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 XXXIII, Coordinatore Prof. Cristian Giardinà

Convolutional neural networks for inverse problems with pseudodifferential operators: an application to limited-angle tomography

Settore Scientifico Disciplinare MAT/08

Dottoranda

Dr. Mathilde Galinier

INdAM-DP-COFUND-2015

Tutore

Prof. Marco Prato

Anni 2017/2020

Acknowledgments

First, I would like to extend my deep appreciation to the INdAM committee, for believing in my abilities and giving me the great chance to be part of INdAM-DP-COFUND-2015[1]. This doctorate in Italy has been an undeniably fascinating adventure from many points of view, and I feel highly grateful for all what I had the opportunity to experience and learn during this period.

I wish to express my profound gratitude to my supervisor, Marco Prato, for his precious advice and guidance. I am thankful for the serene and trustful working relationship he established between us, allowing me to carry out this doctorate in the best conditions. I feel also grateful for all the projects, conferences and schools he gave me the opportunity to get involved in.

I am deeply indebted to Samuli Siltanen and Matti Lassas, from the University of Helsinki, for giving me the chance to collaborate with them, and work on this captivating CT project. I would like to express my profound admiration for their work, and my heartfelt gratitude for their help, receptivity and ingenious suggestions. I am also very grateful to Tatiana Bubba and Luca Ratti, with whom I had the pleasure to work. I thank them for their valuable guidance, their availability and reactivity each time I had doubts or questions, and the warm reception they provided me during my research period in Finland. Thanks should also go to all the people I met at the University of Helsinki who made that visit period a delightful adventure.

I would like to thank all the people I was fortunate to spend time with during those years at the UNIMORE. Special thanks to my wonderful office mates and my dear lunch break companions: Lorenzo, Carla, Salvatore, Elena, Simone and Delfina.

This three-year Italian experience would not have been the same without the amazing friends I have met. In particular, I would like to express my deep gratitude to Federico, Davide and Gianmaria, who have provided me with an invaluable support when I most needed it. Their solicitude, enthusiasm and creativity have been, and keep being, precious and inspiring. Amongst the people I feel very happy to have met, I also would like to thank my football team and coach, who contributed in a different, but no less valuable way, to making my sojourn in Modena an unforgettable experience.

I would like to address my deep and heartfelt thanks to my French friends for their relentless support, their humour, their wisdom and their attentiveness, and especially Clem, Alex, Soso, Boudu, Chloé, Juju, Namalia and Titi.

[1]INdAM Doctoral Programme in Mathematics and/or Applications Cofunded by Marie Sklodowska-Curie Actions, grant number 713485.

Last but not the least, I would like to express all my thankfulness to my family, for their love, encouragement and patience. I dedicate this thesis to my parents, Isabelle and Bruno, and my sister, Lucille, who have never ceased to take care of me.

Abstract

Computed tomography (CT) ranks amongst the most popular, non-invasive medical imaging techniques. It makes use of computer-processed combinations of X-ray measurements, taken from different angles, in order to provide a detailed reconstruction of the inner structure of the body. The focus of this thesis is a particular setup of CT, called *limited-angle* CT (LA-CT), where the X-ray measurements are restricted to a small angular range. This task, which arises naturally in a number of important, practical applications such as breast tomosynthesis or dental X-ray scanning, presents the advantage of lowering the X-ray radiation dose and reducing the scanning time, and has thus become one of the predominant research topics in CT.

The main challenge inherent to LA-CT comes from the fact that the incompleteness of the collected data makes the reconstruction problem extremely ill-posed. As a consequence, classical methods, such as the *filtered-backprojection* (FBP), show poor performance and result most frequently in undesirable artefacts. Iterative algorithms as well as machine learning approaches have also been proposed in the literature to address this problem, but they still present drawbacks and limitations.

The purpose of this thesis is to propose a novel algorithm that combines both the trustworthiness of traditional, variational methods, and the powerful technology of deep learning strategies, in order to provide stable and reliable LA-CT reconstructions. More generally, the convolutional neural network (CNN) presented in this work, called ΨDONet, is designed to learn pseudodifferential operators (ΨDOs) in the broad context of linear inverse problems. Thus, although in this thesis, ΨDONet is mainly investigated in the special case of LA-CT, the theoretical results presented here can be extended to the broader case of convolutional Fourier integral operators (FIOs) and ΨDOs.

We formulate the LA-CT reconstruction problem as a regularised optimisation problem, in which the objective function to be minimised is the sum of a data-fidelity measure and a regularisation term that promotes sparsity in the wavelet basis. A well-known iterative technique for the solution of such a problem is the Iterative Soft-Thresholding Algorithm (ISTA) which, in the case of LA-CT, involves a ΨDO at each of its iterations. The convolutional nature of this very operator makes it possible to implement the unfolded iterations of ISTA as the successive layers of a CNN. We show, furthermore, that it is possible to compute the exact values of the parameters of the CNN in such a way that it reproduces the behaviour of standard ISTA, or a perturbation thereof. The strength of ΨDONet thus rests upon the fact that its parameters can be initialised with such values, and then trained through a learning

process made particularly efficient thanks to the CNN technology.

Two implementations of ΨDO-Net are investigated: Filter- Based ΨDONet (ΨDO-Net-F), where the pseudodifferential operator is approximated by means of a set of filters, whose central part is trainable; and Operator-Based ΨDONet (ΨDONet-O), where the pseudodifferential operator is not approximated but explicitly computed, and the learnable parameters are implemented as an additional operator. Numerical tests are conducted on different datasets of simulated data from limited-angle geometry. Both implementations provide similarly good and noteworthy results that clearly outperform the quality of standard ISTA reconstructions, the main difference being a greater computational efficiency for ΨDONet-O.

The presented approach offers promising perspectives and paves the way to applying the same idea to other convolutional FIOs or ΨDOs.

Sintesi

La tomografia computerizzata (CT) si colloca tra le tecniche di imaging medico non invasivo più diffuse. Si basa su combinazioni elaborate al computer di misurazioni a raggi X, prese da diverse angolazioni, al fine di fornire una ricostruzione dettagliata della struttura interna del corpo. Il fulcro di questa tesi è una particolare configurazione della CT, chiamata CT *ad angolo limitato* (LA-CT), dove le misurazioni dei raggi X sono limitate ad un piccolo intervallo angolare. Questa, che si presenta naturalmente in una serie di applicazioni pratiche come la tomosintesi mammaria o la radiografia dentale a raggi X, offre il vantaggio di abbassare la dose di radiazioni di raggi X e di ridurre il tempo di scansione, ed è così diventato uno dei temi di ricerca predominanti in CT.

La sfida principale intrinseca alla LA-CT deriva dal fatto che l'incompletezza dei dati raccolti rende il problema della ricostruzione estremamente mal posto. Di conseguenza, i metodi classici, come la *filtered-backprojection* (FBP), presentano scarse prestazioni e si traducono solitamente in artefatti indesiderati. Algoritmi iterativi e approcci di apprendimento automatico sono anche stati proposti in letteratura per affrontare questo problema, ma presentano ancora svantaggi e limitazioni.

Lo scopo di questa tesi è di proporre un nuovo algoritmo, che combina sia l'affidabilità dei metodi variazionali tradizionali, sia la potente tecnologia delle strategie di apprendimento profondo, al fine di fornire ricostruzioni LA-CT stabili e attendibili. Più generalemente, la rete neurale convoluzionale (CNN) presentata in questo lavoro, chiamata ΨDONet, è progettata per apprendere gli operatori pseudodifferenziali (ΨDOs) nel contesto generico dei problemi inversi lineari. Pertanto, sebbene in questa tesi, ΨDONet sia prinpipalmente indagato nel caso speciale di LA-CT, i risultati teorici qui presentati possono essere estesi al caso più ampio degli operatori integrali di Fourier (FIOs) e ΨDOs convoluzionali.

Formuliamo il problema di ricostruzione LA-CT come un problema di ottimizzazione regolarizzato, in cui la funzione obiettivo da minimizzare è la somma di una misura di fedeltà ai dati e un termine di regolarizzazione che promuove la sparsità nella base della wavelet. Una tecnica iterativa nota per la soluzione di tale problema è l'Iterative Soft-Thresholding Algorithm (ISTA) che, nel caso di LA-CT, coinvolge un ΨDO a ciascuna delle sue iterazioni. La natura convoluzionale di questo operatore consente di implementare le iterazioni 'srotolate' di ISTA come i successivi strati di una CNN. Mostriamo, inoltre, che è possibile calcolare i valori esatti dei parametri della CNN in modo tale da riprodurre il comportamento di standard ISTA, o una sua perturbazione. Il punto di forza di ΨDONet risiede quindi nel fatto che i suoi parametri

possono essere inizializzati con tali valori, e poi addestrati attraverso un processo di apprendimento reso particolarmente efficiente grazie alla tecnologia dei CNNs.

Vengono indagate due implementazioni di ΨDO-Net: Filter- Based ΨDONet (ΨDO-Net-F), dove l'operatore pseudodifferenziale è approssimato mediante un insieme di filtri, la cui parte centrale è addestrabile; e Operator-Based ΨDONet (ΨDONet-O), dove l'operatore pseudodifferenziale non è approssimato ma esplicitamente calcolato, e i parametri apprendibili sono implementati come un operatore aggiuntivo. I test numerici vengono condotti su diversi set di dati simulati con geometria ad angolo limitato. Entrambe le implementazioni forniscono risultati simili e degni di nota, che superano nettamente la qualità delle ricostruzioni ISTA standard, essendo la differenza principale una maggiore efficienza computazionale per ΨDONet-O.

L'approccio presentato offre prospettive promettenti ed apre la strada per applicare la stessa idea ad altri FIOs o ΨDOs convoluzionali.

List of Symbols

$\odot$	Hadamard product
$\otimes$	product measure
$\sharp S$	number of elements in set S
$\lfloor . \rfloor$	floor function
$\lvert s \rvert$	absolute value of scalar s
$\langle . \, , . \rangle$	inner product in $\mathbb{R}^n$
$\lVert M \rVert$	spectral norm of matrix M
$\lVert . \rVert_p$	p-norm in L^p
$\lVert . \rVert_H$	norm of the Hilbert space H
A^*	adjoint of operator A
$\bar{c}$	conjugate of complex element c
$\mathbb{E}(X)$	mathematical expectation of variable X
$\mathcal{F}, \mathcal{F}^{-1}$	direct and inverse Fourier transforms, respectively
$\nabla_x f(x_0)$	gradient vector of function f w.r.t. x at x_0
I_m	square identity matrix in $\mathbb{R}^{m \times m}$
$\ell_0(\mathbb{N})$	space of sequences with a finite number of non-zero values in $\mathbb{N}$
$\ell_1(\mathbb{N})$	space of sequences whose series is absolutely convergent in $\mathbb{N}$
$\ell_2(\mathbb{N})$	Hilbert space of square-summable sequences in $\mathbb{N}$
$L^1(\Omega)$	space of Lebesgue integrable functions on Ω
$L^2(\Omega)$	Hilbert space of square Lebesgue integrable functions on Ω
M_{ij}	element in the ith row and jth column of matrix M
M^T, M^{-1}	transpose and inverse of matrix M, respectively
$\mathcal{N}(\mu, \sigma^2)$	Gaussian distribution with μ and variance σ^2
$\mathbb{N}, \mathbb{N}^*$	sets of positive and non-zero positive integers
$\mathbb{R}^{m \times n}$	set of real matrices with m rows and n columns
$\mathbb{R}^n$	set of real vectors with n entries
S^1	unit sphere of $\mathbb{R}^n$
$\mathcal{U}(a, b)$	uniform distribution in $[a, b]$
$\omega^\perp$	vector orthogonal to the vector ω

Acronyms

ΨDO	pseudodifferential operator
ΨDO-F	Filter-based ΨDONet
ΨDO-O	Operator-based ΨDONet
n-D	n-dimensional
AI	artificial intelligence
ALISTA	analytical LISTA
ANN	artificial neural networks
ART	algebraic reconstruction technique
ASD-POCS	adaptive-steepest-descent-POCS
CAT	computerised axial tomography
CNN	convolutional neural networks
CRF	conditional random field
CS	compressed sensing
CT	computed tomography
DNA	deoxyribonucleic acid
DNN	deep neural networks
EM	expectation maximisation
ERM	empirical risk minimisation
FBP	filtered back-projection
FIO	Fourier integral operator
GAN	generative adversarial networks
GD	gradient descent
Haar-PSI	Haar wavelet-based perceptual similarity index
HO	hyperparameter optimisation
ICD	iterative coordinate descent
ISTA	iterative soft-thresholding algorithm

LA-CT	limited-angled CT
LISTA	learned ISTA
LS	least squares
LSCG	LS conjugate gradient
MART	multiplicative ART
MBIR	model-based iterative reconstruction
ML	maximum likelihood
MLP	multilayer perceptron
MRF	Markov random field
MRI	magnetic resonance imaging
MSE	mean squared error
ODE	ordinary differential equation
ODL	operator discretisation library
OS	ordered subsets
OSC	OS convex
PCA	principal component analysis
POCS	projection onto convex sets
PSNR	peak signal-to-noise ratio
RE	relative error
ReLU	rectified linear unit
RIP	restricted isometry property
RNN	recurrent neural network
SAR	synthetic-aperture radar
SART	simultaneous ART
SGD	stochastic GD
SIRT	simultaneous iterative reconstruction technique
SSIM	structural similarity measure
SVM	support vector machines
SVMC	SVM for classification
SVMR	SVM for regression
TD	temporal difference
TGV	total generalised variation
TISTA	trainable ISTA
TV	total variation
w.r.t.	with respect to

Contents

1 Introduction

Computed tomography (CT), also known as Computerised Axial Tomography (CAT) scanning, was historically the first method enabling to non-invasively acquire images of the inner structure of an object. In practice, it makes use of a number of X-ray measurements, taken from different angles, and combines the resulting attenuation profiles in such a way as to recover the internal features of the object of interest. This breakthrough, made possible by the discovery of X-rays and the advent of modern computers, opened up a whole host of new prospects in the scientific community. In particular, it revolutionised diagnostic medicine, allowing for the first time to get clear and accurate pictures of the inside of the human body without surgical intervention. Nowadays, CT is routinely used in medicine for diagnostic purpose and, despite the existence of competing methods such as magnetic resonance imaging (MRI), it is still to date the most widely used imaging technology in hospital departments and trauma clinics [31].

The progress allowed by the development of CT was not limited to radiology and on the contrary, has extended to many non-medical application domains. Inasmuch it allows the measurement of a specimen's geometrical dimensions (of both external and internal features), CT has proven to be of particular use for industrial nondestructive testing. As a result, it nowadays contributes to geometric analysis and dimensional inspection for technology companies spanning a variety of industries such as automotive, aerospace, electronics, and metalworking [207]. Additionally, anthropomorphic, forensic and archaeological as well as palaeontological applications of CT have also been developed over the past decades [31].

As of today, X-ray based imaging techniques have been the subject of almost 130 years of active research, starting from W. C. Röntgen's findings about X-rays and their properties in 1895 up to the present. Yet, research in the field of CT is still as exciting as at the beginning of its development during the 1960s and 1970s. In particular, the challenging task of image reconstruction from *limited data*, for a long time believed to inevitably generate artefacts throughout the reconstructed image, aroused a renewed interest in the early 21^{th} century, when the first examples of accurate partial reconstructions from incomplete data appeared. This turning point brought about

new mathematical problems as well as the potential for enthralling new applications. It notably resulted in the possibility to reduce the scanning time and lower the X-ray radiation dose as well as to handle large object while maintaining a high resolution in the reconstructions [41].

The focus of this thesis is a particular subset of limited data problems, the so-called *limited-angle CT* (LA-CT), which is a classical problem from the early days of tomography [127]. It frequently appears in important applications, such as breast tomosynthesis [233, 160], dental X-ray scanning [125, 156], electron tomography [64], damage detection in concrete structures [95] and luggage scanners.

1.1 Historical perspective

The history of X-ray imaging starts in 1895 with the accidental discovery of a new kind of radiation by the German scientist W. C. Röntgen. While he was exploring the path of electrical rays passing from an induction coil through a partially evacuated glass tube, he noticed that a screen coated with fluorescent material was illuminated by the rays, although the tube was covered in black paper and the room was completely dark. He later realised that a number of objects could be penetrated by these rays and soon, conducted a series of experiments to understand and demonstrate the exciting properties of what would quickly be famous as *X-rays*, so named on account of their unknown nature [179, 180]. For his discoveries, Röntgen was awarded a Nobel Prize in 1901, the first one in the field of physics.

Inasmuch as it enabled the observation of internal features of a person without the necessity of surgery, the technique of making X-ray photographs rapidly met with success amongst the doctors and spread around the whole world. X-ray imaging devices for medical purposes started to be built and were improved over the years in order to obtain better and better 2-D images of the inside of the human body. However, the enthusiasm that the diagnostic possibilities of X-radiation had created progressively gave way to a realisation of the limitations of medical imaging methods in only two dimensions. In practice, initial radiographs contained the superimposition of the 2D shadows of all the 3D structures composing the object. The interpretation of the image was thus troublesome, as the level at which the shadows were situated could not be determined, and no precise information about a particular internal feature could be recovered. Therefore, many attempts were carried out to dissociate the superimposed shadows and offer clear reconstructions of cross-sections of the body. In the 1920s, the technique of *analog tomography* saw the light of day, in which the X-ray tube and the detector moved in synchrony to image a motionless patient. In that manner, the plane in focus could be sharply displayed while the out of focus planes where blurred by the motion of the tube and detector. Quite interestingly, many researchers from different countries investigated this technique without being aware of the studies already performed, which resulted in a large number of patent applications (Baese in 1915 [8], Bocage in 1921 [20], Portes and Chausse in 1922 [167] and Pohl in 1927 [165], to name but a few) and journal papers rediscovering similar concepts. As a consequence,

the technique of analog tomography became known under several names: it was respectively called *laminography* by J. Kieffer [120], *stratography* by A. Vallebona [201] and *planigraphy* by A.E.M. Bocage and B.G. Ziedses des Plantes [58]. It is the term *tomography*, from the the Greek words *tomos* (slice) and *graphia* (writing, drawing), widely inspired by Grossmann's tomograph [84], that was ultimately retained.

