SGP	0.883	0.722	**0.961**	**0.972**	0.671
iRestNet	**0.910**	**0.763**	0.931	0.965	**0.872**

Table 4.5: SSIM results on the Poisson images of the BSD500 test set.

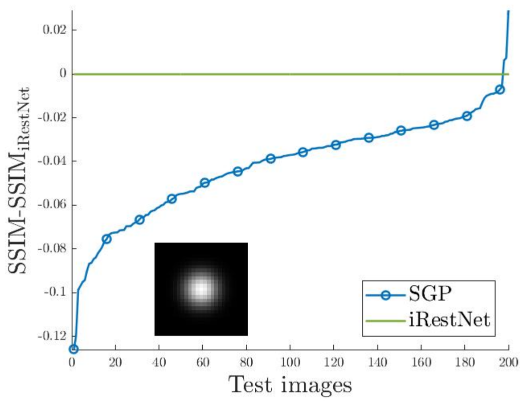
The results on the Flickr30 test set are also reported for the Poisson noise case. Table 4.6 contains the average SSIM obtained with iRestNet and SGP on the Flickr30 test set. Similarly to the BSD500 test set, iRestNet is competitive with SGP on this dataset.

	GaussianA	GaussianB	MotionA	MotionB	Square
Blurred	0.745	0.579	0.400	0.621	0.608
SGP	0.917	0.783	**0.973**	**0.979**	0.692
iRestNet	**0.933**	**0.814**	0.929	0.966	**0.894**

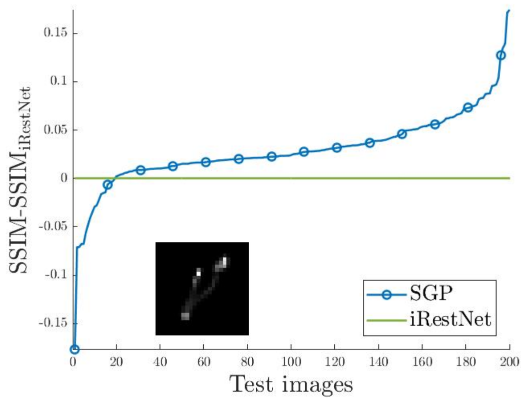
Table 4.6: SSIM results on the Poisson images of the Flickr30 test set.



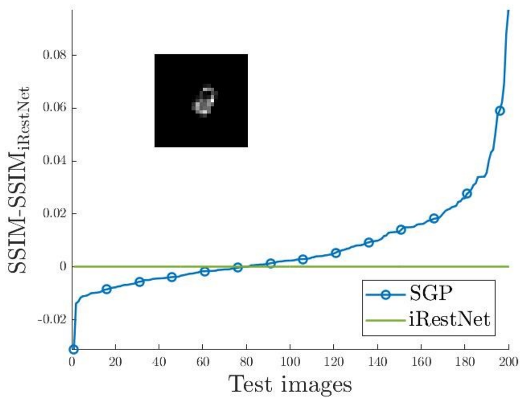
(a)



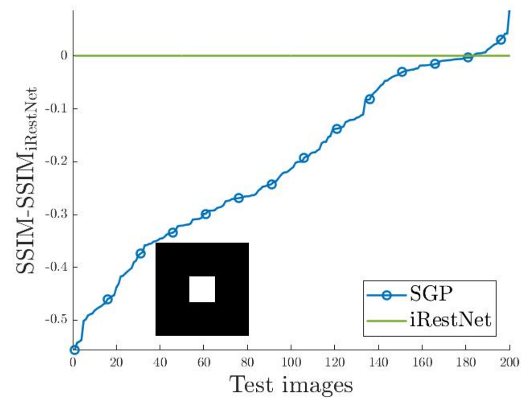
(b)



(c)



(d)



(e)

Figure 4.11: Sorted improvement of iRestNet with regards to SGP method on the BSD500 test set using the SSIM metric: a negative value indicates a better performance of iRestNet. (a): GaussianA, (b): GaussianC, (c): MotionA, (d): MotionB, (e): Square.

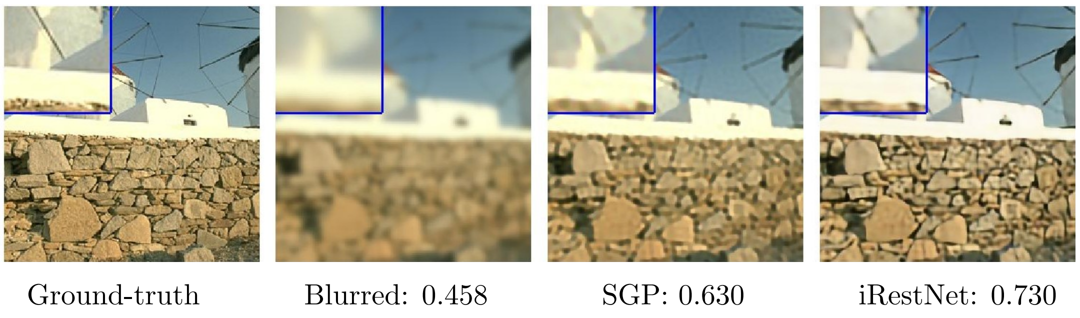


Figure 4.12: Visual results and SSIM obtained with SGP and iRestNet methods on one image from the BSD500 test set degraded with GaussianB.