In spite of the numerous innovations that followed the discovery of X-rays, the real rise of CT was not possible before the advent of modern computers, during the second half of the 20^{th} century. The first person credited for the invention of the actual CT was Allan Cormack, at the time medical physicist in South Africa. In the early 1960s, he posited that the internal structure of a body could be pieced together by taking X-rays images of it from multiple directions. To give a proof of concept, he built a prototype scanner and developed two algorithms for CT reconstruction [44, 45]. These results remained relatively unnoticed for several years, but surfaced and were acknowledged as prescient after the creation of the first clinical CT scanner. The latter was built and then patented in 1968 by Sir Godfrey N. Hounsfield [103], engineer in the British firm EMI, who, unaware of Cormack's contributions, had developed his own approach to the problem of image reconstruction. In 1971, an improved prototype scanner, at the time known as EMI-scanner, was installed at the Atkinson's Morley's Hospital in Wimbledon, and shortly after, the first tomographic examination of patient took place: on the image obtained, the round, dark pathological areas of the patient's brain, suspected to have a tumor, could be clearly distinguished from the healthy areas. By the end of 1973, the first commercial CT scanner was on the market and Hounsfield, along with Cormack, received the 1979 Nobel Prize in Medecine. During the subsequent decades, several generations of CT scanners were designed: *fan-beam* CT scanners appeared in 1976, followed by the *spiral* or *helical* CT scanners in 1989 and superseded in the early 1990s by the *cone-beam* and *multislice* CT scanners. Nowadays, CT images are also used with other modalities, such as SPECT-CT and PET-CT.

From its early stages to these days, CT technology has experienced an uncommonly fast progress. By way of example, contemporary CT scanners can scan in a few hundred milliseconds and reconstruct an image of 2048×2048 pixels. An achievement that would have sounded absolutely unattainable in the early 1970s: Hounsfield's first scanner needed up to 4.5 min to reconstruct a 80×80 image.

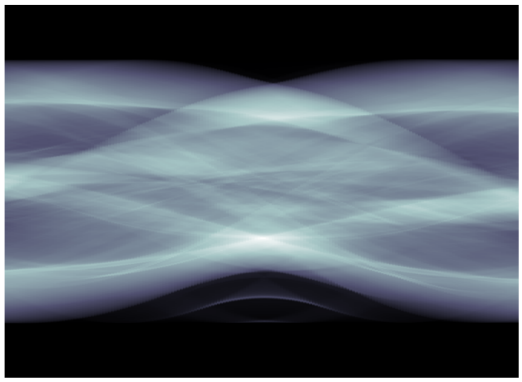
However, there are still numerous active research directions to explore. In particular, current predominant research topics in CT include the lessening of the X-ray radiation and time exposure, the reduction of the slice thickness for scanning and the further development of 3-D image reconstruction and visualisation.

More information about the history of CT can be found, *inter alia*, in [31, 40, 197, 96].

1.2 The fundamental mathematics of CT

The problem of CT consists essentially in reconstructing an object from its shadows, also named *projections*. The fundamental concepts at its origin stem from the ob-

(a) Axial CT slice.



(b) Simulated sinogram.

Figure 1.1: Example of a CT slice from DeepLesion dataset [224] and a simulated, noise-free sinogram thereof. In this simulation, the sinogram is composed of 1000 projections, within the full angular range (0-179°).

servation that different tissues, materials, absorb X-ray in a different way and thus, result in more or less dark shadows. In practice, the measurement of such projections is made possible by the synchronised use of an X-ray source and an array of X-ray detectors, positioned opposite each other in a donut-shaped structure (*gantry*). In clinical CT, the patient is placed on a bed, located in the opening of the gantry and kept stationary while the tube and the detector array rotate simultaneously around it. For a number of rotation angles, the CT apparatus shoots X-ray beams through the body and collects the attenuated X-rays that, once gathered, constitute the so-called *sinogram*. The latter, as it can be observed in fig. 1.1b, is not medically interpretable as it is and needs to be processed in order to recover an approximation of the inner structure of the object to image. This scenario belongs to the class of *inverse problems* and the search of its solution, generally complex, involves techniques of mathematics, physics and computer science [14, 97].

To solve this problem, the main variable of interest is the *linear attenuation coefficient* u, which describes the local X-ray absorbing capacity of the object. Since different anatomical structures have different densities, and thus different attenuation coefficients, the knowledge of u would directly lead to the knowledge of the inner structure of the object.

In physical terms, the photons contained in the X-ray beams are partially absorbed by the object, according to the density of the matter they encounter. The Beer-Lambert law [155] establishes a relation of proportionality between the linear attenuation coefficient u and the X-ray intensity measured by the detector through a formula whose integration reads:

$$\ln\left(\frac{I_0}{I_1}\right) = \int_{x\in\ell} u(x)\mathrm{d}x, \tag{1.1}$$

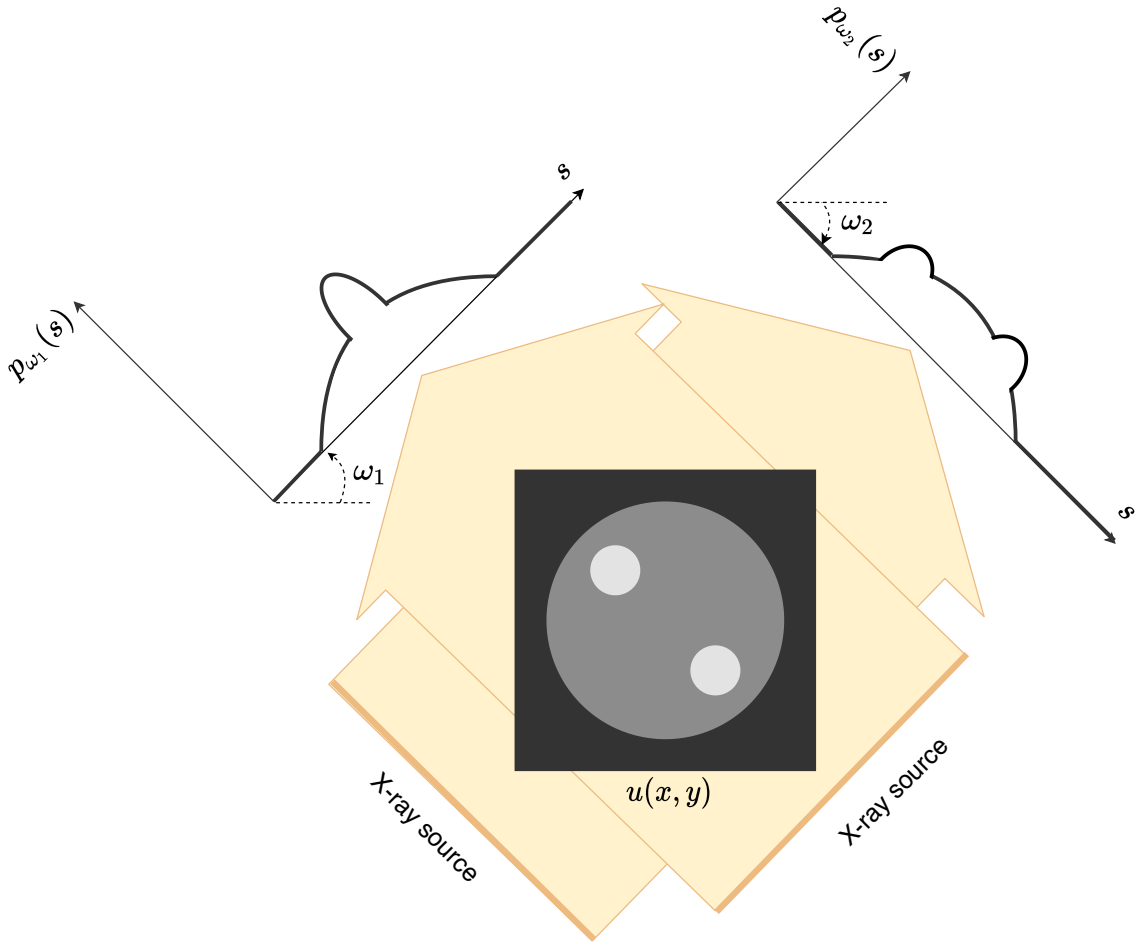


Figure 1.2: Schematic illustration of computed tomography (CT). A object, or phantom, $u(x, y)$ is exposed with X-ray under the projection angles ω_1 and ω_2. Each projection angle produces a specific shadow $p_\omega(s)$, which, measured with the detector array, represents the integral X-ray attenuation profile. Mathematically, this shadow is modelled by the Radon transform: $p_\omega(s) = Ru(s, \omega)$ where R is as described in eq. (1.3).

where I_0 is the intensity at the X-ray emitter, I_1 is the intensity at the detector, and ℓ is the line along which X-rays travel. As the sinogram contains all the measured information about the left member of eq. (1.1), the inverse problem of CT comes down to recovering u from the knowledge of its line integrals.

A detailed, mathematical basis to the solution of this problem had already been published in 1917 by the Austrian mathematician Johann Radon [169]. Due to the depth and complexity of the publication and the fact that it was written in German, however, the consequences of his pioneering work were revealed only many decades later [31]. In his paper, Radon defined the *Radon transform* as being the integral of a function along a particular line. By describing the line of interest ℓ as:

$$\ell(s, \omega) = \{x \in \mathbb{R}^2 \ : \ \langle x, \omega \rangle = s\}, \qquad s \in \mathbb{R}, \omega \in S^1, \tag{1.2}$$

the Radon Transform reads:

$$Ru(s,\omega) = \int_{x\in\ell(s,\omega)} u(x)\mathrm{d}x = \int_{-\infty}^{+\infty} u(s\omega^{\perp} + t\omega)\mathrm{d}t, \qquad s \in \mathbb{R}, \omega, \omega^{\perp} \in S^1, \tag{1.3}$$

where $\omega^{\perp}$ denotes the vector in the unit sphere S^1 obtained by rotating ω counterclockwise by 90° [127]. As part of its ground-breaking results, Radon had shown that, in the continuous setup, it is possible to exactly retrieve u from $R(u)$ thanks to the *Radon inversion formula*. This analytical method, also known in the imaging domain as the *filtered back-projection* (FBP) algorithm, provides very satisfactory reconstructions in the case of *complete tomographic data*, that is, when data are collected on a fairly even distributed set of lines throughout the object. However, it rapidly becomes impractical when dealing with limited tomographic data, such as *sparse-view CT*, where the number of measured projections is low, or LA-CT, in which the projections are acquired only over a limited angular range. In those cases, FBP fails to produce a good reconstruction and leads to the appearance of streaking artefacts [152].

In order to better understand the singularities, *i.e.* the sharp features of u that can be stably recovered based on a partial knowledge of $R(u)$, microlocal analysis proves particularly helpful [127]. The latter can notably be used to read the part of the wavefront set of the target corresponding to the missing angular range from the measurement geometry. The field of microlocal analysis, which appeared in the 1960s as part of the study of linear partial differential equations, led to the emergence of different theories, amongst which the theory of pseudodifferential operators (ΨDOs), defined by Nirenberg and Kohn in 1965 [124], and Fourier integral operators (FIOs), introduced by Hörmander in 1971 [108]. More information about ΨDOs and FIOs can be found in Appendix A.

In the context of complete data CT, microlocal analysis can be used to show (see, *e.g.* [168]) that the normal operator R^*R of the Radon Transform is an elliptic pseudodifferential operator of order -1 and a convolutional operator associated with the Calderón-Zygmund kernel:

$$K(x,y) = \frac{1}{|x-y|}, \qquad \text{for } x \neq y. \tag{1.4}$$

When it comes to LA-CT, the normal operator $R_\Gamma^* R_\Gamma$ of the limited-angle Radon transform R_Γ, where Γ stands for the limited angular range $[-\Gamma, \Gamma]$, is no longer a ΨDO, but belongs to the wider class of FIOs, which includes operators associated with a kernel showing discontinuities along lines [109]. In addition, $R_\Gamma^* R_\Gamma$ is a convolutional operator associated with the kernel:

$$K(x,y) = \frac{1}{|x-y|}\chi_\Gamma(x-y), \qquad \text{for } x \neq y. \tag{1.5}$$

where χ_Γ denotes the indicator function of the cone in $\mathbb{R}^2$ between the angles $-\Gamma$ and Γ.

As it is further explained in chapter 3, the convolutional nature of the operator $R_\Gamma^* R_\Gamma$ is the cornerstone of the approach presented in this work. It indeed allows the

connection between the conventional interpretation of the normal operator of R_Γ and the theory of Convolutional Neural Networks (CNNs), presented in chapter 2.

Although this thesis will focus on the particular case of LA-CT and the corresponding Radon transform, it is important to note that the theoretical results presented thereafter can be extended to all the ΨDOs and FIOs meeting the specified conditions.

1.3 Inverse Problems and Variational Methods

Historically, CT images provided the first example of images obtained by solving a mathematical problem that belongs to the class of the so-called *inverse problems* [14]. Since then, and especially over the last decades, the field of inverse problems has experienced an exponential growth. Thanks to the availability of ever more powerful computers, and the development of reliable and fast numerical methods to perform the solution process, inverse problems received much attention from mathematicians and engineers who continually contributed to the broadening of the range of their applications. Nowadays, they appear in a wide variety of domains, which span from finance and economy to biomedecine and geophysics, astronomy and life science in general.

Inverse problems owe their denomination to Joseph B. Keller who, in his 1976 article [119] since then frequently quoted, stated:

> "We call two problems *inverse* of one another if the formulation of each involves all or part of the solution of the other."

This definition however engenders a certain ambiguity in the distinction between the so-called *direct problem* and its associated *inverse problem*, mainly due to the duality connecting the two: namely, one problem can be obtained from the other by interchanging the role of the data and that of the unknowns, so that it may seem arbitrary to determine which is the direct problem and which is the inverse one. For historical reasons, it was agreed that the direct problem would usually refer to the simpler one or the one that was studied earlier.

In many instances, the direct problem can be viewed as being oriented along a cause-effect sequence; its inverse problem then corresponds to the reversal of the cause-effect sequence and aims at identifying the unknown causes of known consequences. In other words, solving an inverse problem consists in using the outcomes of actual observations or indirect measurements to infer the model or an estimation of a number of parameters describing the system of interest. In practice, the physical process leading from the unknowns of the inverse problem to its data entails a loss of information and thus makes it generally particularly complex to solve.

By way of example, one of the direct problems in the field of classical mechanics is the computation of particle trajectories from the knowledge of the forces at work. Therefore, the corresponding inverse problem is the determination of the forces from the knowledge of the trajectories. Analogously, in radio astronomy, one aims at estimating the shape of celestial objects emitting radio waves from the waves detected by radio

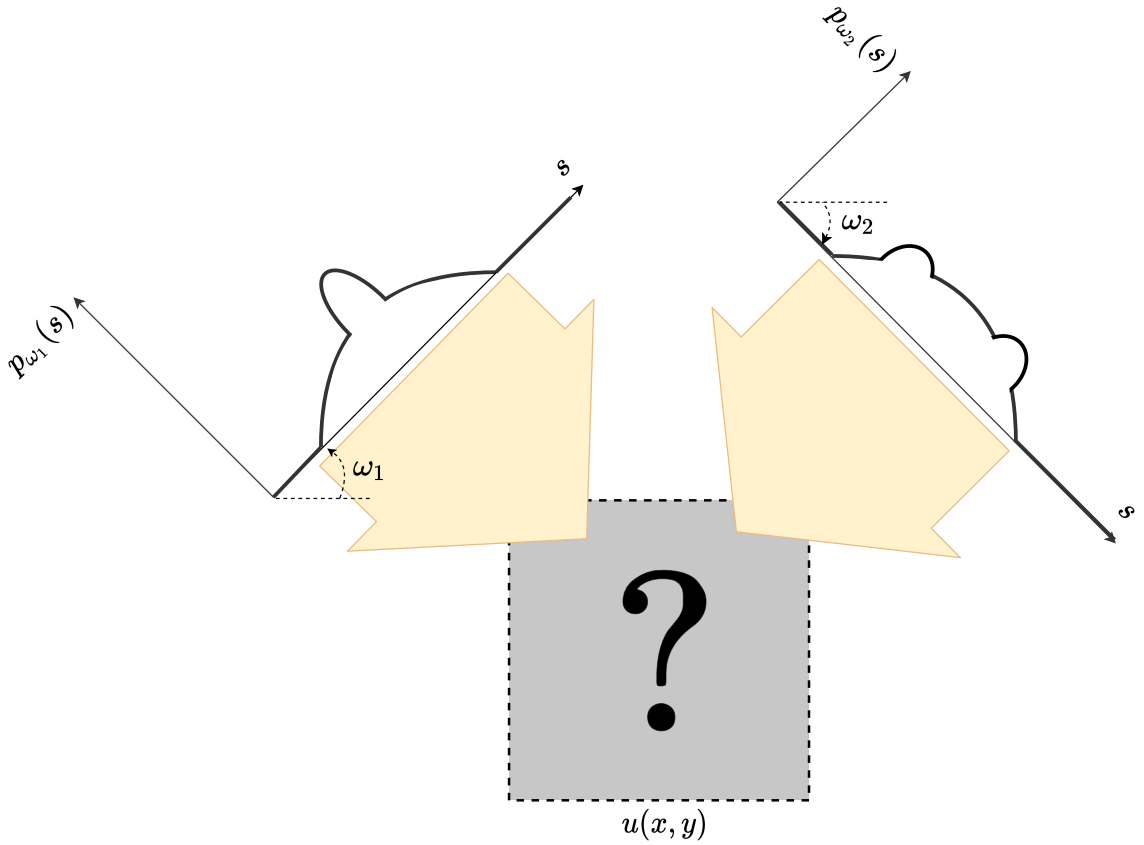


Figure 1.3: Schematic illustration of the inverse problem posed by CT. Attenuation profiles $p_\omega(s)$ have been measured for a set of projection angulations, ω_1 and ω_2. The unknown geometry, or the linear attenuation coefficient u, is then to be recovered from the set of attenuations profiles.

telescopes on the surface of Earth. In life sciences, classical inverse problems include inverse protein folding problems and in finance, one seeks to assess the volatility of the stock price, which expresses the degree of variation of a trading price series over time.

In CT, the meaning of the terms *direct* and *inverse problems* is immediately apparent. While the former describes the 'simple' task of measuring the sinogram of a given object, the latter consists in recovering the spatial structure of the object, not known *a priori*, from the acquired projections. Figures 1.2 and 1.3 respectively illustrate those situations.

In most of the cases, external factors affect the empirical measurements, which results in observations composed of approximate values: it is said that data are corrupted with *noise*. This phenomenon may bring about significant errors in the computed solution, and the estimation process then falls into the category of *ill-posed* problems. The concept of *ill-posedness*, as opposed to the concept of *well-posedness* introduced by the French mathematician Hadamard in [86], encompasses all the problems whose solution may not exist and/or might not be unique and/or might not depend continuously on the data. In the particular case of incomplete data CT, due to the (very) limited amount of acquired information as well as the unwanted fluctuations altering

the measurement process, the inverse problem proves to be severely ill-posed and all the more uneasy to solve as the narrowness of the angular range and/or the sparsity of the projections increase.

How to tackle ill-posedness is still an open research field and remains a tricky topic inasmuch as it is problem-dependent. Nevertheless, it rapidly became clear to the scientific community that, in order to optimally solve problems of this kind, it would be judicious to search for approximate solutions satisfying additional constraints coming from the physics of the problem. In other words, the lost information that, de facto, could not be derived from the collected data, should be added to the fullest extent possible, under the form of *a priori* knowledge expressing some physical properties of the object. With this goal in mind, the mathematician A.N. Tikhonov proposed a novel approach [198] in 1963, baptised *regularisation theory*, which has since become one of the most powerful tools for the solution of inverse problems. This technique aims at approximating an ill-posed problem by a family of neighbouring well-posed problems. Amongst other things, it allows to reduce the set of the reconstructions compatible with the data as well as to discriminate between interesting objects and erroneous ones, resulting from an uncontrolled propagation of the noise affecting the image.

Inverse and ill-posed problems as well as their regularised variational formulations have given rise to an extensive literature, including books, reference proceedings and dedicated journals. Seminal works include *inter alia* [14, 62, 83, 88, 209] and more references can be found therein. The more recent [184] offers an overview of inverse problems and their applications in the field of imaging science.

This framework provides a broad range of appealing methods for the study of limited data CT problems. In this thesis, we focus on a regularisation approach that promotes the sparsity of the reconstructed object when represented in the wavelet basis.

In concrete terms, we consider the mathematical model:

$$m = R_\Gamma u^\dagger + \epsilon, \tag{1.6}$$

where $m \in Y$ is the set of measurements gathered in the sinogram, $u^\dagger \in X$ is the sought image, $\epsilon \in Y$ stands for the perturbation noise and $R_\Gamma : X \to Y$ is the limited-angle Radon transform.

In order to find an adequate solution to the ill-posed problem arising from eq. (1.6), the sought signal can be approximated by the minimiser of a regularised objective function, expressed as the sum of a data-fidelity term, which measures the faithfulness of the solution to the observation model, and an additional regularisation term, which orients the reconstruction process towards a solution with some desired characteristics and improves stability to noise. In particular, one of the simplest and surprisingly effective way to remove noise is to expand the signal in an orthogonal wavelet basis and to enforce the sparsity of such expansion. The justification of such an approach

is that in the wavelet domain, the signal gets usually concentrated in a few significant coefficients while the noise is spread out in smaller and diffuse values that can be more easily suppressed. This *a priori* information can be mathematically incorporated to the model and results in the following variational minimisation problem:

$$\min_{w \in \ell^1(\mathbb{N})} \|R_\Gamma W^* w - m\|_Y^2 + \lambda \|w\|_{\ell^1}, \tag{1.7}$$

where $\lambda \in \mathbb{R}_+$ is the trade-off between the data-fidelity term and the regularisation term, and $W : X \to \ell^2(\mathbb{N})$ is the transform mapping any $u \in X$ to the sequence of its components with respect to the wavelet basis, in such a way that $Wu = w$. As it is straightforward to compute an image based on its wavelet representation and vice versa, the solution w of 1.7, once computed, provides an immediate and unambiguous solution to eq. (1.6).

A well-known reconstruction algorithm for the solution of eq. (1.7), dubbed Iterative Soft-Thresholding Algorithm (ISTA), was introduced in 2004 in the seminal paper by Daubechies, Defrise and De Mol [52]. It iteratively creates the sequence $\{w^{(n)}\}_{n=1}^N$ as follows:

$$w^{(n)} = \mathcal{S}_{\lambda/L}\left(w^{(n-1)} - \frac{1}{L}K^{(n)}w^{(n-1)} + \frac{1}{L}b^{(n)}\right), \tag{1.8}$$

where $K^{(n)} = WR_\Gamma^* R_\Gamma W^*$, $b^{(n)} = WR_\Gamma^* m$ and $\mathcal{S}_\beta(w)$ is the component-wise soft-thresholding operator. Further details of this algorithm, which greatly inspired our work, as well as a full description of the formulation of the inverse problem under consideration are given in chapter 3.

1.4 Machine Learning

The emergence of *Artificial Intelligence* (AI) in the last century allowed the development of brand-new techniques, said to be *data-driven*, as opposed to the class of *model-based* approaches to which belongs, for instance, the aforementioned ISTA. Those new methods indeed do not rely on the mathematical modelling of a problem but instead learn from their environment which actions to take in order to maximise their chance of success. The appearance of such a new strategy has undoubtedly opened up fascinating new perspectives in the field of inverse problems and more generally in a broad range of applications covering not only the areas of engineering, science and mathematics, but also medicine, business, finance and even literature [57].

The first operational definition of intelligence was provided by Allan Turing through the eponymous test [199] which allowed to assess a machine's ability to exhibit intelligent behaviour equivalent to, or indistinguishable from that of human. The capabilities that the computer would need to possess come down to: natural language processing, knowledge representation, automated reasoning and *machine learning*. The latter, which refers to the ability of a machine to acquire its own knowledge by extracting patterns from raw data, gave its name to one of the predominant fields of AI.

Basically, a machine learning system is designed to learn and improve with experience in such a way that it is able to make predictions or decisions without being

explicitly programmed to do so. For this purpose, the machine learning algorithm first experiences a set of *training* examples that come from some generally unknown probability distribution, and then builds a mathematical model that enables it to produce sufficiently accurate predictions on new cases or situations coming from that same probability distribution. While machine learning in general has been having an enormous impact on the recent scientific and industrial advances, the particular subset of *deep learning* methods have drawn even more attention.

The concept of deep learning originated from the computational modelling of biological learning, that is, the modelling of how learning happens or could happen in the brain. Inasmuch as they were engineered systems inspired by the biological brain, deep learning systems became known, *inter alia*, under the name of *Artificial Neural Networks* (ANNs). They consist of a collection of connected basic units, also called *nodes* or *neurons*, arranged in layers whose number allows the ANN to model functions with different degrees of complexity.

Although the field of deep learning has become recognised as a crucial technology only recently, its story actually goes back to the 1940s and has since then experienced several waves of development before reaching its current popularity. The seminal work on neural networks is credited to Warren McCulloch and Walter Pitts in their 1943 paper [146], where they showed that ANNs could, in principle, compute any arithmetic or logical function. The first practical application of ANNs, however, emerged only about fifteen years later, with the invention of the perceptron network and associated learning rule, by Frank Rosenblatt [176]. In spite of those exciting discoveries, learning algorithms were limited to quite simple problems and interest in ANNs faltered during the late 1960s, because of the lack of new ideas and potent computers with which to conduct the experiments. As both of these impediments were overcome in the 1980s, a second wave of development brought about new breakthroughs that reinvigorated neural networks. Amongst them, the back-propagation algorithm [177] provided an effective way to train multilayer perceptron networks and even today, remains the dominant approach to train ANNs. ANN research experienced a second decrease of popularity in the mid-1990s which lasted until 2006, with the invention of the actual deep learning. Until then, ANNs had been built with shallow-structured architectures (that is, containing very few layers) and consequently, could not surpass more traditional approaches, except for a few specialised problems. The possibility to aggregate more and more layers in an ANN, thus said to be *deep* neural networks (DNNs), constituted a veritable milestone. Subsequently, the availability of ever-more powerful computers and larger datasets, as well as the development of efficient techniques to train deeper and deeper networks, have contributed to the dramatic growth in popularity and usefulness of deep learning methods.

Since the 1980s, deep learning has been the subject of thousand of papers and has constantly been applied with success to broader and broader sets of applications. DNNs can now handle tasks as complex as self-driving cars, helping the diagnosis of life-threatening diseases such as cancer and automatically translating written or spoken contents into other languages, to name but a few. They are used by many cutting-edge technology companies, including Google, Facebook, Microsoft, Apple and Netflix.

More information about the history, functioning and applications of deep learning techniques can be found, *inter alia* in the books [178, 77, 90, 57, 3] and the references therein.

The wide-ranging list of applications in which DNNs have established themselves as an indispensable tool include the field of imaging inverse problems. In particular, the so-called *Convolutional Neural Networks* (CNNs) have shown outstanding results in different topics, such as denoising [232], deblurring [222], super-resolution [59] and even CT-reconstruction [115].

1.5 Our proposed method

In spite of their numerous alluring abilities, however, DNNs and CNNs present the drawback of a questionable explainability and reliability when used as mere black-boxes. Many research works have thus proposed to merge machine or deep learning techniques with more traditional, model-based methods. The resulting hybrid strategies would therefore allow to take advantage of the performance of the former, while benefiting from the trustworthiness of the latter.

Amongst the strategies so far explored in that direction, a straightforward and yet provably efficient way to connect variational models to deep learning techniques consists in unfolding an iterative algorithm into a network structure, by turning each iteration into a layer of the network. Deep unfolding has been applied *inter alia* to primal-dual solvers [171, 208, 210], bilevel optimisation [158], proximal interior point algorithm [15] and proximal gradient methods [122, 145, 225]. Other unrolling approaches have drawn inspiration from nonlinear diffusion processes, such as the network introduced in [36], based on a discrete Partial Differential Equation (PDE) solver, and the residual neural network proposed in [34], making use of a discrete Ordinary Differential Equation (ODE) solver. As far as probabilistic models are concerned, they also provide an interesting support for deep unfolding. Notable unrolled architectures encompass Markov Random Fields (MRFs) [98], Conditional Random Fields (CRFs) [139, 185, 234] and topic models [38], to name but a few. Further examples of recent methods employing algorithm unrolling in practical signal and image processing can be found in [151].

In our work, we propose a novel CNN architecture called ΨDONet [29], which is obtained by unfolding ISTA over a finite number of iterations. It is actually already well known that the unrolled iterations of ISTA can be considered as the layers of a neural network. Nevertheless, to the best of our knowledge, no approach has so far offered to implement the operator K in eq. (1.8) in such a way that each unfolded iteration can be considered as the layer of a CNN while allowing to recover standard ISTA for a specific choice of the parameters involved. Those two interesting properties implies that our algorithm can not only imitate the behaviour of standard ISTA, but also and above all, it can learn how to improve it, through a learning process made particularly efficient thanks to the CNN technology. The concrete development of such a strategy rests upon the convolutional nature of the operator K, whose implementation is thus perfectly compatible with the framework of CNNs. Our algorithm can be applied to

the special instance of the Radon transform as it is further investigated in this thesis, but it can also be extended to the broader case of convolutional FIOs and ΨDOs, whence its name.

The key feature of ΨDONet is the splitting of the convolutional kernel into $K = K_0 + K_1$, where K_0 is the known part of the model (in the limited angle case, K_0 stands for $R_\Gamma^* R_\Gamma$) and K_1 is an unknown ΨDO to be determined, or better, to be learnt. This splitting has mainly two interesting consequences. First, K_1 provides the potential to add information to the known (fixed) part of the model K_0 in the reconstruction process and as such, can be seen as an adjunct for the direct improvement of the back-projection operator. Secondly, the operators of ΨDONet offer the possibility to be encoded as a combination of upscaling, downscaling and convolution operations, as it is common practice in deep learning. Remarkably, we show that such operations can be exactly determined combining the convolutional nature of the limited angle Radon transform and basic properties defining an orthogonal wavelet system. All the parameters to be learnt can thus be initialised in such a way that ΨDONet before training is exactly equivalent to standard ISTA; the learning process then allows the modification of K_1 so that the output reconstructions present an ever improving image quality. While this may appear contrary to the machine learning philosophy, which finds its strength in avoiding any predefined structure for neural networks, our recipe imparts a veritable interpretability of the results of the proposed CNN without actually penalising its training process. At the same time, the possibility to deploy such operations with small convolutional filters enables a significant reduction of the parameters involved, specifically when compared to the standard interpretation of ISTA as a recurrent neural network: this is crucial when it comes to a practical numerical implementation of the proposed algorithm.

We provide two different implementations of ΨDONet: Filter-Based ΨDONet (ΨDO Net-F), where the back-projection operator is approximated by its filter-equivalent, and Operator-Based ΨDONet (ΨDONet-O), where the back-projection operator encoded in K_0 is not approximated but explicitly computed. Numerical experiments are then conducted on different sets of simulated data to validate the theoretical results.

1.6 State-of-the-art

From a historical perspective, the FBP algorithm was the first analytical method for CT reconstruction and it has remained one of the predominant approaches in the case of complete tomographic data. Nonetheless, when it comes to sparse-view or limited-angle CT, FBP provides reconstructions with streaking artefacts and view aliasing due to missing data. This is the reason why reconstruction with incomplete projections has attracted more and more interests and have finally resulted in a vast and still flourishing literature.

To overcome the limitations of analytical methods, and thanks to the ever-increasing growth of computer technology that enabled the employment of computationally heavier techniques, iterative reconstruction approaches for CT started to be elaborated.

One of the main reasons to prefer the latter over analytical methods is the possibility to complement the insufficient measurements by imposing *a priori* information on the solution. Incidentally, reconstruction of the very first clinical image was generated with an iterative algorithm, called Algebraic Reconstruction Technique (ART) [80] and based on the Kaczmarz method for solving linear systems of equations. Although at the time, ART did not involve a complex modelling of the CT system, it nevertheless demonstrated the efficacy of iterative techniques for X-ray CT. One variant of ART is the Simultaneous ART (SART) [5], which performs updates for complete raw data projections instead of processing a single pixel at the time. Compared to the original ART, SART reconstructed images are smoother and the stripe-shaped artefacts are better suppressed. Other ART-based methods were proposed, such as the Simultaneous Iterative Reconstruction Technique (SIRT) [75], its variant Ordered Subsets SIRT (OS-SIRT) [221] or else the Multiplicative Algebraic Reconstruction Technique (MART) [7]. Iterative algorithms also boasts statistical techniques, which can be subdivided into two groups: maximum likelihood (ML) principle based methods and least squares (LS) principle based methods. One of the most popular approaches among the ML class is undoubtedly the Maximum-Likelihood Expectation-Maximisation (ML-EM) algorithm [130]. It consists of two alternating steps: the E-step, which computes the expected log-likelihood function, and the M-step, which searches for the next estimate by maximising the previously computed function. Several variants of this approach were introduced with the aim to speed convergence and enable an easy parallelisation: the OS-EM [144] and Ordered Subset Convex Algorithm (OSC) [117, 63, 11], for instance. As far as the class of LS methods is concerned, noteworthy algorithms include the LS Conjugate Gradient (LSCG) algorithm [162, 163, 65], the iterative coordinate descent (ICD) [182, 25, 196] and its faster variant OS-ICD [134, 237]. Parallelising such methods however proves to be hard as single pixels or coordinates are iteratively updated to minimise a cost function. Finally, the family of iterative methods encompasses the model-based iterative reconstruction techniques (MBIR) [196, 229], in which the acquisition process is sought to be modelled as accurately as possible, by taking into account both photon statistics and geometry modelling. More detailed review and comparison of analytical and iterative reconstruction techniques can be found in [55, 12, 104].

The advent of the Compressed Sensing (CS) theory [32], which provided new techniques for finding solutions to under-determined linear systems, fostered the emergence of Total-Variation (TV) regularised iterative algorithms for limited data CT. Such algorithms make use of a sparsity prior on the image gradient. The wide literature relating to this strategy include, *inter alia* ASD-POCS [186], soft-thresholding TV [228], improved TV (iTV) [172], spatio-temporal TV (STTV) [195], anisotropic TV (aTV) [37] and reweighted aTV [211], scale space anisotropic TV (ssaTV) [106] and total generalised variation (TGV) [157]. Those techniques however are based on a piecewise constant image model and as such, they may lose some fine features and generate a blocky appearance in incomplete and noisy cases. The use of sparsity priors in other domains than the image domain has also been investigated. For instance, the approach introduced in [141] seeks for sparsity both through TV and wavelet tight

frame regularisations, while the work in [69] offers a sparse regularisation of curvelet coefficients as a reconstruction method.

Although the above-mentioned iterative techniques prove to be successful in a number of cases, they still present some limitations, such as the expensive time consumption required by the successive iterative steps and the complexity of the parameter selection.

Another class of techniques for limited data CT reconstruction that deserves mention is the class of inpainting-like approaches. Unlike the previously-mentioned methods which take place in the image domain, they focus on the interpolation or completion of the missing data in the projection domain. Noteworthy attempts in this field encompass sinusoid-like curves [136], which mainly consists in S-curve-based sinogram discretization, sinusoid fitting, and eigenvector-guided interpolation; PDE based inpainting [126], which considers diffusion based regularisers coupled with optical flow information; directional interpolation [16] which uses a structure tensor approach; to name but a few. Such approaches achieved enhanced image quality for certain scanning configurations and particular subjects only, but present limited performance when it comes to clinical applications.

More recently, the appearance of machine learning has brought a breath of fresh air to CT reconstruction methods. Among the new methods that have arisen, some, for example, form part of the dictionary learning theory, which aims at finding a sparse representation of the image in terms of a redundant dictionary, learnt from data [223]. But most importantly, it is the emergence of deep learning that marked a real turning point in the history of image reconstruction techniques. Several lines of investigation where considered to make the most of deep learning for insufficient data CT problems: artefact post-processing in the image domain and image transformed domains (image-to-image reconstruction) [218, 87, 51, 226], sinogram inpainting in the projection domain (sinogram-to-sinogram reconstruction)[74, 6, 138, 133, 60], and direct projection-to-image reconstruction (sinogram-to-image reconstruction)[236, 220, 70, 137]. While some approaches rest upon the exclusive use of deep DNNs such as the U-Net architecture [175] or generative adversarial networks (GANs) [78, 227], other techniques draw their strength from the merging of deep learning and variational approaches [2, 30, 105]. The latter present the advantage of a better explainability and reliability, as they make use of a mathematical understanding of the problem and involve a data-driven strategy only in a limited but thoughtful manner. A straightforward way to do so is to unfold an iterative algorithm into a network structure.

In the general context of sparse coding, Gregor and LeCun [81] explored the unfolding of ISTA (see [35] for theoretical linear convergence results) and offered to learn two matrices of standard ISTA instead of using pre-computed ones. Their proposed Learned ISTA (LISTA), which is aimed at calculating good approximations of optimal sparse codes, inspired many papers, all deriving from the unrolling of ISTA: the algorithm in [116] is intended to fine-tune the thresholding functions of standard ISTA thanks to DNNs; trainable ISTA (TISTA) [111] is based on an error variance estimator and learns only a very small number of parameters; deep ℓ_0 encoders [213] address an ℓ_0-regularisation of the minimisation problem; analytic LISTA (ALISTA) [140] is a neural network whose weights are not to be trained, but rather analytically calculated;

while ISTA-Net [231] is designed to learn a non-linear transform function to sparsify the image. Except for the two last ones, the above-mentioned algorithms are not CNNs, which make them essentially different from our approach. The convolutional version of ALISTA, conv ALISTA, offers to learn only the stepsize and threshold of each iteration, while the convolutional weights are analytically determined. It thus differs from ΨDONet, where the convolutional filters are to be trained, at least partially. As for ISTA-Net, it mainly aims at learning the transform defining the representation domain in which the image is sought to be sparse. In ΨDONet, we do not seek to learn such a transform, as we assume it to be the wavelet transform and instead, we seek to improve the back-projection operator in the wavelet domain. In [115], the authors investigate the relationship between CNNs and iterative optimisation techniques, including ISTA, for the case of normal operators associated with a forward model that is a convolution. Nevertheless, the resulting U-net, dubbed FBPConvNet, does not aim at imitating an unrolled version of an iterative method and is thus fundamentally different in spirit to the methodology we propose.