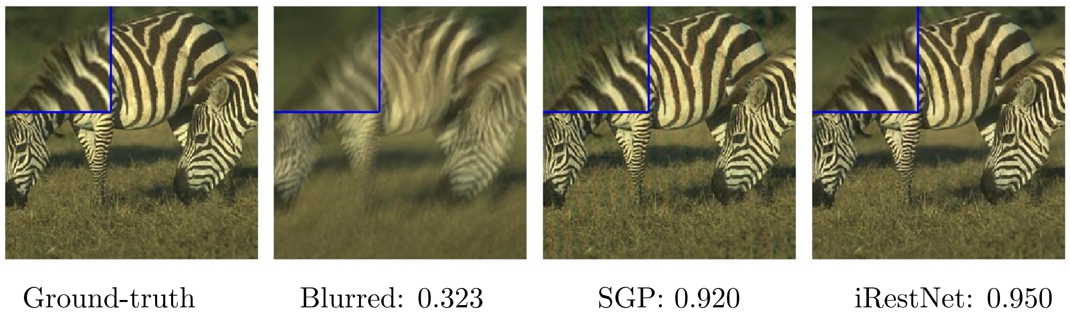


Figure 4.13: Visual results and SSIM obtained with SGP and iRestNet methods on one image from the BSD500 test set degraded with MotionA.

Examples of visual results obtained with iRestNet and SGP methods can be found in Figures 4.12, 4.13 and 4.14 for three images from the BSD500 test set and the blur kernels GaussianB, MotionA and Square, respectively. We also provide, in Figure 4.15, the visual results obtained for one image from the Flickr30 test set, degraded with Square.
As one can see from these pictures, details in all the examples are well retrieved with iRestNet, which provides more visually-satisfactory results than SGP. In Figure 4.12, it is particularly noticeable how the details and the contrasts in the stone wall are better reconstructed with our approach, than with SGP. It is also worth noticing that SGP reconstruction shows artifacts in some examples, particularly those that belong to the test sets degraded with Square. See, for example, the smooth green area behind the zebra, reported in the top–left corner of Figure 4.13, or the doll hands pointed out in Figure 4.15. These artifacts do not appear in iRestNet reconstructions. Conversely, it can be seen in Figure 4.14 that iRestNet can reconstruct both the more detailed parts of the fox fur and the smoothness in petals well.
Finally, Figure 4.16 shows the stepsize, barrier parameter and regularization weight sequences obtained by passing the image of Figure 4.13 through the 40 layers of iRestNet.

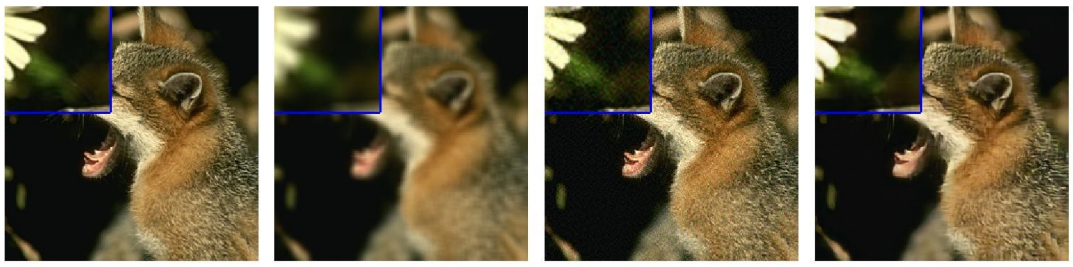


Figure 4.14: Visual results and SSIM obtained with the SGP and iRestNet methods on one image from the BSD500 test set degraded with Square.

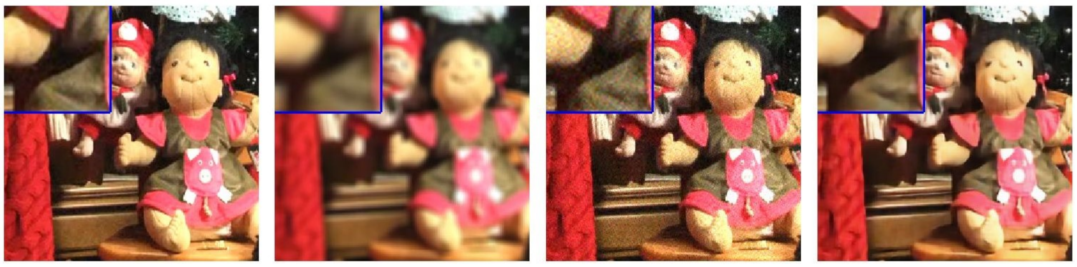


Figure 4.15: Visual results and SSIM obtained with the SGP and iRestNet methods on one image from the Flickr30 test set degraded with Square.

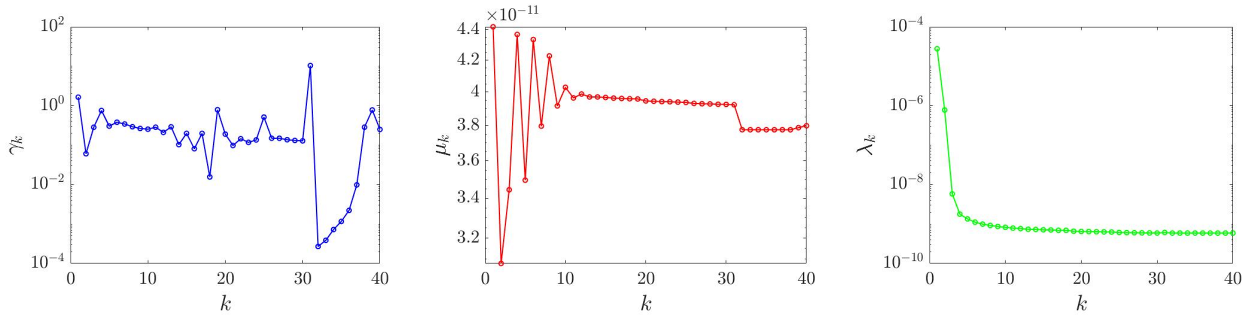


Figure 4.16: *Left to right*: estimated stepsize $(\gamma_k)_{0\leq k\leq K-1}$, barrier parameter $(\mu_k)_{0\leq k\leq K-1}$ and regularization weight $(\lambda_k)_{0\leq k\leq K-1}$ for the image from Figure 4.13 passed through the network layers.

Conclusions

The research activity presented in this thesis has dealt with the study and the analysis of Deep Learning and variational methods for inverse problems in imaging. The work mainly concerned the design of a suitable neural network architecture, called iRestNet, which derives from a variational formulation of an inverse problem. In particular, we constructed the structure of iRestNet, by unfolding the iterates of a proximal interior point algorithm, considered to address the variational problem, into the layers of the network. It is to be noted that the variational formulation of the problem involves the minimization of an objective function, which is parametrized by several variables, like the regularization parameter, the stepsize and the barrier parameter. An advantage of our method is to automatically determine these variables, without performing any parameter search, that is usually time–consuming when applying traditional IPMs. Also, useful constraints on the sought solution are enforced thanks to a logarithmic barrier, so providing more control over the output of the network.
We have shown for three standard types of constraints that the involved proximity operator can easily be computed, and that its derivatives, needed in the network training, are well-defined and computable. We provided their expressions in Chapter 3.
In the case of a quadratic cost function, the theoretical result of Section 4.3 regarding the robustness of the network with respect to an input perturbation, ensures the reliability of the proposed method, which is crucial for many practical applications. In a future work, it would be interesting to extend the scope of this study to a wider class of problems, and to illustrate this stability result by numerical experiments on different applications like classification.

The proposed method have been tested on a set of problems of image reconstruction. Numerical experiments are conducted on datasets of natural color images, blurred with different types of blur and corrupted with Gaussian noise. In this case, the problem is formulated as the constrained minimization of an objective function expressed as the sum of a mean–squared loss function and a smoothed Total Variation regularizer. The choice of the data–fidelity function is strictly related to the noise that affects the images, while the regularization function is chosen for its edge–preserving properties. From this problem formulation we derive the IPM iteration, and we use it to construct the layers of iRestNet. The network is then trained in a supervised fashion, in order to find an optimal reconstruction quality, and it is tested on a test set of images. Experimental results show that iRestNet performs favourably compared to state-of-the-art variational and machine learning methods, demonstrating that in many cases the proposed approach yields a better reconstruction quality than the methods we have been confronted with. Advantages of the proposed approach are that, in contrast with its evaluated competitors, it does not require any knowledge about the noise level and it does not involve any hand-selection of the regularization parameters. Also, once iRestNet is trained, running the model on the test images is fast. The time taken per test image is only about 1.4 seconds on the used GPU.

One limitation of iRestNet is that the network needs to be trained for a given blur kernel. A direction for future works is to extend the method to situations in which the observation model is not fully known, so as to address blind or semi-blind deconvolution problems. For now, we are focusing on extending the approach to a wider variety of applications. As a first step in this direction, we adapted iRestNet to address the image reconstruction problem on images corrupted with Poisson noise. This is useful, because in imaging applications such as emission tomography, microscopy and astronomy, the main source of noise corrupting the images is photon counting, which is described by a Poisson process.

Numerical experiments on Poisson data demonstrate that the proposed approach yields to a good reconstruction quality. Preliminary results show that iRestNet is competitive with the standard variational method with which we confronted, but we think the results can be further improved, by working on the iterative procedure used to construct the layers of iRestNet.

Appendix A

Mathematical Background

A.1 Elements of Functional Analysis

Let us introduce the concept of Hilbert space, that will be a key point in the study of inverse problems.

Definition A.1. *A complex vector space X is called a inner product space if there exists a map $(.,.)_X : X \times X \to \mathbb{C}$, called inner (or scalar) product, that for each $x, y, z \in X$ and $\lambda \in \mathbb{C}$ satisfies:*

a) $(x, x)_X \geq 0$ *and* $(x, x)_X = 0$ *if and only if* $x = 0$*;*

b) $(x + y, z)_X = (x, z)_X + (y, z)_X$*;*

c) $(\lambda x, y)_X = \lambda(x, y)_X$*;*

d) $(x, y)_X = \overline{(y, x)_X}$*, where* $\overline{x}$ *denotes the complex conjugate of* x*.*

Among the straightforward consequences of such properties we can remark that:

i) $(x, y)_X = 0, \forall y \in X \iff x = 0$ by combining a) and c);

ii) $(x, y + z)_X = (x, y)_X + (x, z)_X$ from b) and d);

iii) $(x, \lambda y)_X = \overline{\lambda}(x, y)_X$ from c) and d).

Definition A.2. *Given two complex vector spaces X and Y, a function $f : X \to Y$ is said to be linear if it preserves the operations of addition and scalar multiplication, i.e., for any two vectors $x, y \in X$ and any scalar $\lambda \in \mathbb{C}$ the following conditions are satisfied:*

- $f(x + y) = f(x) + f(y)$*;*

- $f(\lambda x) = \lambda f(x)$.

It follows that an inner product is linear with respect to the first variable and, therefore, the map

$$(.,y)_X : x \to (x,y)_X$$

is, for a given $y \in X$, a linear function on X.

Definition A.3. *A function $\|.\|_X : X \to \mathbb{R}$ is called norm if, for each $x, y \in X$ and $\lambda \in \mathbb{C}$, the following conditions hold true:*

a) $\|x\|_X \geq 0$ and $\|x\|_X = 0$ if and only if $x = 0$;

b) $\|\lambda x\|_X = |\lambda|\|x\|_X$;

c) $\|x+y\|_X \leq \|x\|_X + \|y\|_X$ (triangle inequality).

A vector space X endowed with a norm is called normed vector space.

In particular, the function

$$\|x\|_X := \sqrt{(x,x)_X} \tag{A.1}$$

is a norm in X called norm induced by the inner product $(.,y)_X$. In Chapter 3, when the context is clear, we write $(.,.)$ and $\|.\|$ instead of $(.,.)_X$ and $\|.\|_X$, except in some cases where the distinction is necessary.