1.7 Thesis outline

This thesis consists of four chapters and three appendices. The present chapter serves as an introduction to this doctoral dissertation. It includes an extensive summary of the historical development of CT and a description of its mathematical principles. It furthermore contains a presentation of inverse problems as a general framework for the limited-angle CT problem and a historical description of machine learning. It finally presents the general outline of the proposed approach and offers a literature review on reconstruction strategies for the LA-CT problem.

The purpose of chapter 2 is to provide the reader with the fundamental notions of machine learning and deep learning, necessary for a proper understanding of the proposed algorithm. The first part of the chapter is dedicated to the description of the machine learning framework with a particular focus on the supervised learning setup. We *inter alia* explore concepts such as loss functions, risks, generalisation, and capacity, and provide a brief summary of existing regularisation techniques and gradient-based optimisation strategies. The second part of chapter 2 offers an overview of the theory of deep learning, and in particular, aims at presenting the functioning of the so-called *artificial neural networks*. The main notions subtending the learning process of the latter are investigated through the fundamental example of *multilayer perceptrons*. We also give a glimpse of the training challenges and optimisation strategies widespread in the domain of deep learning. Finally, the third part of that chapter deals with a particular kind of networks, called *convolutional neural networks*, which have shown remarkable results and efficiency in solving imaging problems, and are at the core of the proposed algorithm.

In chapter 3, we review the theoretical background of sparsity promoting regularisation, and recall the formulation of the *Iterative Soft-Thresholding Algorithm*. We then detail the key idea of our approach, that is, the convolutional interpretation of ISTA

using the wavelet transform. Finally, we describe the general design of the proposed CNN, ΨDONet, and the convergence results thereof. The discussions contained in that chapter involve *inter alia* microlocal analysis and wavelet theory, whose general concepts are briefly summarised in Appendix A and Appendix B respectively. Appendix C provides some further proofs of the results presented in chapter 3.

Chapter 4 explores the implementational aspects of ΨDONet when applied to the special case of LA-CT. We first provide a numerical verification of the theoretical results posited in chapter 3 and then offer a detailed description of two different implementations of ΨDONet, thoroughly investigated: Filter-based ΨDONet (ΨDONet-F) and Operator-based ΨDONet (ΨDONet-O). Numerical experiments are then conducted on simulated data: the testing setups as well as the obtained results are presented, and a final discussion is provided, where the results achieved with ΨDONet are compared with FBP and ISTA reconstructions.

The last chapter summarises the conclusions and gives suggestions for further work and perspectives.

The research presented in this doctoral thesis appears in the following publication:

> T.A. Bubba, M. Galinier, M. Lassas, M. Prato, L. Ratti, S. Siltanen, 2020. Deep neural networks for inverse problems with pseudodifferential operators: an application to limited-angle tomography, arXiv:2006.01620. To appear in *SIAM Journal on Imaging Sciences (SIIMS)*.

All the codes of the proposed algorithm are available at the link: `https://github.com/megalinier/PsiDONet/`.

2 Machine and deep learning theory

The purpose of this self-contained chapter is to introduce the fundamental principles of machine learning and to provide an overview of the theory and applications of deep learning.

Essentially, building a machine learning algorithm amounts to specifying a model that represents certain beliefs and conceiving a cost function that quantifies the correctness of those beliefs with respect to reality. Training such an algorithm then means to minimise this cost function by exploiting the information contained in the provided training set. This basic procedure and its underlying concepts, inherent to all machine learning approaches, are presented in section 2.1.

Being a specific kind of machine learning, deep learning inherits the same main components, namely a model, a cost function, an optimisation algorithm and a dataset. Its distinctive feature lies in the particular structure of its model, which is built as a network of artificial neurons. Section 2.2 presents the functioning of ANNs as well as their evolution through history. We also provide a detailed description of the so-called *back-propagation*, a fundamental tool of the learning process, and give some examples of gradient-based learning algorithms. We finally introduce some methods for the regularisation of DL techniques, and practical issues that can be encountered when dealing with such algorithms.

In section 2.3, a particular attention is given to CNNs, which are part of the ANN family, since a clear understanding of their functioning is needed to apprehend the choices of the design of ΨDONet and the challenges that had to be faced.

2.1 Machine learning basics

2.1.1 Learning algorithms

Inasmuch as they can learn and improve with experience, machine learning algorithms are particularly exciting for two main reasons. The first one is that they offer the possibility to tackle problems that are too complex to be solved by fixed programs designed and implemented by human beings. The second one is that developing a

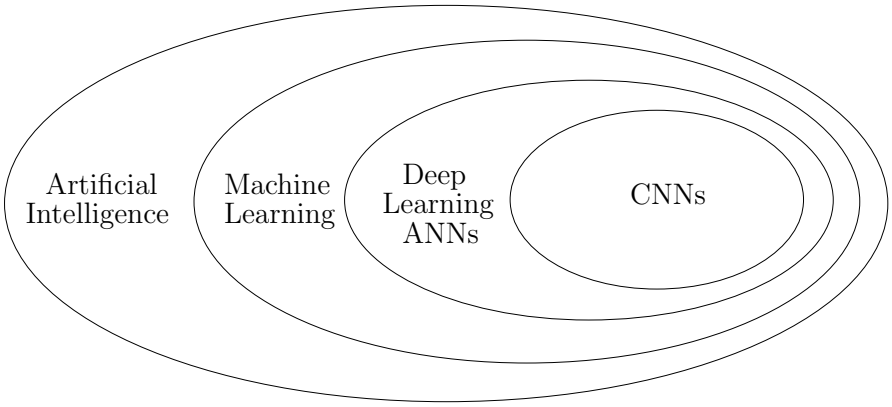


Figure 2.1: A Venn diagram showing the relation between convolutional neural networks, deep learning artificial neural networks, machine learning and artificial intelligence.

better understanding of machine learning involves developing a better understanding of the principles subtending intelligence.

A formal definition of such algorithms was proposed by Tom M. Mitchell in [150], where he stated:

> "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."

As such, *learning* can be interpreted as the process leading to the capability of performing a task (T). A very broad range of tasks can be addressed with machine learning techniques, including *e.g. classification*, where the computer program is asked to indicate which of k categories some input belongs to; *regression* which aims at predicting a numerical value given some input; *machine translation* where the input consists of a sequence of symbols in some language that the computer program is intended to convert into a sequence of symbols in another language; and *anomaly detection* where the program sifts through a set of objects or events and reports some of them as being atypical or abnormal. Direct real-life applications of the above-mentioned tasks respectively encompass object recognition such as the automatic tagging of people and objects in photo collections, price prediction in the context of insurance or trading, automatic text translation and credit card fraud detection. Those are only few examples of the numerous problems that have contributed to the success of machine learning in recent years.

The ability of a machine learning algorithm to perform the task of interest can be evaluated by means of a performance measure (P). Generally speaking, this performance measure, which is usually task-specific, is intended to assess how relevant are the outputs generated by the algorithm given a set of inputs. The purpose of machine learning algorithms being to be deployed in the real world and thus to perform as well as possible on data that has not been seen before, their efficiency is measured on a *test set* of examples, which is separate from the data used during the training process. A

common way to quantify the performance of a model is to measure its *accuracy*, that is, the proportion of examples for which the model produces the correct output over the total number of examples [147]. This performance measure proves to be indeed well suited for tasks such as classification, but finds its limits when it comes to more complex problems such as density estimation or regression. In these cases, it is more judicious to use a performance metric that, instead of assigning a binary value (1: correct output, 0: otherwise) to each example, give them a continuous-valued score. For image reconstruction tasks, the Mean Squared Error (MSE) and the Peak Signal-to-Noise Ratio (PSNR) are useful performance measures, but related works have also proposed alternative metrics such as the Structural SImilarity Measure (SSIM) [212] or the Haar wavelet-based Perceptual Similarity Index (HaarPSI) [170], which aim at evaluating the perceptual difference between two similar images. Those performance measures have been applied to the test set of our numerical experiments in order to gauge the effectiveness of our proposed algorithm and compare the different variants of its implementation (cf chapter 4).

As for the experience (E), it relies upon a dataset, that is, a set of examples of objects or events related to the task to be performed. If we consider a dataset of S elements, $\{x_1, \cdots, x_S\}$, each example, represented as a vector $x_i \in \mathbb{R}^p$, is a collection of *p features* that have been quantitatively measured from the object or event of interest. When the examples of the dataset are images, as it is the case in image deblurring, image denoising, or image reconstruction, the features to be processed are the values of the pixels.

Depending of the kind of dataset they are allowed to experience during their training, learning algorithms can be roughly divided into 3 categories: *unsupervised learning*, *supervised learning*, and *reinforcement learning*.

- *Unsupervised learning* algorithms are aimed at learning to make sense of the data without any guidelines nor human supervision. The examples they are trained on are said to be unclassified, or *unlabelled*, as no information about the expected output of the network is available. Such algorithms are able to classify, organise, or correctly process data by identifying previously unknown patterns that appear in the examples of the training set. As such, they prove useful for learning properties or relevant characteristics of the dataset structure: by observing several examples of a random variable, unsupervised learning algorithms can get information on the corresponding probability distribution that generated the dataset. A classic unsupervised learning task is to find the simplest and most condensed way to represent the data while preserving as much information as possible about the input. The well-known Principal Component Analysis (PCA) algorithm, which reduces the dimensionality of a multivariate data to few principal components, provides a good illustration of unsupervised learning algorithm [77]. Unsupervised learning can also be used to find a manifold to which the data lies near, cluster the data into groups of related examples, or draw new samples from the distribution (leading *inter alia* to fascinating applications in the field of art and photography, such as [73, 219, 118]).

- *Supervised learning* algorithms experience a dataset in which each example has not only features x_i, but also a corresponding *label* or *target* y_i. This target, supplied by an instructor or a teacher, shows the machine learning system what output it is expected to generate from a given input. Labels can be of different types, depending on the kind of task that is to be carried out: in the case of classification for example, the target is the scalar symbolising the correct class, while in image reconstruction, it usually consists of the complete, clean signal. Those input-output pairs are analysed by the learning algorithm and used to infer the underlying function that maps the features to the proper label. Thus, it is hoped that the training of the given examples will enable the algorithm to produce the correct outputs also for unobserved inputs. This may not be as simple as it appears, as it entails the learning algorithm to be able to generalise from the training data to unseen events (cf section 2.1.3). Supervised learning is often involved in tasks such as classification and regression, and is now widely employed for applications including medical diagnoses, handwriting or speech recognition, weather forecasting, spam detection and many other tasks. In addition to most of the ANNs, supervised learning algorithms encompass Support Vector Machines (SVM) [22, 46], decision trees [174], or the k-nearest neighbours [4]. It is to be noted that the boundary between supervised and unsupervised learnings may be not well defined and in-between learning paradigms are possible, such as the so-called *semi-supervised learning*, where some, but not all, examples include a supervision target.

- *Reinforcement learning* does not experience a fixed dataset, unlike the above-mentioned paradigms. Instead, the algorithm continuously interacts with its environment and thus benefits from a feedback loop between the learning system and its experiences. In practice, a reinforcement learning algorithm receives a numerical reward that encodes the success of its actions' outcome, and it seeks to learn the behaviour that maximises this reward, based on its past experiences (*exploitation*) as well as new choices (*exploration*). It thus can learn what is the best action to select based on its current state. Some common reinforcement learning methods are for example Q-learning [214, 215] or Temporal Difference (TD) [192]. Reinforcement learning is studied in a very broad range of disciplines, from robot navigation to information theory and from game theory to statistics and simulation-based optimisation. Amongst the applications that made reinforcement learning particularly popular, we may mention the field of self-driving cars as well as the area of board games played by computer (Chess, Go). More information can be found in the reference books [17, 193]

In the remainder of this thesis, we will consider a purely supervised learning framework.

2.1.2 From theory to a practical framework

One of the major problems of supervised machine learning can be formulated in the following terms: how to estimate the value of an unknown function at a new point given the values of this function at a set of sample points? From a statistical point of view, the learning problem can be seen as getting the best possible understanding of the unknown probability distributions from which the samples have been drawn, in such a way to be able to determine from which distribution some new sample comes from. The framework of statistical learning theory [205, 206], which aims at studying the concept of inference in machine learning, provides useful tools to apprehend and solve those problems.

2.1.2.1 Data generating distribution and hypothesis space

Formally, let $X \subseteq \mathbb{R}^p$ represent the vector space of all possible inputs, drawn independently from a fixed but unknown probability distribution $p_X(x)$. Furthermore, let $Y \subseteq \mathbb{R}^q$ 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 $p_Y(y|x)$, also fixed but unknown. The probabilistic relationship between X and Y can then be formalised in the form of a joint probability distribution $p_{XY}(x, y)$, defined over the set $X \times Y$, such that $p_{XY}(x, y) = p_X(x)p_Y(y|x)$. Such a distribution is called the *data generating distribution*. Implicitly, the main goal of supervised learning is to collect enough knowledge about $p_{XY}(x, y)$ in order to infer the function mapping the inputs to the correct outputs. In other words, the inference problem consists in providing a function or *estimator* f_θ such that for every sample (x, y) drawn from $p_{XY}(x, y)$, the output value y can be accurately predicted by the value $f_\theta(x)$.

Such an estimator f_θ is sought by the algorithm through a family $\mathcal{H}$ of prediction functions, also called the *hypothesis space*,

$$\mathcal{H} := \{f_\theta : X \to Y | \theta \in \Theta\}, \tag{2.1}$$

parametrised by θ in the space of all possible parameter values Θ. As further discussed in section 2.1.3, the choice of $\mathcal{H}$ is of particular importance as it determines the shape and characteristics of the candidate estimators, and thus influences the ability of the model to correctly infer the outputs.

2.1.2.2 Loss function

In order to find an accurate estimator f_θ through all the possible functions in $\mathcal{H}$, one needs to determine a criterion on which to base the quality assessment of an estimator. Usually, the sought estimator is intended to minimise the errors caused by incorrect predictions. As such, the quality criterion is most often chosen to be a *loss function* that quantifies the inaccuracy of the prediction with respect to the true output. Formally, a *loss function* is a function $\mathcal{L}(x, y, f_\theta(x)) : X \times Y \times Y \to [0, +\infty)$ such that:

$$\mathcal{L}(x, y, y) = 0, \quad \forall x \in X, \forall y \in Y. \tag{2.2}$$

In so far as it must dependably condense all aspects of the model down into a single number such that improvements on that scalar are a sign of a better model, the loss function has a significant impact on the solution f_{θ^*} found by the learning algorithm and is thus to be carefully chosen. Several factors may influence this choice, such as the task under consideration or the kind of model that is used. Moreover, a loss function may be required to have some specific properties. For instance, it may be necessary to use a loss function with a gradient large and predictable enough to be used in the training process of gradient-based learning algorithms, introduced in 2.2.3.

The possibilities for the choice of a loss function are multiple, not to say endless as it can be tailored to the problem of interest. We list the most commonly used ones hereunder, starting with the classic loss functions for **binary classification**, that is, when $Y = \{-1, 1\}$. Typically, such loss functions exploit the product $yf_\theta(x)$ of the true and the predicted output values.

- the 0-1 indicator function:

$$\mathcal{L}(x, y, f_\theta(x)) = H(-yf_\theta(x)), \tag{2.3}$$

  where $H : \mathbb{R} \to \{0, 1\}$ is the *Heaviside step function*, defined as:

$$H(t) = \begin{cases} 0 & \text{if } t < 0 \\ 1 & \text{if } t \geq 0. \end{cases} \tag{2.4}$$

  When the 0-1 loss is used, it is usually assumed that the set of predicted outputs is the same as Y, that is, $f_\theta(x) \in \{-1, 1\}$. Then, this loss function takes the value 0 if the predicted output $f_\theta(x)$ is the same as the true output y, and 1 otherwise. Although it seems to be a natural choice, the 0-1 loss proves quite intractable inasmuch as it is discontinuous and non-convex, and therefore prevents (sub)gradient methods from being applied. Most often, machine learning algorithms are based on continuous, convex loss functions, such as the functions mentioned below.

- the Hinge loss, or soft-margin loss:

$$\mathcal{L}(x, y, f_\theta(x)) = \max(0, 1 - yf_\theta(x)) =: |1 - yf_\theta(x)|_+. \tag{2.5}$$

  Unlike the 0-1 loss, the Hinge function is designed for situations in which the predicted outputs $f_\theta(x)$ are not restricted to the set $\{-1, 1\}$ and instead are allowed to assume real values. This setup is useful when, given an input x, one needs not only to predict which class the corresponding output y belongs to, but also to provide a certain confidence interval for this estimate. In that case, the label of the class is given by the sign of the function $f_\theta(x)$, while the absolute value of $f_\theta(x)$ offers a measure of prediction confidence. The Hinge function penalises the predictions such that $|yf_\theta(x)| < 1$, that is, it penalises the cases where the prediction has a sign different from the true output as well as the cases where they have the same signs but the confidence interval is too small. The Hinge

function is an example of smooth, convex loss function that is frequently used for maximum-margin classification and most notably by the well-known Support Vector Machines for Classification (SVMC) approach [48].

In the case of **multi-class classification** problems, the true labels are generally encoded as one-hot vectors, that is: each label y is represented as a vector whose length is equal to the number k of classes and all its entries are zeros, except for the entry corresponding to the class it belongs to, which is set to one. As for the outputs of the learning algorithm, they constitutes probabilities to belong to the different classes. The loss function most commonly employed in this situation is:

- the cross-entropy loss, or log-loss:

$$\mathcal{L}(x, y, f_\theta(x)) = -\sum_{i=1}^{k} y_i \log\left(f_\theta(x)_i\right). \tag{2.6}$$

The true output y being a one-hot vector, the only value resulting from this loss function for a given sample (x, y) is actually the negative logarithm of the predicted probability to belong to the true class. In the ideal case where this probability is predicted by the model to be 1, the loss function returns 0. On the contrary, the smaller this probability, the larger the penalisation. Fig. 2.2 shows the evolution of the cross-entropy loss as a function of the predicted probability to belong to the true class.