Definition A.4. *A sequence x_n in a normed vector space $(X, \|.\|)$ is said to be a Cauchy sequence if $\forall \epsilon > 0 \ \exists N \in \mathbb{Z}$ such that $\|x_m - x_n\| < \epsilon$, $\forall m > N$ and $\forall n > N$.*
A normed vector space $(X, \|.\|)$ is called complete if any Cauchy sequence in X converges to an element of X.
An inner product space that results to be complete with respect to the norm induced by the inner product itself is called Hilbert space.

Finally, we recall some definitions and properties, that will be useful in the discussion of Chapter 3.

Definition A.5. *Let X, Y be Hilbert spaces. A linear operator $A : X \to Y$ is said to be bounded if there exists a constant $C > 0$ such that $||Ax||_Y \leq C||x||_X$ for every $x \in X$.*
Let us also define the operator norm of A as

$$\begin{aligned} ||A|| &= \inf\{C \geq 0 \ : \ ||Ax||_Y \leq C||x||_X \ \ \forall x \in X\} \\ &= \sup\{||Ax||_Y \ : \ x \in X, \ ||x||_X = 1\} \end{aligned} \tag{A.2}$$

An immediate consequence of the definition of operator norm is given by the following inequality

$$||Ax||_Y \leq ||A|| \, ||x||_X. \tag{A.3}$$

Definition A.6. *The kernel (or null space) of an operator A between the Hilbert spaces X and Y is the subspace $N(A) = \{x \in X \mid Ax = 0\} \subseteq X$, while the range of A is the subspace $R(A) = \{y \in Y \mid y = Ax \text{ for some } x \in X\} \subseteq Y$.*

Denoting with $D(A)$ the domain of an operator A, we can recall the following properties:

- If A is bounded, then $N(A)$ is a closed subspace of X.
- If A is injective, then $N(A) = 0$.
- If A is surjective, then $R(A) = Y$.
- If A is bijective, then there exists the inverse operator $A^{-1} : R(A) \to D(A)$ defined by $A^{-1}y = x$ for all $y = Ax \in R(A)$.

Theorem A.1. *A linear operator A between the Hilbert spaces X and Y is bounded if and only if is continuous. Moreover, if A is bounded and bijective, then A^{-1} is continuous.*

To conclude, we recall the Implicit Function Theorem.

Theorem A.2 (Implicit Function Theorem)**.** *Let U be an open subset of $\mathbb{R}^m \times \mathbb{R}^n$ and $F : U \to \mathbb{R}^n$ a function of class $\mathcal{C}^1$. Let $(a,b) \in U$ and consider the system of equations*

$$F(x,y) = F(a,b).$$

Assume that the Jacobian matrix

$$J_F^{(y)}(a,b) := \left(\frac{\partial F_i}{\partial y_k}(a,b) \right)_{1 \leq i,k \leq n}$$

is invertible. Then there exists an open subset $V \in \mathbb{R}^m$ containing a and an open subset $W \in \mathbb{R}^n$ containing b such that $V \times W \subset U$ and such that, for all $x \in V$, there exists a unique $y = \varphi(x) \in W$ such that (x,y) is a solution to the above system of equations. Moreover, the function $\varphi : V \to W$ is of class $\mathcal{C}^1$ and the derivative at $x \in V$ satisfies the equation

$$J_\varphi(x) = -J_F^{(y)}(x,\varphi(x))^{-1} J_F^{(x)}(x,\varphi(x)). \tag{A.4}$$

A.2 Elements of Convex Analysis

Let us recall some useful definitions and properties of convex analysis, that will be used for introducing interior point methods, in Chapter 3.

Definition A.7. *The set $A \subset \mathbb{R}^n$ is said to be closed if and only if it contains all of its limit points, i.e., the points $x \in X$ for which every neighbourhood of x contains at least one point of A different from x itself.*

Definition A.8. *Given a set $A \subseteq \mathbb{R}^n$, a point x is an interior point of A if $x \in A$ and there exists a neighbourhood of x that is entirely contained in A. The interior of A, denoted by* $\operatorname{int} A$, *is the collection of all interior points of A.*

Definition A.9. *Given a set $A \subseteq \mathbb{R}^n$, a point x is a boundary point of A if every neighbourhood of x contains at least one point in A, and at least one point not in A. The boudary of A is the collection of all boundary points of A.*

Remark A.1. It is straightforward to show that a closed set contains all its boundary points.

Definition A.10. *The set $A \subset \mathbb{R}^n$ is a convex set if the affine combination* $(1-t)x + ty \in A$, $\forall x, y \in A, \forall t \in [0, 1]$.

Definition A.11. *The set $A \in \mathbb{R}^n$ is said to be bounded if and only if it is contained inside some ball $x_1^2 + ... + x_n^2 \leq \rho^2$ of finite radius ρ.*

Definition A.12. *The domain of a function $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ is the set* $\operatorname{dom}(f)$ *given by*

$$\operatorname{dom}(f) := \{x \in \mathbb{R}^n \mid f(x) < +\infty\}.$$

Definition A.13. *A function $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ is said to be proper if there exists $\bar{x} \in \mathbb{R}^n$ such that $f(\bar{x}) < +\infty$; namely if* $\operatorname{dom}(f) \neq \emptyset$.

Remark A.2. We recall that the lower limit of a function can be characterized in the following way [114, Lemma 1.7]

$$\liminf_{y \to x} f(y) = \min\{\alpha \in \bar{\mathbb{R}} : \ \exists \ \{x^{(k)}\}_{k \in \mathbb{N}} \subseteq \mathbb{R}^n \text{ such that } x^{(k)} \to x, \ f(x^{(k)}) \to \alpha\}.$$

Similarly, there is an upper limit ($\limsup$) formula,

$$\limsup_{y \to x} f(y) = \max\{\alpha \in \bar{\mathbb{R}} : \ \exists \ \{x^{(k)}\}_{k \in \mathbb{N}} \subseteq \mathbb{R}^n \text{ such that } x^{(k)} \to x, \ f(x^{(k)}) \to \alpha\}.$$

Definition A.14. *[114, Definition 1.5] A function $f : \mathbb{R}^n \to \bar{\mathbb{R}}$ is lower semicontinuous (lsc) at x if*

$$f(x) = \liminf_{y \to x} f(y) = \sup_{\delta > 0} \left(\inf_{y \in B(x,\delta)} f(y) \right). \tag{A.5}$$

Similarly, f is upper semicontinuous at x if

$$f(x) = \limsup_{y \to x} f(x) = \inf_{\delta > 0} \left(\sup_{y \in B(x,\delta)} f(y) \right). \tag{A.6}$$

Remark A.3. The function f is continuous at x if and only if f is both lower and upper semicontinuous at x.

Definition A.15. *Let $A \subseteq \mathbb{R}^n$ be a convex set. A function $f : A \to \mathbb{R}$ is convex if*

$$f((1-\lambda)x + \lambda y) \leq (1-\lambda) f(x) + \lambda f(y), \quad \forall\ x, y \in A,\ \forall\ \lambda \in [0,1].$$

For every $q \in \mathbb{N}$, we will denote as $\Gamma_0(\mathbb{R}^q)$ the set of functions which are convex, proper, lower semicontinuous on $\mathbb{R}^q$, and which take values in $\mathbb{R} \cup \{+\infty\}$.

Definition A.16. *A function $f : \mathbb{R}^n \to \bar{\mathbb{R}}$ is called coercive if and only if $f(x) \to +\infty$ as $\|x\| \to +\infty$.*

Bibliography

[1] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[2] B. Amos and J. Z. Kolter. OptNet: Differentiable optimization as a layer in neural networks. In *Proceedings of the International Conference on Machine Learning*, pages 136–145, 6–11 Aug 2017.

[3] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas. Learning to learn by gradient descent by gradient descent. In *Proceedings of Advances in Neural Information Processing Systems*, pages 3981–3989, Barcelona, Spain, 5–10 Dec 2016.

[4] J.-F. Aujol. Some first-order algorithms for total variation based image restoration. *Journal of Mathematical Imaging and Vision*, 34(3):307–327, July 2009.

[5] M. S. Bartlett. An inverse matrix adjustment arising in discriminant analysis. *The Annals of Mathematical Statistics*, 22(1):107–111, 1951.

[6] H. H. Bauschke and P. L. Combettes. *Convex analysis and monotone operator theory in Hilbert spaces.* Springer, 2017.

[7] T. Bayes. Lii. an essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, frs communicated by mr. price, in a letter to john canton, amfr s. *Philosophical transactions of the Royal Society of London*, (53):370–418, 1763.

[8] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

[9] M. Bertero and P. Boccacci. *Introduction to inverse problems in imaging.* CRC press, 1998.

[10] M. Bertero, P. Boccacci, G. Talenti, R. Zanella, and L. Zanni. A discrepancy principle for poisson data. *Inverse problems*, 26(10):105004, 2010.

[11] M. Bertero, C. De Mol, and E. R. Pike. Linear inverse problems with discrete data: Ii. stability and regularisation. *Inverse problems*, 4(3):573, 1988.

[12] M. Bertero, H. Lantéri, L. Zanni, et al. Iterative image reconstruction: a point of view. *Mathematical Methods in Biomedical Imaging and Intensity-Modulated Radiation Therapy (IMRT)*, 7:37–63, 2008.

[13] C. Bertocchi, E. Chouzenoux, M.-C. Corbineau, J.-C. Pesquet, and M. Prato. Deep unfolding of a proximal interior point method for image restoration. *Inverse Problems*, 36(3):034005, feb 2020.

[14] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic programming*, volume 5. Athena Scientific Belmont, MA, 1996.

[15] C. M. Bishop et al. *Neural networks for pattern recognition.* Oxford university press, 1995.

[16] S. Bonettini and M. Prato. New convergence results for the scaled gradient projection method. *Inverse Problems*, 31(9):1–20, 2015.

[17] S. Bonettini and T. Serafini. Non-negatively constrained image deblurring with an inexact interior point method. *Journal of Computational and Applied Mathematics*, 231(1):236 – 248, 2009.

[18] S. Bonettini, R. Zanella, and L. Zanni. A scaled gradient projection method for constrained image deblurring. *Inverse problems*, 25(1):015002, 2008.

[19] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[20] L. Bottou. Online learning and stochastic approximations. *On-line learning in neural networks*, 17(9):142, 1998.

[21] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

[22] D. Boublil, M. Elad, J. Shtok, and M. Zibulevsky. Spatially-adaptive reconstruction in computed tomography using neural networks. *IEEE Transactions on Medical Imaging*, 34(7):1474–1485, 2015.

[23] Y.-L. Boureau, J. Ponce, and Y. LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 111–118, 2010.

[24] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–222, 2011.

[25] T. A. Bubba, G. Kutyniok, M. Lassas, M. März, W. Samek, S. Siltanen, and V. Srinivasan. Learning the invisible: A hybrid deep learning-shearlet framework for limited angle computed tomography. *Inverse Problems*, 35(6):064002, 2019.

[26] T. F. Chan and C.-K. Wong. Total variation blind deconvolution. *IEEE transactions on Image Processing*, 7(3):370–375, 1998.

[27] P. Charbonnier, L. Blanc-Féraud, G. Aubert, and M. Barlaud. Deterministic edge-preserving regularization in computed imaging. *IEEE Transactions on Image Processing*, 6(2):298–311, 1997.