When it comes to **regression** problems, the output space is extended to real values: $Y \subseteq \mathbb{R}$ and as such, the quality of an estimate is not based anymore on the product $yf_\theta(x)$ but rather on the difference $y - f_\theta(x)$. Some of the most popular loss functions for regression include:

- the MSE, or ℓ_2-norm:

$$\mathcal{L}(x, y, f_\theta(x)) = (y - f_\theta(x))^2. \tag{2.7}$$

This widely used loss function, which minimises the sum of squared residuals, is usually involved when the outputs y are assumed to be corrupted by additive Gaussian noise.

- the ϵ-insensitive loss function:

$$\mathcal{L}(x, y, f_\theta(x)) = \max(0, |y - f_\theta(x)| - \epsilon) =: |y - f_\theta(x)|_\epsilon. \tag{2.8}$$

The ϵ-insensitive loss, which can be seen as an extension of the soft-margin loss defined in eq. (2.5), penalises the predictions when the distance between the true outputs and the predicted ones is greater than some scalar $\epsilon > 0$. Such choice is for example adopted by the Support Vector Machines for Regression (SVMR) methodology [48].

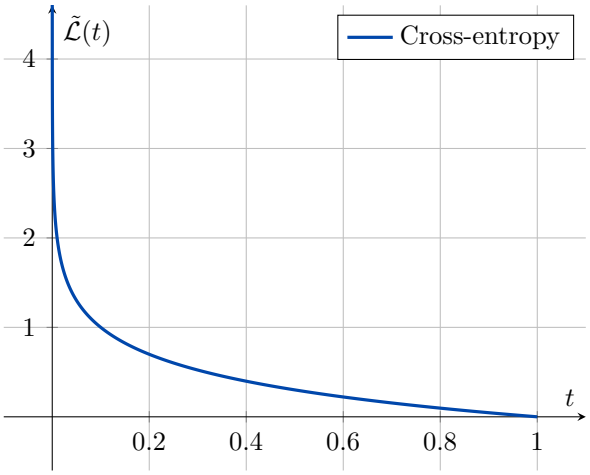


Figure 2.2: Cross-entropy loss, for multi-class classification. The label y is considered to be a one-hot vector, that is: it has only zero-valued entries, except for the entry that represents the class the sample belongs to, which is set to 1. We represent $\tilde{\mathcal{L}}(t) = \mathcal{L}(x, y, f_\theta(x)) = -\sum_{i=1}^{k} y_i \log\left(f_\theta(x)_i\right)$, where k is the number of different classes, and t stands for the probability predicted by the learning system for the sample to belong to the right class. Mathematically, $t = f_\theta(x)_j$, where j is such that $y_j = 1$. The other components of $f_\theta(x)$ actually do not affect the cross-entropy loss, as they are multiplied by the zero entries of y.

- the absolute value loss, or ℓ_1-norm:

$$\mathcal{L}(x, y, f_\theta(x)) = |y - f_\theta(x)|. \tag{2.9}$$

  This loss can actually be seen as an ϵ-insensitive loss for the particular case in which $\epsilon = 0$.

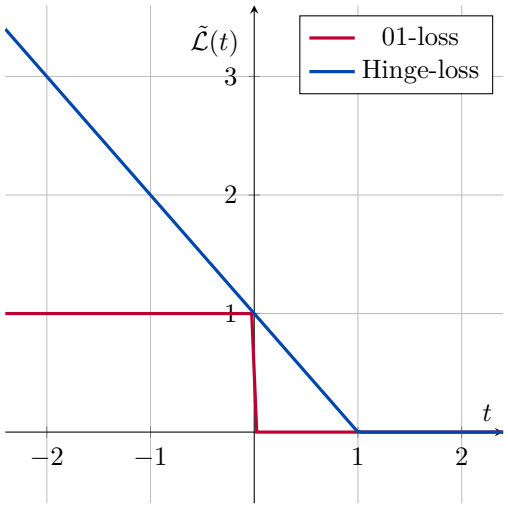
Fig. 2.3 illustrates the aforementioned loss functions for binary classification and regression. As for the loss functions involved in image processing problems, the ℓ_2-norm constitutes an interesting and prevalent choice. Further options are detailed in section 4.2.1.5.

2.1.2.3 Expected and empirical risks

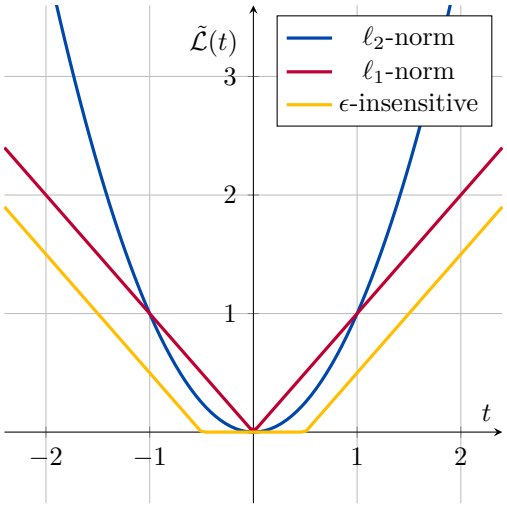
The loss function, once chosen, provides a clear way to penalise each triple $(x, y, f_\theta(x))$. The following question is then: how to combine these penalties in order to evaluate a particular estimate f_θ? The answer is given by the *expected risk*, or *generalisation error*, which measures the global predictive capability of the estimator f_θ. In other words, it is the expectation of the loss function with respect to the underlying probability distribution $p_{XY}(x, y)$:

$$\mathcal{J}(\theta) := \mathbb{E}[\mathcal{L}(x, y, f_\theta(x))] = \int_{X \times Y} \mathcal{L}(x, y, f_\theta(x)) p_{XY}(x, y) \mathrm{d}x\mathrm{d}y. \tag{2.10}$$

The learning process aims at minimising such a function $\mathcal{J}(\theta)$, also called the *objective function* or *cost function*, over the set of candidate estimators $\mathcal{H}$.



(a) Loss functions for binary classification. The variable t is such that $t = yf_\theta(x)$.



(b) Loss functions for regression. The variable t is such that $t = f_\theta(x) - y$.

Figure 2.3: Graphics of different loss functions. We represent $\tilde{\mathcal{L}}(t) = \mathcal{L}(x, y, f_\theta(x))$, where t depends on y and $f_\theta(x)$, according to the task under consideration.

In the ideal case, the data generating distribution $p_{XY}(x, y)$ is known and the minimisation of $\mathcal{J}(\theta)$ can be significantly simplified. Even though such model may still incur some error, the so-called *Bayes error* [71], on problems where there is some noise in the distribution, finding an accurate estimator f_θ is likely to become a straightforward problem to solve.

In most situations, however, the data generating distribution is not known, and minimising such a cost function is thus impracticable. Instead, the available information consists of a dataset of S training examples, that is, a set of random independent identically distributed (i.i.d) samples drawn from $p_{XY}(x, y)$. Let us call such a *training set*:

$$\mathcal{S} = \{(x_i, y_i) \mid \quad x_i \in X, \quad y_i \in Y, \quad i = 1, \ldots, S\}, \tag{2.11}$$

where every x_i is an input vector with p features, and y_i is the corresponding label. As the training set provides a certain number of examples of the probabilistic relationship between X and Y, the unknown distribution $p_{XY}(x, y)$ can be replaced with an *empirical probability distribution*. This alternative leads to the *empirical risk minimisation* (ERM), formulated in [204] where Vapnik suggests to minimise an approximation of the expected risk $\mathcal{J}(\theta)$, constructed by averaging the loss function on the available examples of the training set $\mathcal{S}$. The resulting surrogate $\mathcal{J}_S : \Theta \to \mathbb{R}$, called *training error*, or *empirical risk*, reads:

$$\mathcal{J}_S(\theta) = \frac{1}{S} \sum_{i=1}^{S} \mathcal{L}(x_i, y_i, f_\theta(x_i)). \tag{2.12}$$

It can be shown, using the law of large numbers, that the empirical risk converges in probability to the expected risk for a sufficiently high number of examples [206]. This entails that for a fairly large training set, the minimisation of the empirical risk $\mathcal{J}_S$

can lead to a good estimate of the minimum expected risk $\mathcal{J}$. Therefore, the learning process comes down to find the best parameter θ in such a way that the corresponding function $f_\theta \in \mathcal{H}$ minimises the empirical risk $\mathcal{J}_S$.

2.1.3 Generalisation and capacity

2.1.3.1 Generalisation, overfitting and underfitting

The central challenge in machine learning consists in finding an estimator f_θ that is able to predict outputs from inputs that do not belong to the training set. This ability to accurately perform on new, previously unseen inputs, is called *generalisation*.

The generalisation ability of a model is usually assessed by measuring its performance on a set of new examples, different from those used in the training process. To this aim, the dataset initially available is partitioned into two distinct subsets: the training set, used for the learning process, and the *test set*, exclusively employed to evaluate the performance of the model. This is known as the *hold-out method* [123]. In this manner, the examples forming the test set are independent from the examples on which the model is trained but they follow the same probability distribution.

When training the algorithm, one seeks to minimise the empirical risk on the training set, *i.e.* one seeks to reduce the so-called *training error*. However, unlike classic optimisation problems, machine learning requires more than a low training error: it also needs its *generalisation error*, that is, the error computed on the test set, to be small as well. The factors determining how well a trained model will perform come down to:

- its ability to make the training error small

- its aptitude to make the gap between test and training error small

These two factors are closely related to two of the main issues in machine learning: *underfitting* and *overfitting*. Underfitting occurs when the model does not achieve a sufficiently low error value on the training set. This means that the model is not able to fit the data. Overfitting happens when the gap between the test error and the training error is too large. This, one the other hand, entails that the model has memorised some properties of the training set but will not be able to generalise to other data. The main challenge of machine learning is therefore to select and train a model while avoiding those two phenomena.

2.1.3.2 Model capacity

The propensity of a model to fall into overfitting or underfitting can be controled 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, while models with a too high capacity may fit too closely to the training set and therefore fail to perform well on unseen data. Hence the importance of opting for a

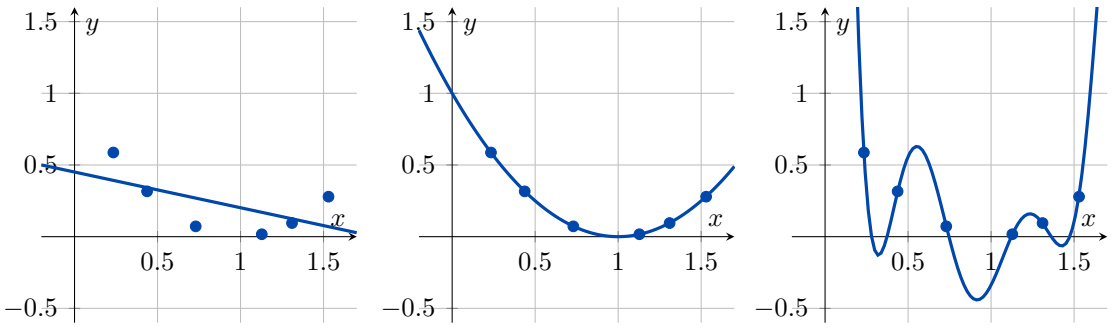


Figure 2.4: Consider a set of training samples generated by a quadratic function. On the left: the linear predictor is not able to capture the data curvature (underfitting). At the center: a quadratic predictor matches the underlying function (appropriate capacity). On the right: a degree-8 predictor pass exactly through all the training points, but does not extract the correct structure (overfitting).

capacity adequate to the complexity of the task and the amount of training data on hand.

An illustrative example of the link between the concepts of underfitting and overfitting and the capacity of the model is shown in fig. 2.4. We compare a linear, a quadratic and a degree-8 predictors attempting to fit a set of samples generated by a quadratic function. The linear predictor is not able to seize the curvature in the true underlying problem and results in an underfitted model. On the contrary, the degree-8 polynomial, which is defined by more parameters than there are training examples, is capable of representing infinitely many functions that do pass through the training points. Thus, the probability of selecting a solution that generalises well is low, given the endless number of different solutions that exist: the model is said to overfit. In this example, the appropriately chosen capacity of the model is given by the quadratic predictor, which perfectly matches the underlying function and is thus able to generalise well to data.

Generally speaking, when increasing the capacity, the training error decreases until it asymptotes to the minimum possible error value. As for the test error, it assumes a U-shaped curve, as illustrated in fig. 2.5. As it can be observed, the gap between the two errors increases and eventually outweighs the decrease of the training error. In short, simpler models are more likely to generalise while more complex models are more likely to achieve low training error.

It has been shown, thanks to major advances in the field of statistical learning theory, that the gap between training error and generalisation error is bounded from above by a quantity that increases as the model capacity increases, but narrows as the number of training examples grows [205]. These bounds offer a theoretical justification that machine learning algorithm can work. They are, however, rarely employed in the context of deep learning, which is the framework of the research presented in this thesis. The main reason is that determining the capacity of a deep learning algorithm is often an arduous task due to the little theoretical understanding of the non-convex optimisation problems involved in the training of DNNs, [77].

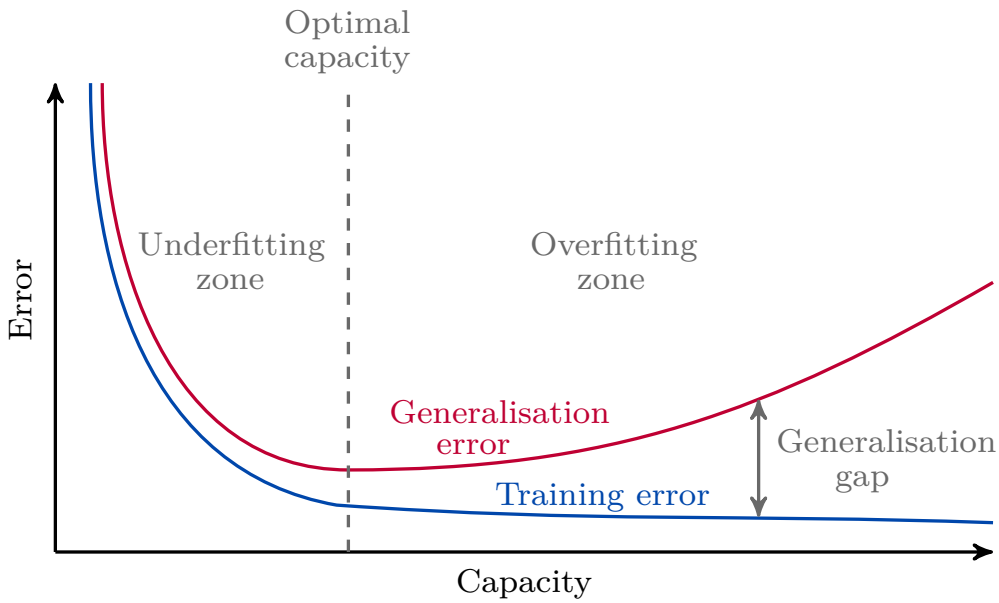


Figure 2.5: Illustration of the relationship between capacity and error. When the model has a too small capacity, both the training error and the generalisation error are high, which corresponds to the underfitting regime. As the capacity increases, the training error decreases but the gap between the training and generalisation errors increases. When this generalisation gap becomes too large, the model enters the overfitting regime. The optimal capacity offers a compromise between a low training error and a small generalisation gap. Illustration inspired from [77].

One way to adjust the model's capacity is to wisely select the hypothesis space $\mathcal{H}$ of the functions the learning algorithm can consider as candidate solutions. Inasmuch as it controls the shape and complexity of potential solutions, the choice of the hypothesis space is of critical importance. In our previous example, we saw that a linear predictor is unable to fit a problem where the true underlying function is quadratic, and with good reason: if $\mathcal{H}$ allows only linear models, it is clear that the learning algorithm cannot do otherwise than merely discover functional dependencies of the linear type. Extending $\mathcal{H}$ to polynomials with a degree $d \geq 1$ is, for example, a way to increase the model's capacity and thus avoid underfitting.

The breadth of the hypothesis space $\mathcal{H}$ actually defines what is called the *representational capacity* of the model, which can be thought of as the theoretical capacity of the model. In practice, finding the best function within the whole hypothesis space is usually a difficult optimisation problem and the learning algorithm confines itself to finding a solution that reduces the training error at best. Limitations, such as the imperfection of the optimisation algorithm, thus imply that the learning algorithm's *effective capacity* may be less than the representational capacity of the model family.

2.1.3.3 Bias-variance trade-off

The field of statistics offers diverse tools to understand generalisation and characterise the notions of underfitting and overfitting. We recall concepts such as bias and variance, which measure different sources of error in an estimator.

By definition, an estimator $\hat{\theta}_S$ is a function or variable that aims at providing a prediction of some quantity of interest θ. In general, the quantity of interest, which is assumed to be fixed but unknown, can be a single parameter or a vector of parameters in some parametric model, such as the weights of a neural network, or even a whole function. More formally, given a set $\{x_1, \cdots, x_S\}$ of S i.i.d. data points, an estimator or *statistic* $\hat{\theta}_S$ is any function of the data:

$$\hat{\theta}_S = g(x_1, \cdots, x_S). \tag{2.13}$$

According to this general definition, almost any function qualifies an estimator, since it is not required that g return a value that is close to the true θ, or even that the range of g be the same as the set of admissible values of *theta*. However, a good estimator is such that its output is close to the true underlying θ that generated the data.

As a function of data drawn from a random process, the estimate $\hat{\theta}_S$ can be considered as a random variable and as such, can be investigated from a statistical point of view. Below, we report some commonly studied properties of estimators and discuss what kind of useful information they provide.

A first characteristic of random variables and estimators is their *bias*. It measures the expected deviation of the estimator from the true value θ:

$$\text{bias}(\hat{\theta}_S) = \mathbb{E}(\hat{\theta}_S) - \theta, \tag{2.14}$$

where the expectation is over the data (seen as samples of a random variable) and θ is the true underlying value used to define the data generating distribution. Informally, the bias quantifies the error that comes from flawed assumptions in the learning algorithm. A high bias can cause the learning algorithm to miss the relevant relations between features and labels, and thus fall into underfitting. Thus, one of the qualities desired for an estimator is a low, not to say nil bias. If $\text{bias}(\hat{\theta}_S) = 0$, the estimator is said to be *unbiased*, which implies that $\mathbb{E}(\hat{\theta}_S) = \theta$. It is said to be *asympotically unbiased* if $\lim_{S\to\infty} \text{bias}(\hat{\theta}_S) = 0$, which entails that $\lim_{S\to\infty} \mathbb{E}(\hat{\theta}_S) = \theta$.

Another property of the estimator we may be interested in is how much we expect it to vary as a function of the data sample. This can be evaluated through the computation of the *variance*. The variance of an estimator is simply the variance

$$\text{Var}(\hat{\theta}_S) = \mathbb{E}[(\hat{\theta}_S - \mathbb{E}(\hat{\theta}_S))^2]. \tag{2.15}$$

Alternatively, one can consider the square root of the variance, called the *standard error*. The variance, or the standard error, of an estimator gives a measure of the deviation from the expected estimator value that any particular sampling of the data is likely to cause. In other words, the variance can be seen as the error related to the sensitivity of the model to small fluctuations in the training set. A high variance estimator may lead to overfitting and is thus to be avoided.

Just as we might like an estimator to have low bias, we would also like it to exhibit a fairly low variance. A way to strike a balance is the MSE of the estimates.

$$\begin{aligned} \text{MSE} &= \mathbb{E}[(\hat{\theta}_S - \theta)^2] && (2.16)\\ &= \mathbb{E}[\hat{\theta}_S^2 + \theta^2 - 2\hat{\theta}_S\theta] && (2.17)\\ &= \mathbb{E}[\hat{\theta}_S^2] + \theta^2 - 2\theta\mathbb{E}[\hat{\theta}_S] && (2.18)\\ &= \text{bias}(\hat{\theta}_S)^2 + \text{Var}(\hat{\theta}_S). && (2.19) \end{aligned}$$

The MSE measures the overall expected deviation between the true value of the parameter θ and the estimator $\hat{\theta}_S$. As it can be seen in eq. (2.19), the MSE incorporates both the bias and the variance. Thus, desirable estimators have low MSE, as it ensures they have both low bias and low variance.

The relationship between bias and variance is intimately linked to the machine learning concepts of capacity, overfitting and underfitting. At the beginning of the training process, the data has little influence on the learnt function and therefore, the model output is far from the desired function: the bias is large and the variance is low (underfitting). Then, as the model capacity increases, bias tends to decrease and variance tends to increase. If trained too long, the algorithm eventually learns the noise specific to the dataset, and the variance becomes large (overfitting). Fig. 2.6 illustrates these phenomena and shows again the U-shaped curve of generalisation as a function of capacity.

To summarise this subsection, the choice of the appropriate model capacity is a crucial issue in machine learning: if the model is too simple, it will not be able to capture the relevant information in the data; if the model is too complex, it will generalise poorly to unseen data. The happy medium between those two phenomena, respectively called underfitting and overfitting, is tightly linked to the so-called bias-variance trade-off. The strategies to control such a trade-off, known under the name of *regularisation*, are introduced in section 2.1.4.

2.1.4 Regularisation techniques

Regularising a learning algorithm consists in modifying some of its characteristics, such as its objective function, its capacity or even its training set, with the purpose of reducing its generalisation error but not its training error. In other words, regularisation offers a way to solve overfitting. We report some of the most commonly used regularisation techniques below.

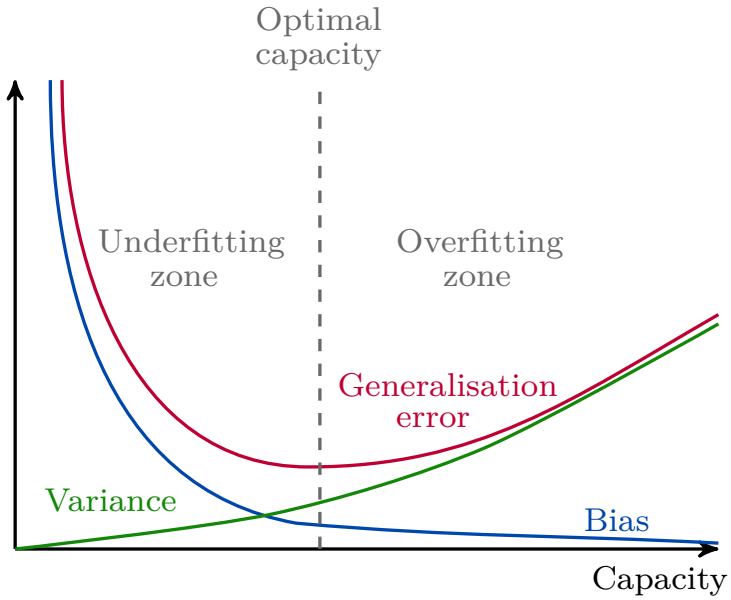


Figure 2.6: Representation of the relationship between bias, variance and generalisation error in function of the model capacity. As capacity increases, bias tends to decrease and variance tends to increase, resulting in U-shaped curve for generalisation error. Illustration inspired from [77].

2.1.4.1 Regularised objective function

We have already mentioned that the performance of the learning algorithm can be modified by adding or removing functions from the hypothesis space $\mathcal{H}$ of candidate solutions. It is true that the behaviour of the algorithm is strongly affected by how large is $\mathcal{H}$, but not only: it also greatly depends on the kind of functions the learning system is allowed to pick from $\mathcal{H}$. It is possible, for instance, to specify a preference for one solution over another in the hypothesis space. This means that both functions are eligible, but one may be preferred, and the unpreferred solution will be selected only if it performs a significantly better fitting of the training data than the preferred one.

More specifically, one can narrow the set within which the parameters are chosen by adding a norm penalty $\mathcal{R}(\theta)$, also called *regulariser*, to the objective function. This regularisation strategy, which aims at limiting the model capacity, is broadly used in diverse machine learning approaches, such as neural networks, linear regression or logistic regression. The regularised objective function is denoted $\tilde{\mathcal{J}}_S(\theta)$ and reads:

$$\tilde{\mathcal{J}}_S(\theta) = \mathcal{J}_S(\theta) + \lambda\mathcal{R}(\theta), \tag{2.20}$$

where $\lambda \in [0; +\infty)$ is a hyperparameter that weights the contribution of the norm penalty term $\mathcal{R}$, relative to the standard cost function $\mathcal{J}_S$. If λ is set to 0, the model is not regularised, that is, we impose no preference about the functions to pick from $\mathcal{H}$. Larger values of λ results in more regularisation. Different choices for the norm penalty can lead to different solutions being preferred.

A classic choice for the regulariser $\mathcal{R}(\theta)$ is the ℓ_2-norm of the parameters θ, which promotes weight decay: $\mathcal{R}(\theta) = \theta^T\theta$. This approach, known as the *Tikhonov regularisation* leads to the so-called *structural risk minimisation* problem. In that case, large values of λ force the weights to become smaller and minimising $\tilde{\mathcal{J}}_S(\theta)$ amounts to finding weights that make a trade-off between fitting the training data and being small. The preferred solutions are thus solutions that have a smaller slope, or that put weight on fewer of the features. In short, even though the model is able to represent functions with much more complex shape, weight decay encourages the use of simpler functions and thus constrains the predictor to be more stable with respect to data fluctuations.

2.1.4.2 Early stopping

We mentioned in section 2.1.3 that during the training process, the training error tends to decrease while the generalisation error presents a U-shaped curve, whose minimum defines the boundary between the underfitting and overfitting regimes. It thus appears judicious to stop the learning process at the very moment when the generalisation error is the smallest possible, before it starts to increase again. How to carry out such a technique, known as *early stopping*?

Earlier we discussed how a hold-out test set, which consists of examples coming from the same distribution as the training set, can be employed to estimate the generalisation error of a learner. However, it is crucial that these test examples are not involved in any way in the choices about the model, including the moment when to stop the training process. For this reason, the generalisation error is always assessed during the learning process on a different set, the *validation set*, which is built from the training data. Specifically, the training set is split into two disjoint subsets: one of these subsets is used to learn the parameters, the other one is used to estimate the generalisation error. Usually, the subset of data employed to learn the parameters is still called the training set, although this may be confused with the bigger collection of examples used for the entire training process. The second subset of data is the validation set. It not only enables to apply early stopping methods, but also allows for a sound choice of the algorithm hyperparameters, which will be introduced in section 2.1.6. Generally, 80% of the training data are assigned to the training set, and 20% are assigned to the validation set [77]. Once the hyperparameters have been definitely selected and the resulting model has been trained according to the early stopping method, the generalisation error can finally be estimated on the test set, so far not used in any way by the learner.

In practice, applying early stopping methods entails to periodically evaluate the error on the validation set during the minimisation process. Every time this error decreases, a copy of the model parameters θ is saved, in order to be able to restore the best parameter setup at the end of the training. The learning process stops when no set of parameters significantly improves the error on the validation set for some number of iterations.

Inasmuch as they allow for the training process to be stopped even if a minimum

of the empirical risk has not been reached yet, early stopping methods may lead to a reduction of the computational cost and training time while effectively regularising the model. The only cost of early stopping is caused by the periodic evaluation of the error on the validation set. Nevertheless, this cost is generally negligible compared to the total training time. Besides, it can be further reduced by using a smaller validation set, or by decreasing the frequency of these observations, although this may lead to a worst estimate of the optimal parameters.

2.1.4.3 Data augmentation

The best way to help a machine learning model generalise better is to train it on more data. In practice, however, the amount of available data is limited. One way to circumvent this problem is to generate fake data and incorporate it to the training set.

A specific problem for which data augmentation has been particularly effective is object recognition. The high dimensionality of images, as well as the fact they include a wide range of factors of variation, many of which can be easily simulated, make them particularly suitable for data augmentation. Operations such as translating the training image a few pixels in each direction, rotating or scaling the image may often greatly improve generalisation. Injecting a small random noise in the input can also be seen as a form of data augmentation [187]. In the overall, many data augmentation strategies can be thought of, but it is important that the chosen transformations do not change the correct output. For example, optical character recognition tasks require recognising the difference between '6' and '9', or between 'b' and 'd', so vertical or horizontal flips are not suitable ways of augmenting datasets in those cases.

Although data augmentation is not applicable to all machine learning tasks, it has proven quite effective for several problems beyond object classification, such as speech recognition [113] for instance.

2.1.4.4 Further techniques

There exist many other ways to make a machine learning model generalise better. *Bagging* (short for *bootstrap aggregating*), for example, is a technique for reducing generalisation error by combining several models [28]. It consists in training several different models independently and then use all the trained models to predict the outputs of the test set.

Another example is the *dropout* technique, popular in the field of neural networks [190]. It entails randomly "dropping out", or omitting, units of neural network during its training. It actually can be thought of as a special kind of bagging.

A detailed review of existing regularisation techniques for deep learning algorithms can be found in [129] and the references therein.

2.1.5 Gradient-based optimisation

2.1.5.1 Gradient-based general strategy

Minimising the empirical risk (2.12) or its regularised form (2.20) is not a trivial task. Most of the time, calculating a closed form solution is impracticable and the optimisation problem needs to be solved through an iterative process. Prominent approaches, said to be *gradient-based*, aim at exploiting information about the objective function's gradient in order to generate a sequence of iterates $\{\theta^{(k)}\}_{k\in\mathbb{N}} \subset \Theta$ so that $\mathcal{J}_S(\theta^{(k)})$ decreases along the iterations.

The simplest gradient-based minimisation method, known as *gradient descent* (GD), offers to iteratively adjust the internal parameters $\theta^{(k)}$ by making small steps in the direction opposite to the first-order gradient of $\mathcal{J}_S$. Moving along that direction means following the so-called *steepest descent*, that is, the direction in which the function is locally decreasing the most from the current point. This strategy, attributed to Cauchy [33], can be formally written in the general form of algorithm 1.

Algorithm 1 Gradient descent update

Input: $\mathcal{S} = \{(x_i, y_i)| \quad x_i \in X, \quad y_i \in Y, \quad i = 1, \dots, S\}$, training set
$\theta^{(0)}$, initial set of parameters
$\alpha^{(0)}$, initial steplength

for $k = 0, 1, \dots$ **do**
Compute gradient estimate $g_k(\theta^{(k)})$
Define steplength $\alpha^{(k)}$
Update parameters: $\theta^{(k+1)} = \theta^{(k)} - \alpha^{(k)} g_k(\theta^{(k)})$
end for

In algorithm 1, $g_k(\theta^{(k)})$ stands for the gradient of the objective function $\mathcal{J}_S$ with respect to the internal parameters θ, evaluated at $\theta^{(k)}$. There exist three broad categories of gradient descent algorithms, which differ in the amount of data used to compute gradient g_k. Those variants are detailed in section 2.1.5.2.

As for the steplength $\alpha^{(k)}$, it represents the size of the move that is made at each iteration k in the parameter space Θ. It is also called *learning rate* in the special context of deep learning. A brief review of existing methods to determine such a scalar is given in section 2.1.5.3.

2.1.5.2 Stochastic, batch and minibatch gradient-descent methods

The essential difference between the gradient descent categories detailed in this section rests upon the number of training examples they exploit to compute the gradient g_k at each iteration k. Broadly speaking, the amount of data chosen for such a computation offers a trade-off between the accuracy of the parameter update and the computational cost needed to perform the update.

Batch gradient descent Standard GD algorithm computes the gradient g_k of the cost function over the entire training set. For this reason, it is usually referred to as *batch gradient descent*, since an entire *batch* of data is used to update the parameters $\theta^{(k)}$ at each iteration. Mathematically, the gradient g_k is an average over all the training examples:

$$g_k(\theta^{(k)}) = \frac{1}{S}\nabla_\theta \sum_{i=1}^{S} \mathcal{L}(x_i, y_i, f_{\theta^{(k)}}(x_i)). \tag{2.21}$$

Such an algorithm guarantees, under some conditions about $\alpha^{(k)}$, to converge to a local minimum of $\mathcal{J}_S$ [23]. When the cost function is convex, all local minima are also global minima and therefore, gradient descent can converge to the global solution. When the cost function is non-convex, as it is the case in deep learning applications, it is hoped that the found local minimum performs nearly as well as the global one. Specialised gradient descent methods designed to deal with this challenge are further detailed in section 2.2.4.

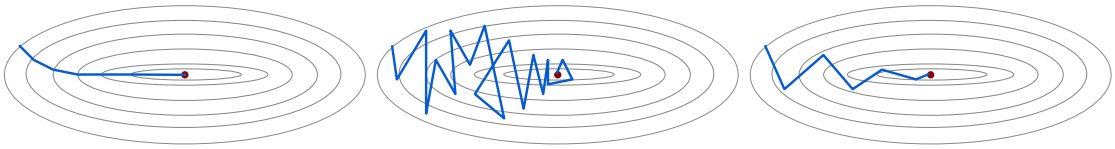
The main drawback of batch GD comes from the fact that computing the average of the gradients over the whole training set at each iteration is burdensome and requires significant computer resources. Although it can benefit from parallelisation due to the sum structure of the objective function, such an approach remains expensive, not to say impracticable when the training set and the number of input features are very large.

Stochastic gradient descent The *online* or *stochastic gradient descent* approach (SGD), initially proposed by Robbins and Monro [173], overcomes the aforementioned problem. Instead of averaging the gradient of the cost function over the complete training set, it computes the gradient on a single sample $(x_{i_k}, y_{i_k}) \in \mathcal{S}$, randomly chosen at each iteration, and updates the parameters $\theta^{(k)}$ accordingly. In that case, g_k can be regarded as an estimate of the actual gradient of $\mathcal{J}_S$ and reads:

$$g_k(\theta^{(k)}) = \nabla_\theta \mathcal{L}(x_{i_k}, y_{i_k}, f_{\theta^{(k)}}(x_{i_k})). \tag{2.22}$$

SGD is not deterministic as its behaviour relies on the random sequence $\{i_k\}_k$. In practice, the training set is usually shuffled in advance and a new example is picked from that mixed training set at each iteration. In this way, it is ensured that all training examples are seen while guaranteeing their randomisation. In case the training set is to be observed several times by the learning algorithm, it can be shuffled before each pass, to prevent cycles.

As each iteration of SGD involves the computation of the gradient on a single training example, this strategy does reduce the computational burden and allows to carry out iterations significantly faster than batch GD. This simplification of the gradient, however, introduces some random noise in the minimisation process, as illustrated in fig. 2.7. Concretely, each direction $-\nabla_\theta \mathcal{L}(x_{i_k}, y_{i_k}, f_{\theta^{(k)}}(x_{i_k}))$ is not necessarily a descent direction of $\mathcal{J}_S(\theta^{(k)})$. Therefore, parameter updates have a greater variance that can cause large fluctuations in the objective function and it is not guaranteed that a minimum (even local) will be reached in a reasonable amount of time. On the other hand,



(a) Batch GD. (b) Stochastic GD. (c) Minibatch GD.

Figure 2.7: Illustration of the behaviour of a gradient-descent algorithm according to the amount of data used to compute the gradient. The grey ellipses represent the level sets of a convex function. In the case of batch GD, all the training examples are employed at each iteration and the optimisation algorithm converges to the minimum. When using SGD or minibatch GD, the gradient is computed on much less examples of the training set: this approximation tends to cause fluctuations in the optimisation process.

this noise can also prove useful to find better solutions, in particular if the objective function has numerous local minima, as it is the case in deep learning. While batch GD converges to the minimum of the basin in which the initial parameters were placed, even if its value is high with respect to that of the global minimum, the noise generated by stochastic learning may enable the parameters to jump into another basin, with a possibly deeper local minimum [24, 99].

It has been shown that if the learning rate $\alpha^{(k)}$ is gradually decreased during the training and subject to relatively mild assumptions, SDG converges almost surely to a global or local minimum, in the convex and non-convex case respectively [23]. Even better, the comparison between the convergence rates of batch GD and SGD is in favour of the latter for *big data* regime [24].

Middle ground: minibatch gradient descent A compromise between the exact calculation of the gradient of $\mathcal{J}_S$ and its approximation made on a single example is given by the average of the gradients over a small subset of the training samples. This kind of algorithm, called *minibatch gradient descent*, combines the best properties of deterministic and stochastic methods and as such, has become the algorithm of choice in many machine and deep learning applications.

At each iteration k, a minibatch of S' examples:

$$\mathcal{S}_k = \{(x_i, y_i) \mid i \in \mathcal{I}_k \subset \{1, \dots, S\},\ \sharp\mathcal{I}_k = S' < S\} \subset \mathcal{S} \tag{2.23}$$

is drawn uniformly from the training set $\mathcal{S}$. These minibatch selections are performed without repetitions, until the training set has been entirely seen by the algorithm. The estimate of the gradient g_k reads:

$$g_k(\theta^{(k)}) = \frac{1}{S'} \nabla_\theta \sum_{i \in \mathcal{I}_k} \mathcal{L}(x_i, y_i, f_{\theta^{(k)}}(x_i)). \tag{2.24}$$

The size of the minibatch offers a way to balance the accuracy of the parameter update and the generalisation abilities of the learning system, as well as the computational resources and time it requires. A minibatch usually contains tens or few

hundreds examples, depending on the context. This choice generally relies on the considerations listed below.