[28] P. Chaudhari, A. Choromanska, S. Soatto, Y. LeCun, C. Baldassi, C. Borgs, J. Chayes, L. Sagun, and R. Zecchina. Entropy-sgd: Biasing gradient descent into wide valleys. *arXiv preprint arXiv:1611.01838*, 2016.

[29] C. Chaux, P. L. Combettes, J.-C. Pesquet, and V. R. Wajs. A variational formulation for frame-based inverse problems. *Inverse Problems*, 23(4):1495–1518, June 2007.

[30] Y. Chen and T. Pock. Trainable nonlinear reaction diffusion: A flexible framework for fast and effective image restoration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1256–1272, 2017.

[31] J.-T. Chien and C.-H. Lee. Deep unfolding for topic models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(2):318–331, 2018.

[32] E. Chouzenoux, M.-C. Corbineau, and J.-C. Pesquet. A proximal interior point algorithm with applications to image processing. *HAL preprint hal-02120005*, 2019.

[33] M. Claesen and B. De Moor. Hyperparameter search in machine learning. *arXiv preprint arXiv:1502.02127*, 2015.

[34] P. L. Combettes and J.-C. Pesquet. Proximal splitting methods in signal processing. In H. H. Bauschke, R. Burachik, P. L. Combettes, V. Elser, D. R. Luke, and H. Wolkowicz, editors, *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, pages 185–212. Springer-Verlag, New York, 2010.

[35] P. L. Combettes and J.-C. Pesquet. Deep neural network structures solving variational inequalities. *arXiv preprint arXiv:1808.07526*, 2018.

[36] P. L. Combettes and V. R. Wajs. Signal recovery by proximal forward-backward splitting. *Multiscale Modeling and Simulation*, 4(4):1168–1200, Nov 2005.

[37] M.-C. Corbineau, C. Bertocchi, E. Chouzenoux, M. Prato, and J.-C. Pesquet. Learned image deblurring by unfolding a proximal interior point algorithm. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 4664–4668. IEEE, Sep. 2019.

[38] M.-C. Corbineau, E. Chouzenoux, and J.-C. Pesquet. Geometry-texture decomposition/reconstruction using a proximal interior point algorithm. In *Proceedings of the IEEE Sensor Array and Multichannel Signal processing Workshop*, 8–11 Jul 2018.

[39] M.-C. Corbineau, E. Chouzenoux, and J.-C. Pesquet. PIPA: a new proximal interior point algorithm for large-scale convex optimization. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Calgary, Canada, 15–20 Apr 2018.

[40] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[41] F. Cucker and S. Smale. Best choices for regularization parameters in learning theory: on the bias-variance problem. *Foundations of computational Mathematics*, 2(4):413–428, 2002.

[42] M.-O. Czarnecki, N. Noun, and J. Peypouquet. Splitting forward-backward penalty scheme for constrained variational problems. *Journal of Convex Analysis*, 23(2):531–565, 2016.

[43] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pages 2933–2941, 2014.

[44] C.-A. Deledalle, S. Vaiter, J. Fadili, and G. Peyré. Stein Unbiased GrAdient estimator of the Risk (SUGAR) for multiple parameter selection. *SIAM Journal on Imaging Sciences*, 7(4):2448–2487, 2014.

[45] S. Diamond, V. Sitzmann, F. Heide, and G. Wetzstein. Unrolled optimization with deep priors. *arXiv preprint arXiv:1705.08041*, 2017.

[46] D. C. Dobson and F. Santosa. Recovery of blocky images from noisy and blurred data. *SIAM Journal on Applied Mathematics*, 56(4):1181–1198, 1996.

[47] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[48] C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, and R. Garcia. Incorporating second-order functional knowledge for better option pricing. In *Proceedings of Advances in Neural*

Information Processing Systems (NIPS), pages 472–478, Vancouver, Canada, 3–8 Dec 2001.

[49] S. Durand and M. Nikolova. Stability of minimizers of regularized least squares objective functions I: study of the local behaviour. *Applied Mathematics and Optimization (Springer-Verlag New York)*, 53:185–208, 2006.

[50] H. W. Engl, M. Hanke, and A. Neubauer. *Regularization of inverse problems.* Kluwer, Dordrecht, 1996.

[51] T. Evgeniou, M. Pontil, and T. Poggio. Regularization networks and support vector machines. *Advances in computational mathematics*, 13(1):1, 2000.

[52] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

[53] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, 1984.

[54] F. Girosi. Regularization theory, radial basis functions and networks. In *From statistics to neural networks*, pages 166–187. Springer, 1994.

[55] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[56] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[57] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint arXiv:1312.6082*, 2013.

[58] I. J. Goodfellow, O. Vinyals, and A. M. Saxe. Qualitatively characterizing neural network optimization problems. *arXiv preprint arXiv:1412.6544*, 2014.

[59] K. Gregor and Y. LeCun. Learning fast approximations of sparse coding. In *Proceedings of the International Conference on International Conference on Machine Learning*, pages 399–406, Haifa, Israel, 21-24 Jun 2010. Omnipress.

[60] L. Grippo and M. Sciandrone. Metodi di ottimizzazione per le reti neurali. *Rapporto Tecnico*, pages 09–03, 2003.

[61] J. Hadamard. Sur les problèmes aux dérivées partielles et leur signification physique. *Princeton university bulletin*, pages 49–52, 1902.

[62] J. Hadamard. *Lectures on Cauchy's Problem in Linear Partial Differential Equations*, volume 37. Yale University Press, 1923.

[63] P. C. Hansen. *Discrete inverse problems: insight and algorithms*, volume 7. Siam, 2010.

[64] P. C. Hansen, J. G. Nagy, and D. P. O'leary. *Deblurring images: matrices, spectra, and filtering*, volume 3. Siam, 2006.