Larger minibatches provide a more accurate estimate of the gradient, and reduce the variance of the parameter updates, leading to a possibly more stable convergence (see fig. 2.7). Smaller minibatches, on the contrary, add some noise to the learning process, which can result in a beneficial regularising effect [217]. The generalisation error thus shows better results for small minibatches and often proves best for a minibatch size of 1 [77], which actually corresponds to the SGD setup. Training such small minibatches, however, might require a very small learning rate to ensure the stability of the algorithm, due to the high variance in the estimate of the gradient. As a consequence, the training may demand a very high running time caused by the reduced learning rate and the greater number of steps needed to observe the entire training set.

From a practical point of view, minibatch GD has the advantage of using a relatively small number $S' < S$ of examples that remains fixed even when the training set grows, making possible the handling of very large datasets. Furthermore, this approach enables some degree of parallelisation to be exploited in the computation of the minibatch gradients. Nowadays, such computations can be made notably efficient thanks to the modern parallel computing architectures as well as the state-of-the-art machine and deep learning libraries. In particular, for a proper choice of the minibatch size, minibatch GD allows to make the most of multicore architectures, while those are underutilised in the case of extremely small minibatches and SGD.

2.1.5.3 Determining the steplength

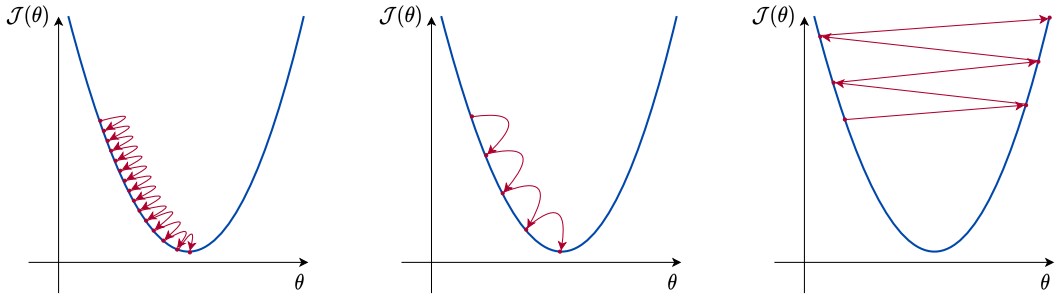
The choice of the steplength $\alpha^{(k)}$ is fundamental for the convergence of GD algorithms. If the $\alpha^{(k)}$ is too small, significant updates of parameters may be extremely slow and thus reaching an acceptable value for the cost function may take a very long time. On the other hand, if $\alpha^{(k)}$ is too large, the local information carried by the gradient at the current point may not be relevant anymore and the parameters may never reach acceptable loss at all. Fig. 2.8 illustrates the behaviour of the optimisation algorithm in function of the steplength.

Several strategies were proposed to judiciously choose $\alpha^{(k)}$ at each iteration k. The simplest one, known as *exact line search* selects the steplength such that:

$$\alpha^{(k)} = \min_{\alpha>0} \mathcal{J}_S(\theta^{(k)} - \alpha g_k(\theta^{(k)})). \tag{2.25}$$

Alternative approaches intend to determine $\alpha^{(k)}$ at a lower cost. For instance, inexact line search techniques make use of sets of inequalities to restrict the domain of the steplength search, such as Wolfe rules or Goldstein conditions [67]. More recently, the two-point step size algorithm proposed by Barzilai and Borwein [10] opened up new interesting perspectives about computationally cheap steplength search methods.

While those approaches have become widely spread in several machine learning applications, they are little popular in the special context of deep learning, partly due to the computational cost they add to the training process while a simpler update



(a) Steplength too small. (b) Optimal steplength. (c) Steplength too large.

Figure 2.8: Illustration of the influence of the steplength (also called *learning rate*) on the convergence of the optimisation algorithm. When it is too small, the convergence becomes very slow. On the contrary, when it is too large, it may lead to the divergence of the gradient-descent algorithm.

rule may already perform very well. Instead, a learning rate schedule is usually hand-designed before starting the training, according to few simple rules. When using a batch GD algorithm, the learning rate can actually be constant over the training, as the exact gradient becomes small and then zero when getting closer to the minimum. In the case of stochastic or minibatch GD, however, the approximation of the gradient results in fluctuations that do not vanish around a minimum. In fact, the variance of such fluctuations around a local minimum is proportional to the learning rate [23, 153]. This is why, it is usual practice to gradually decay the learning rate over the iterations of the training. Formally, it has been demonstrated that the following conditions about the learning rate $\alpha^{(k)}$:

$$\sum_{k=1}^{\infty} \alpha^{(k)} = \infty, \tag{2.26}$$

and

$$\sum_{k=1}^{\infty} (\alpha^{(k)})^2 < \infty, \tag{2.27}$$

are sufficient to guarantee the convergence of SGD [173]. Intuitively, the first condition (2.26) ensures that the parameters $\theta^{(k)}$ will reach the basin of attraction of a minimum, while the second constraint (2.27) guarantees that the learning rate decays sufficiently for the parameters to converge to the minimum instead of just oscillating around it [216]. In other words, the initial noisy optimisation may be beneficial inasmuch as it enables to explore a larger fraction of the parameter space, while keeping a low probability to be trapped in local minima. Once a promising region of the parameter space has been found, reducing the learning rate, and thereby the gradient fluctuations, allows to fine-tune the parameters.

By way of example, the following update rule is part of the step decay methods commonly used in deep learning:

$$\alpha^{(k)} = \alpha^{(0)} \cdot q^{\left\lfloor \frac{k}{t} \right\rfloor}, \tag{2.28}$$

where $q < 1$ is the factor by which the initial learning rate $\alpha^{(0)}$ is reduced every t iteration. Another approach, fully heuristic, consists in monitoring the validation error while training with a fixed learning rate and decrease the latter by a constant whenever the validation error stops improving.

2.1.6 Hyperparameter selection

Most machine learning systems are characterised by a set of special parameters, which are called *hyperparameters*. They are distinguishable from the parameters θ of the model in that they are not learnt by the system and instead, need to be tuned before starting the training process. They are usually used to control the algorithm behaviour, as they partly define the minimisation process or the architecture of the algorithm itself. Thus, they may significantly affect the performance of the resulting model as well as the time required by the training process [42].

The regularisation parameter λ in eq. (2.20), used to control the trade-off between the data-fidelity cost function and the regulariser, provides a first example of hyperparameter. Other examples include *inter alia* the (initial) steplength of the iterative learning algorithm, the minibatch size, the amount of times (called *epochs*) the training set is seen by the learning system or else the parameters describing the loss function, such as the threshold ϵ in the case of the ϵ-insensitive loss function. Hyperparameters specific to the field of deep learning also encompass the number and size of the layers in the neural network.

There are mainly two reasons to consider a setting as a hyperparameter that the learning algorithm does not learn. The first one is that this setting may be difficult to optimise. The second reason, which applies in particular to the settings controlling model capacity, is that it is not appropriate to learn this hyperparameter on the training set. Such hyperparameters, if learnt on the training set, would always select the maximum possible model capacity and thus bring about overfitting. To illustrate this phenomenon, let us consider the polynomial example in fig. 2.4. If the degree of the polynomial function used to fit the data is regarded as a trainable parameter, the learning algorithm will tend to increase the degree until the model perfectly fits the data, resulting in a small training error, but poor generalisation performance. This is why the polynomial degree should be seen as a hyperparameter and chosen independently of the training.

The simplest way to select proper hyperparameters is to train different models (described by different hyperparameters) and choose the setup leading to the smallest generalisation error [19]. As in the case of early stopping, however, hyperparameter tuning cannot depend on the test examples and as such, the generalisation errors used to compare the trained models are computed on the validation set. In the polynomial example, the algorithm is to be trained on the training examples for different polynomial degrees. The polynomial leading to the best generalisation error on the validation set determines the best version of the model, whose final performance is evaluated on the test set.

Although hyperparameter search is still performed manually in most of the cases,

the field of hyperparameter optimisation (HO) has been receiving an increasing attention over the last years. While the most intuitive and widely spread HO methods are grid search and random search [13], several alternative procedures that aim at automatically finding the best hyperparameter setting in a low-cost manner have recently emerged [189, 68].

2.2 Deep learning theory

Although most traditional machine learning algorithms perform well on a wide variety of important tasks, they have not succeeded in solving central problems of AI, such as object or speech recognition. Partly motivated by the failure of classic algorithms on such tasks, a new branch of machine learning was developed, with greater power and flexibility, and was given the name of deep learning due to the depth of its models. Such a class of algorithms provides a potent framework in which previous obstacles, such as the generalisation to new examples when working with high-dimensional data, or the learning of complicated functions in high-dimensional spaces, can be overcome.

Deep feedforward networks, also called *multilayer perceptrons* (MLPs), are the quintessential deep learning models. In this section, we introduce the mathematical theories underlying such structures and the algorithms used to train them. We also provide some historical perspectives about neural network development and briefly introduce more advanced ANN architectures.

2.2.1 The power of deep learning

One way to introduce feedforward networks is to consider some of the limitations of simpler machine learning algorithms and contemplate how to overcome them. Let us start with the simplest configuration: linear models. Linear models, such as linear regression or logistic regression, present the advantage of fitting efficiently and dependably, whether by closed form or by convex optimisation. However, they also have the evident defect their model capacity is restricted to linear functions. This implies amongst others, that the model cannot understand (and thus take advantage of) the interaction between any two input variables.

To overcome this first limitation, and extend linear models to represent nonlinear functions of x, it is possible to apply the linear model not to x itself, but to a transformed input $\Phi(x)$, Φ being a nonlinear transformation. Such a function Φ can be regarded as providing a new representation of x, for which it is hoped that the linear model can be successfully applied. Fig. 2.9 shows an illustrative example of such a process.

The question is then: how to choose the mapping Φ?

1. A first option, which remained the dominant approach until the advent of deep learning, consists in manually engineering Φ. Such a strategy however gives rise, at the cost of tremendous amounts of time and energy, to models that are designed for specific tasks only, with little transfer between domains.

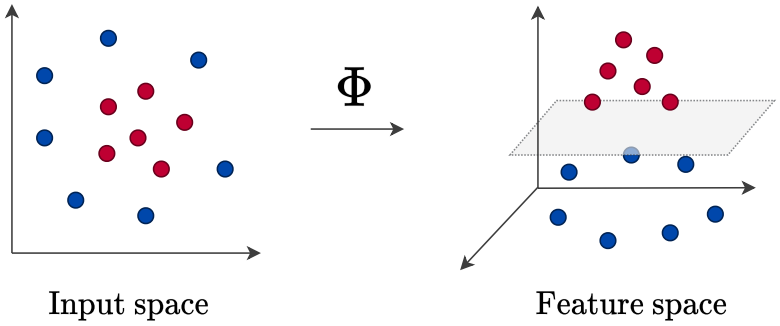


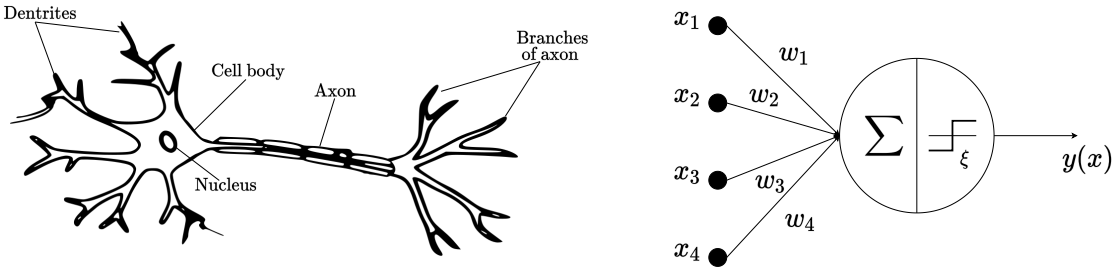
Figure 2.9: Illustration of the trick that allows linear models to be used on initially nonlinear functions. The transformation Φ applied to the inputs x provides a new representation thereof in which a linear model can be successfully applied.

2. A second approach, which was *inter alia* employed in machine learning algorithms such as SVM or more generally in *kernel methods*, consists in using a very generic Φ. In that case, one usually selects Φ amongst a variety of finite or infinite-dimensional popular functions whose properties have already been studied in a broad context. If Φ is chosen of high enough dimension, the model capacity is large enough to allow a proper fitting of the training set, but generalisation to the test set often remains poor. In fact, very generic mappings usually depend on the principle of local smoothness, that is, on the implicit assumption that the function being learnt should not change significantly within a small region on the input space, and do not encode enough prior information to solve advanced problems.

3. The breakthrough proposed by deep learning approaches is to learn Φ. In that case, the mapping Φ is described by a set of parameters that are to be trained through an optimisation process, such as the ones described in section 2.1.5, until it offers a good representation transformation. This strategy is the only one of the three that gives up on the convexity of the optimisation problem, but the advantages outweigh the harms. It can capture the benefit of the first approach: human practitioners can encode their knowledge to help generalisation by designing families of candidate functions Φ that are likely to perform well, but they do not need to find the exact right function, providing a good general function family is enough. Moreover, this strategy can also capture the benefit of the second approach, as Φ can be chosen to be highly generic. Thanks to the countless possibilities it opened up in the learning domain, it has nowadays become the predominant approach.

Learning the function Φ in deep learning can take different forms. In the specific context of artificial neural networks, this process draws inspiration from biological learning mechanisms.

In human brain, the nervous system is composed of cells, which are referred to as *neurons*. The neurons are connected to one another thanks to *dendrites* and *axons*, as



(a) Biological neuron (b) Formal neuron

Figure 2.10: Schematic illustration of a biological neuron (left), and the formal neuron (right). The functioning of the latter was inspired by that of the former, inasmuch as it receives input signals from previous neurons, applies a nonlinear function to the weighted sum of those signals, and sends the resulting activation value to the next neurons.

illustrated in fig. 2.10a. The regions connecting axons and dendrites, called *synapses*, allow to convey signals from a neuron to the next one. Roughly speaking, a neuron receives multiple signals from a set of other neurons with which it shares synapses. Each input signal has its own amplitude, which is weighted by the so-called *synaptic weight* that defines the level of influence of each synapse. If the sum of these weighted signals is high enough, the neuron is *activated*, meaning that it fires an output signal to the following neurons; it remains inactivated otherwise. Studies showed that the strengths of synaptic connections (the synaptic weights) often change in response to external stimuli, and it is this continual modification that allows living organisms to learn [3].

Inasmuch as their structure is highly inspired by the aforementioned biological mechanism, ANNs are commonly believed to reproduce human brain functioning and their popularity has undeniably benefited from this advantageous biological comparison. The parallel between ANNs and neuroscience is however often criticised as a poor caricature of the workings of the human brain, and in spite of the popular belief, ANNs are not designed to be realistic models of biological function. Nonetheless, it is true that the principles of neuroscience have proven useful in designing neural networks and motivating innovative architectures, such as CNNs.

Fundamentally, ANNs are built as networks of interconnected computational nodes, called neurons in reference to the biological units they draw inspiration from. The neurons are connected to one another through weights, which are the equivalent of synaptic weights in living organisms. An artificial neuron receives several signals from its inputs neurons, process them in order to compute its own activation number, and propagates the computed value to its output neurons, using the weights as intermediate parameters (cf fig. 2.10b). The process of learning then comes down to changing the weights connecting the neurons. Just as external stimuli are needed for biological organisms to learn, ANNs receive external stimuli by means of the training set and, by exploiting the training input-output pairs through some optimisation process, they can gradually adjust the weights until they reach a satisfactory generalisation error.

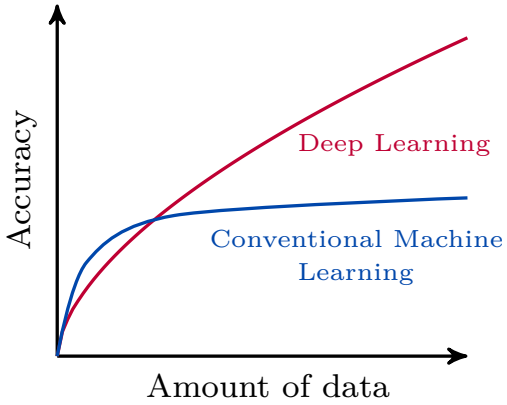


Figure 2.11: An illustrative comparison between the accuracy of a conventional machine learning algorithm and that of a large ANN. Deep learners are essentially more appealing than traditional methods as long as they are provided sufficiently large datasets and powerful computers. Thanks to the recent dramatic increase in data availability and computational power, deep learning technology has become the technology of choice in a large number of applications. Figure inspired from [3].

A network can be described by three main characteristics: i) the type of operations the neurons perform, ii) the values of the weights in the connections between neurons, iii) its architecture, that is, the number of nodes, their arrangement in multiple layers, the type of connections. When a neural network is used in its most basic form, without combining multiple units, the learning system often reduces to a classical machine learning model. The real power of an ANN manifests when these elementary computational nodes are hooked up and the weights are simultaneously trained, using their dependencies on one another. The predominant (and so far most powerful) way to combine such units is to incorporate them in a multi-layered graph. By augmenting the number of neurons within each layer and the number of those layers, one increases the ability of the model to learn more complicated functions. The fact that neural networks offer such a simple way to adjust the complexity of a model, which is then chosen in function of the availability of training data and computational capacity, gives them a real asset over traditional machine learning algorithms. Actually, most of the recent success of ANNs are attributable to the fact that the ever-growing power of modern computers and data availability have outgrown the limitations of classic machine learning systems, which fail to make the most of what is now possible [3]. Fig. 2.11 illustrates such a situation. While traditional machine learning may achieve better results for small datasets, due to greater ease to interpret the model and the possibility to hand-craft interpretable features, it is clearly outperformed by deep learning as soon as the dataset or the input space are high-dimensional.

Nowadays, most tasks that involve mapping an input vector to an output vector can be performed by deep learning algorithms, as long as they are provided sufficiently large models and sufficiently large datasets of labelled training samples. For the future, the rapid advances associated with the ever increasing amount of data collections, the

development of increasingly powerful computers that can handle such large datasets, and the ever greater speed of computation that results in reduced training and testing times, have lead to very optimistic perspectives for deep learning.

2.2.2 From perceptron to deep MLPs

The origin of the neural network field is usually attributed to Warren McCullock and Walter Pitts [146], who showed in 1943 that networks of artificial neurons could, theoretically, compute any logical or arithmetic function. Their work inspired the *perceptron* network, developed by Frank Rosenblatt [176] in the late 1950s. Although such an algorithm generated a great deal of interest in the machine learning community due to its ability to perform pattern recognition, it rapidly became clear that the basic perceptron network could solve only a limited class of problems. The necessity to consider more complex structures gave later rise to the *multilayer perceptrons*. This section aims at giving an overview of such an evolution of the ANNs, from their simplest form to the architectures widely used nowadays.