[65] T. Hastie, S. Rosset, R. Tibshirani, and J. Zhu. The entire regularization path for the support vector machine. *Journal of Machine Learning Research*, 5(Oct):1391–1415, 2004.

[66] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[67] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, Las Vegas, NV, USA, 26 Jun – 1 Jul 2016.

[68] J. R. Hershey, J. Le Roux, and F. Weninger. Deep unfolding: model-based inspiration of novel deep architectures. *arXiv preprint arXiv:1409.2574*, 2014.

[69] T. M. Heskes and B. Kappen. On-line learning processes in artificial neural networks. In *North-Holland Mathematical Library*, volume 51, pages 199–233. Elsevier, 1993.

[70] G. E. Hinton. A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade*, pages 599–619. Springer, 2012.

[71] S. Hochreiter and J. Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997.

[72] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[73] C.-W. Hsu, C.-C. Chang, C.-J. Lin, et al. A practical guide to support vector classification. 2003.

[74] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[75] A. N. Iusem. On the convergence properties of the projected gradient method for convex optimization. *Computational Applied Mathematics*, 22(1):37–52, 2003.

[76] K. H. Jin, M. T. McCann, E. Froustey, and M. Unser. Deep convolutional neural network for inverse problems in imaging. *IEEE Transactions on Image Processing*, 26(9):4509–4522, 2017.

[77] C. A. Johnson, J. Seidel, and A. Sofer. Interior-point methodology for 3-d pet reconstruction. *IEEE Transactions on medical imaging*, 19(4):271–285, 2000.

[78] U. S. Kamilov and H. Mansour. Learning optimal nonlinearities for iterative thresholding algorithms. *IEEE Signal Processing Letters*, 23(5):747–751, 2016.

[79] A. Kaplan and R. Tichatschke. Proximal methods in view of interior-point strategies. *Journal of Optimization Theory and Applications*, 98(2):399–429, 1998.

[80] A. Karpathy. Convolutional neural networks: Architectures, convolution / pooling layers. In *CS231n: Convolutional Neural Networks for Visual Recognition.* Stanford CA, 2019.

[81] J. B. Keller. Inverse problems. *The American Mathematical Monthly*, 83(2):107–118, 1976.

[82] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[83] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[84] K. C. Kiwiel. Convergence and efficiency of subgradient methods for quasiconvex minimization. *Mathematical programming*, 90(1):1–25, 2001.

[85] R. Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Montreal, Canada, 1995.

[86] N. Komodakis and J.-C. Pesquet. Playing with duality: An overview of recent primal-dual approaches for solving large-scale optimization problems. *IEEE Signal Processing Magazine*, 32(6):31–54, Nov. 2014.

[87] D. Krishnan and R. Fergus. Fast image deconvolution using hyper-laplacian priors. In *Advances in neural information processing systems*, pages 1033–1041, 2009.

[88] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[89] R. L. Lagendijk and J. Biemond. *The Handbook of Image and Video Processing.* Academic Press, New-York, N. J., USA, 2005.

[90] Y. LeCun et al. Generalization and network design strategies. In *Connectionism in perspective*, volume 19. Citeseer, 1989.

[91] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[92] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceeding of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 105–114, Honolulu, HW, USA, 21–26 Jul 2017.

[93] A. Levin, Y. Weiss, F. Durand, and W. T. Freeman. Understanding and evaluating blind deconvolution algorithms. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1964–1971, Miami, FL, USA, 20-25 Jun 2009.

[94] K. Li and J. Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

[95] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

[96] S. Mallat. *A wavelet tour of signal processing.* Elsevier, 1999.

[97] M. Mardani, H. Monajemi, V. Papyan, S. Vasanawala, D. Donoho, and J. Pauly. Recurrent generative adversarial networks for proximal learning and automated compressive image recovery. *arXiv preprint arXiv:1711.10046*, 2017.

[98] M. T. McCann, K. H. Jin, and M. Unser. Convolutional neural networks for inverse problems in imaging: A review. *IEEE Signal Processing Magazine*, 34(6):85–95, 2017.

[99] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[100] T. Meinhardt, M. Moller, C. Hazirbas, and D. Cremers. Learning proximal operators: Using denoising networks for regularizing inverse imaging problems. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1781–1790, Venice, Italy, 22–29 Oct 2017.

[101] C. E. Metz. Basic principles of roc analysis. In *Seminars in nuclear medicine*, volume 8, pages 283–298. Elsevier, 1978.

[102] T. M. Mitchell et al. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37):870–877, 1997.

[103] J.-J. Moreau. Fonctions convexes duales et points proximaux dans un espace hilbertien. *Comptes Rendus de l'Académie des Sciences (Paris) Série A*, 255:2897–2899, 1962.

[104] H. Mülthei. Iterative continuous maximum likelihood reconstruction methods. *Mathematical Methods in the Applied Sciences*, 15:275–286, 1993.

[105] H. Mülthei and B. Schorr. On properties of the iterative maximum likelihood reconstruction method. *Mathematical Methods in the Applied Sciences*, 11:331–342, 1989.

[106] N. Murata, K.-R. Müller, A. Ziehe, and S.-i. Amari. Adaptive on-line learning in changing environments. In *Advances in Neural Information Processing Systems*, pages 599–605, 1997.

[107] Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course.* Springer Publishing Company, Incorporated, 1 edition, 2014.

[108] Y. E. Nesterov. A method for solving the convex programming problem with convergence rate $o(1/k^2)$. *Dokl. Akad. Nauk SSSR*, 269:543–547, 1983.

[109] J. M. Ortega and W. C. Rheinboldt. *Iterative solution of nonlinear equations in several variables*, volume 30. Siam, 1970.

[110] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[111] F. Porta, M. Prato, and L. Zanni. A new steplength selection for scaled gradient methods with application to image deblurring. *Journal of Scientific Computing*, 65(3):895–919, 2015.

[112] A. Ramadhan, F. Mahmood, and A. Elci. Image denoising by median filter in wavelet domain. *arXiv preprint arXiv:1703.06499*, 2017.

[113] H. Robbins and S. Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[114] R. T. Rockafellar, R. J.-B. Wets, and M. Wets. *Variational Analysis*, volume 317 of *Grundlehren der Mathematischen Wissenschaften.* Springer, Berlin, 1998.

[115] L. Rokach and O. Z. Maimon. *Data mining with decision trees: theory and applications*, volume 69. World scientific, 2008.

[116] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[117] S. Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[118] L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D*, 60(1–4):259–268, 1992.

[119] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[120] A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

[121] O. Scherzer. The use of Morozov's discrepancy principle for Tikhonov regularization for solving nonlinear ill-posed problems. *Computing*, 51(1):45–60, 1993.

[122] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[123] U. Schmidt and S. Roth. Shrinkage fields for effective image restoration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2774–2781, Columbus, OH, USA, 24–27 Jun 2014.

[124] C. Schuler, H. C. Burger, S. Harmeling, and B. Schölkopf. A machine learning approach for non-blind image deconvolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1067–1074, Portland, OR, USA, 23–28 Jun 2013. IEEE.

[125] C. J. Schuler, M. Hirsch, S. Harmeling, and B. Scholkopf. Learning to deblur. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(7):1439, 2016.

[126] L. A. Shepp and Y. Vardi. Maximum likelihood reconstruction for emission tomography. *IEEE Transactions on Medical Imaging*, 1(2):113–122, 1982.

[127] J. Sun, H. Li, Z. Xu, and Y. Yang. Deep ADMM-Net for compressive sensing MRI. In *Proceedings of Advances in Neural Information Processing Systems*, pages 10–18, Barcelona, Spain, 5–10 Dec 2016.

[128] J. Sun and Z. Xu. Color image denoising via discriminatively learned iterative shrinkage. *IEEE Transactions on Image Processing*, 24(11):4148–4159, 2015.

[129] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[130] R. S. Sutton, A. G. Barto, et al. *Introduction to reinforcement learning*, volume 2. MIT press Cambridge, 1998.

[131] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[132] A. Tikhonov. Solution of incorrectly formulated problems and the regularization method. *Soviet Mathematics Doklady*, 4:1035–1038, 1963.

[133] A. Tikhonov and V. Arsenin. *Solution of Ill Posed Problems.* Wiley, New York, 1977.

[134] T. B. Trafalis, T. A. Tutunji, and N. P. Couëllan. Interior point methods for supervised training of artificial neural networks with bounded weights. In P. M. Pardalos, D. W. Hearn, and W. W. Hager, editors, *Network Optimization*, pages 441–470, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[135] V. Vapnik. *Estimation of Dependences Based on Empirical Data: Springer Series in Statistics (Springer Series in Statistics).* Springer-Verlag, Berlin, Heidelberg, 1982.

[136] V. Vapnik. Principles of risk minimization for learning theory. In *Advances in neural information processing systems*, pages 831–838, 1992.

[137] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity*, pages 11–30. Springer, 2015.

[138] C. R. Vogel. *Computational methods for inverse problems*, volume 23. SIAM, 2002.

[139] S. Wang, S. Fidler, and R. Urtasun. Proximal deep structured models. In *Proceedings of Advances in Neural Information Processing Systems*, pages 865–873, Barcelona, Spain, 5–10 Dec 2016.

[140] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[141] M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688, 2011.

[142] M. H. Wright. Interior methods for constrained optimization. *Acta numerica*, 1:341–407, 1992.

[143] L. Xu and J. Jia. Two-phase kernel estimation for robust motion deblurring. In *Proceedings of the European Conference on Computer Vision*, pages 157–170. Springer, 5–11 Sep 2010.

[144] L. Xu, J. S. J. Ren, C. Liu, and J. Jia. Deep convolutional neural network for image deconvolution. In *Proceedings of Advances in Neural Information Processing Systems*, pages 1790–1798, Montreal, Canada, 8–13 Dec 2014.

[145] R. Zanella, P. Boccacci, L. Zanni, and M. Bertero. Efficient gradient projection methods for edge-preserving removal of poisson noise. *Inverse Problems*, 25(4):045010, 2009.

[146] L. Zanni, A. Benfenati, M. Bertero, and V. Ruggiero. Numerical methods for parameter estimation in poisson data inversion. *Journal of Mathematical Imaging and Vision*, 52(3):397–413, 2015.

[147] J. Zhang and B. Ghanem. ISTA-Net: Interpretable optimization–inspired deep network for image compressive sensing. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1828–1837, Salt Lake City, UT, USA, 18–22 Jun 2018.

[148] J. Zhang, J. Pan, W.-S. Lai, R. W. H. Lau, and M.-H. Yang. Learning fully convolutional networks for iterative non-blind deconvolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3817–3825, Honolulu, HW, USA, 21–26 Jul 2017.

[149] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang. Beyond a Gaussian denoiser: Residual learning of deep CNN for image denoising. *IEEE Transactions on Image Processing*, 26(7):3142–3155, 2017.

[150] K. Zhang, W. Zuo, S. Gu, and L. Zhang. Learning deep CNN denoiser prior for image restoration. In *Proceeding of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, Honolulu, HW, USA, 21–26 Jul 2017.

[151] Y.-T. Zhou and R. Chellappa. Computation of optical flow using a neural network. In *IEEE International Conference on Neural Networks*, volume 1998, pages 71–78, 1988.

[152] D. Zoran and Y. Weiss. From learning models of natural image patches to whole image restoration. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 479–486. IEEE, 6–13 Nov 2011.