2.2.2.1 Formal neuron

In their work, McCulloch and Pitts defined the simplest form of the artificial neuron, or *formal neuron*, according to the following principle: the inputs of a neuron are multiplied by weights, and the weighted sum of those signals is compared to a threshold value. If the computed value is larger than the threshold, the neuron outputs 1; it outputs -1 (or 0) otherwise. In mathematical terms, if we denote $x \in \mathbb{R}^p$ the input vector, $y \in \{1, -1\}$ its corresponding output, $w \in \mathbb{R}^p$ the vector of weights and $\xi \in \mathbb{R}$ the threshold value, the output of the formal neuron w.r.t. its input reads:

$$y(x) = g\left(\sum_{i=1}^{p} w_i x_i - \xi\right) = g(w^T x - \xi), \tag{2.29}$$

where g is called the *activation function* of the neuron. It can be, for instance, the Heaviside step function, defined in eq. (2.4), which outputs a value $g(t) \in \{0, 1\}$, or else, the function $sign : \mathbb{R} \to \{-1, 1\}$, defined as:

$$g(t) = sign(t) = \begin{cases} 1 & \text{if } t \geq 0 \\ -1 & \text{if } t < 0. \end{cases} \tag{2.30}$$

The structure of the formal neuron is shown in fig. 2.10b.

2.2.2.2 Rosenblatt's single layer perceptron

The first practical application of artificial neurons was proposed by Rosenblatt in 1958. He reinterpreted the formal neuron as a binary classifier, able to sort the elements of a given dataset into two groups. Indeed, the artificial neuron, even in its simplest form, is able to perform binary classification by assigning to a generic vector $x \in \mathbb{R}^p$ described by p features, the value $y(x) = 1$ or $y(x) = -1$ according to the sign of the

linear function $w^T x - \xi$. The accuracy of such a classification then relies on the value of the weights w and threshold ξ. Rosenblatt's major achievement was precisely the intuition that those parameters can be learnt from data. He designed a fairly simple algorithm that allows the training of w and ξ through an iterative process based on a dataset of S input-output pairs:

$$\mathcal{S} = \{(x_i, y_i) \mid x_i \in \mathbb{R}^p,\ y_i \in \{-1, 1\},\ i = 1, \ldots, S\}. \tag{2.31}$$

Once trained on the dataset $\mathcal{S}$, such a network can then be used to classify new inputs x that do not belong to $\mathcal{S}$. The neural architecture thus devised was given the name *perceptron*.

The training algorithm of the perceptron network, described in algorithm 2, was designed according to the following considerations. A proper binary classification will be achieved on the training samples as long as the parameters w and ξ are such that:

$$\begin{cases} w^T x_i - \xi > 0 & \text{if} \quad y_i = \ \ 1, \\ w^T x_i - \xi < 0 & \text{if} \quad y_i = -1, \end{cases} \qquad s = 1, ..., S. \tag{2.32}$$

From a geometric point of view, system (2.32) can be interpreted as the search for the separating hyperplane $\mathcal{H} = \{x \in \mathbb{R}^p \mid w^T x = \xi\}$ that divides the training samples into the two sets:

$$\mathcal{A} = \{x_i \mid (x_i, y_i) \in \mathcal{S},\ y_i = 1\} \quad \text{and} \quad \mathcal{B} = \{x_i \mid (x_i, y_i) \in \mathcal{S},\ y_i = -1\}.$$

The existence weights w and threshold ξ that solve (2.32) is ensured if and only if the sets $\mathcal{A}$ and $\mathcal{B}$ are linearly separable. If this is indeed the case, solving problem (2.32) is equivalent to solving:

$$\begin{cases} w^T x_i - \xi > 0 & \text{if} \quad x_i \in \mathcal{A} \\ w^T x_i - \xi < 0 & \text{if} \quad x_i \in \mathcal{B}. \end{cases} \qquad s = 1, ..., S. \tag{2.33}$$

At this point, it is common practice to add fictitious components $x_0 = 1$ and $w_0 = -\xi$ to the input and weight vectors so that they read $x_i = [1, x_{i_1}, \ldots, x_{i_p}]^T, \forall i \in \{1, \ldots, S\}$ and $w = [-\xi, w_1, \ldots, w_p]$. In this manner, the threshold is regarded an additional *bias* term with weight ξ, and the system to be solved comes down to:

$$\begin{cases} w^T x_i > 0 & \text{if} \quad x_i \in \mathcal{A} \\ w^T x_i < 0 & \text{if} \quad x_i \in \mathcal{B}. \end{cases} \qquad s = 1, ..., S. \tag{2.34}$$

in which it can be assumed, without loss of generality, that $\|x_i\| = 1, \quad \forall i = 1, \ldots, S$.

In order to automatically determine the parameters w that meet the conditions in eq. (2.34), Rosenblatt provided an iterative training algorithm (algorithm 2) that adjusts its parameters by repeatedly observing the samples of the training set, until all the samples are classified properly. At each iteration, the parameter vector w is updated if the currently observed sample is misclassified, by adding a correction term to the value of w.

Algorithm 2 Perceptron training algorithm, from [82]

Input: $\mathcal{S} = \{(x_i, y_i) |\ x_i \in \mathbb{R}^{p+1} \text{ s.t. } \|x_i\| = 1,\ y_i \in \{-1, 1\},\ i = 1, \ldots, S\}$ training set
Set $w^{(0)} = 0$, $k = 0$, nbclass= 0.

while $nbclass < S$ **do**
  for $i = 0, 1, \ldots$ **do**
    if $sign\left(w^{(k)^T} x_i\right) = y_i$ **then**
      nbclass = nbclass +1
    else
      $w^{(k+1)} = w^{(k)} + y_i x_i$
      k = k+1
    end if
  end for
  if nbclass $< S$ **then**
    nbclass = 0
  end if
end while

It can be demonstrated that if the set $\mathcal{A}$ and $\mathcal{B}$ are linearly separable, the perceptron algorithm can determine a weight vector $\bar{w}$ in a finite number of iterations [82], such that all the training samples are classified correctly, that is, satisfying:

$$y_i = sign(\bar{w}^T x_i), \quad i = 1, \ldots, S. \tag{2.35}$$

However, although it proves efficient to solve a variety of simple problems, the perceptron network exhibits a rather arbitrary behaviour as soon as the data is not linearly separable, and thus suffers from major limitations. A well-known example is provided by the XOR function, a logical operation on binary inputs that returns 1 if and only if the input features differ, and 0 otherwise. The image space of such a function is represented on fig. 2.12. It can be intuited that no linear model is able to represent the XOR function, as there is no straight line able to separate the outputs into two sets of same value.

Some of those limitations can be overcome by applying a nonlinear transformation Φ to the input space in order to represent the data in a linearly separable space, as already mentioned in section 2.2.1. In that case, the input-output mapping of the formal neuron becomes:

$$y(x) = g\left(\sum_{i=1}^{p+1} w_i \Phi_i(x)\right). \tag{2.36}$$

Even such a possibility, already proposed in Rosenblatt's perceptron, is still subject to important limitations. The restricted expressiveness of perceptrons was in particular criticised in the famous book by Minsky and Papert [148], which resulted in a drastic lessening of the scientific interest in neural networks in the 1970s.

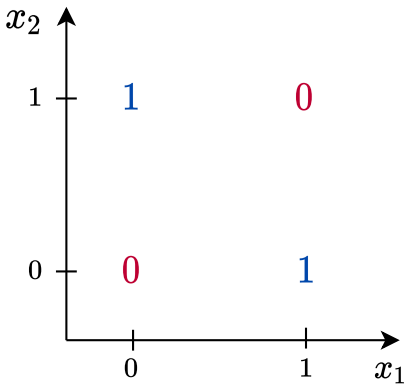


Figure 2.12: XOR function.

2.2.2.3 Multilayer perceptrons

The limitations of the perceptron, that is, of networks containing only one layer of formal neurons, motivated the study of more complex architectures. The development of MLPs, which consist of several layers of artificial neurons connected in chain, reinvigorated the field of deep learning, giving rise to a second wave of development of neural networks in the 1980s. This new kind of ANNs, also called *feedforward neural networks* inasmuch as they propagate one-way information from inputs to outputs, without feedback connections, indeed provided a way to approximate a very broad range of functions and thus to solve a wide variety of complex problems. Although they are not the only existing ANN architecture, MLPs constitute the quintessential deep learning models and offer a very good initial framework to understand the fundamental principles underlying ANN functioning.

MLP architecture A MLP consists of $N \geq 2$ layers of artificial nodes, organised in a chain structure. The first layer, which is actually not counted as part of the N layers of the MLP inasmuch as it has no processing capacity, is called the *input layer*: its role is to contain the features of the input examples of the dataset $x_i \in \mathbb{R}^p,\ i = 1, \ldots, S$. It is therefore composed of p units.

The last layer, or the *output layer*, returns the value(s) computed by the ANN and has as many nodes as outputs desired by the user. Even in the case of classification, this number may not necessarily be 1: for instance in multi-class classification, the ANN is usually designed not to output the exact predicted class but rather the probability to belong to each of the k classes of interest and has thus k output nodes.

The remaining $N - 1$ internal layers are said to be *hidden*, since the training data does not give explicitly information about the desired output for each of these layers. Each hidden unit, that is, each node of a hidden layer, represents a vector-to-scalar function. It receives inputs from the neurons of the previous layer, computes its own activation value, and sends the output as a contribution to the inputs of the neurons in the subsequent layer. As such, MLPs are said to be *fully connected*: all the neurons of a layer are connected to the neurons of the next layer. An example of MLP is

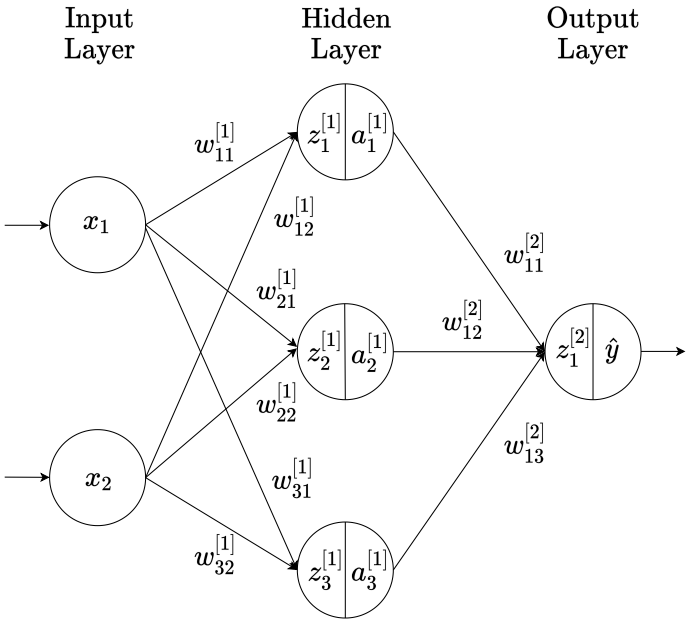


Figure 2.13: Example of a feedforward neural network with depth $N = 2$ and a width of 3. The oriented edge between the j^{th} neuron of the $(n-1)^{\text{th}}$ layer and the k^{th} neuron of the n^{th} layer has a weight $w_{kj}^{[n]}$. Although they are not explicitly represented, ANNs usually involve additional biases. The values reported inside the nodes of the graph are the preactivation and postactivation values resulting from the forward operations of the neural network applied to an input $(x, y) \in \mathcal{S}$ of the dataset. The predicted output is denoted $\hat{y}$.

illustrated in fig. 2.13.

The *depth* of the model, which strongly influences the capacity of the MLP, is given by the length N of the chain. As for its *width*, it is determined by the number of hidden units in the largest hidden layer: $\max_{n \in [\![1, N-1]\!]} d_n$, where d_n is the dimension of the n^{th} layer. As mentioned in section 2.1.6, both the depth of the MLP and the number of hidden units in each layer are hyperparameters, meaning that they are not learnt through the training process and instead need to be tuned by the user.

Weights and hidden units The oriented connections between nodes are characterised by *weights*, which define the influence of the information respectively propagated from the neurons of a layer to the neurons of the next one. We will denote $w_{kj}^{[n]}$ the weight of the connection between the j^{th} neuron of the $(n-1)^{\text{th}}$ layer and the k^{th} neuron of the n^{th} layer (cf fig. 2.13). Although they are usually not explicitly represented on ANN schemes, biases are also included as part of the parameters to be learnt. As one usually adds one bias per layer, we will use the notation $b_k^{[n]}$ to represent the contribution of such an additive parameter to the k^{th} neuron of the n^{th} layer.

A hidden unit makes use of those weights and biases to compute a weighted sum of the inputs it receives from the previous layer. However, if its role were limited to computing such a linear combination of its inputs, an MLP, even deep, would come

down to a simple linear model, and one would miss all the interest of using a deep learning structure. The ability of MLPs to represent complex nonlinear functions substantially rests upon the fact that each unit applies a nonlinear function to the weighted sum of its inputs before outputting it to the next neurons. Such functions are called *activation functions* and will be denoted $g^{[n]} : \mathbb{R} \to \mathbb{R}$. We respectively call *preactivation value* and *(post)activation value* the values computed before and after applying the activation function. An overview of the most commonly used activation functions is given in the next paragraph.

Formally, for a given input vector $x_i \in \mathbb{R}^p$, the output ${a_{i,k}}^{[n]}$ of the k^{th} neuron of the n^{th} layer is obtained as follows:

$$\begin{cases} {z_{i,k}}^{[n]} = \sum_{j=1}^{d_{n-1}} w_{kj}^{[n]} a_{i,j}^{[n-1]} + b_k^{[n]} \\ {a_{i,k}}^{[n]} = g^{[n]}\left({z_{i,k}}^{[n]}\right), \qquad \text{for } k = 1, \dots, d_n, \quad n = 1, \dots, N. \end{cases} \tag{2.37}$$

provided that in the input layer ${a_{i,k}}^{[0]} = x_{i,k}$ for $k = 1, \dots, d_0 = p$. Formula (2.37) can be reformulated in matrix form, as follows:

$$\begin{cases} {z_i}^{[n]} = W^{[n]} a_i^{[n-1]} + b^{[n]} \\ {a_i}^{[n]} = g^{[n]}\left({z_i}^{[n]}\right), \qquad \text{for} \quad n = 1, \dots, N. \end{cases} \tag{2.38}$$

where $W^{[n]} \in \mathbb{R}^{d_n \times d_{n-1}}$ is the weight matrix of the k^{th} layer and $b^{[n]} \in \mathbb{R}^{d_n}$ is the bias vector. Thus, given an input vector $x_i \in \mathbb{R}^p$ that determines the values of the neurons ${a_i}^{[0]}$ in the first layer, one can use eq. (2.38) to compute the final output $\hat{y}_i = {a_i}^{[N]}$ of the neural network.

Activation functions The design of activation functions is a highly active area of research. So far, there are not many definitive guiding theoretical principles for the choice of such functions and although a wide variety of nonlinear functions may perform well, only a small subset of them is actually used in practice.

Sigmoids constitute a first family of popular activation functions. They monotonically increase between two finite values by describing a S-shaped curve, or sigmoid curve. This family mainly encompasses the *logistic function*:

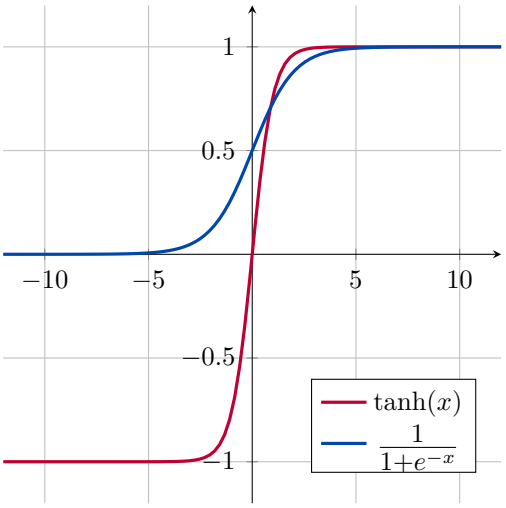
$$g(t) = \frac{1}{1 + e^{-t}}, \quad t \in \mathbb{R}, \tag{2.39}$$

and the *hyperbolic tangent function*:

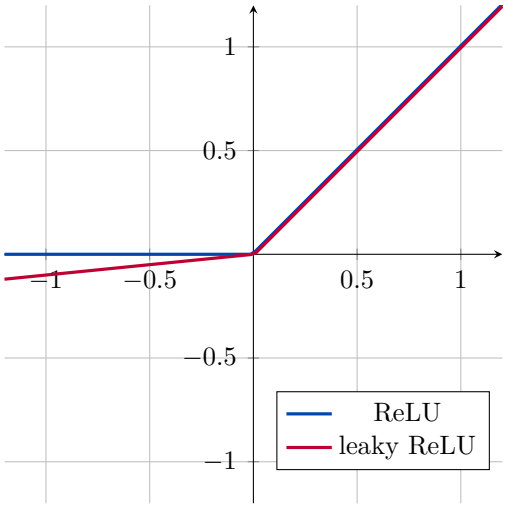
$$g(t) = \tanh(t) = \frac{e^t - e^{-t}}{e^t + e^{-t}}, \quad t \in \mathbb{R}, \tag{2.40}$$

which are plotted on fig. 2.14a. As it can be observed, they essentially differ in the range of their outputs: while the former returns a value in $(0; 1)$, the latter outputs a number in $(-1; 1)$.

For a long time, sigmoidal functions have been used as the default activation functions as they offer appealing properties, amongst which their continuity, differentiability and the fixed range of their output. However, they also present a major drawback:



(a) Sigmoid activation functions. In blue: logistic function, in red: hyperbolic tangent.



(b) ReLU-like functions. In blue: ReLU function, in red: leaky ReLU with $\nu = 0.1$.

Figure 2.14: Graphics of different activation functions.

they saturate across most of their domain, as soon as $|t|$ is large, and are only strongly sensitive to their input when t is close to 0. As this widespread saturation phenomenon may make gradient-based learning very difficult, the use of such functions in hidden units is now discouraged. They may only be employed in the output layer if suitable.

The sigmoidal functions have mainly been supplanted by a second family of activation functions, which provide better performance: the *Rectified Linear Units* (ReLU) and their generalisations. The ReLU function is defined by:

$$g(t) = \max\{0, t\}, \quad t \in \mathbb{R}. \tag{2.41}$$

Being a piecewise linear function, ReLU preserves many of the properties that make linear units easy to optimise with gradient-based methods. It also has the advantage of being less computationally expensive than sigmoidal functions since it involves simpler mathematical operations. In the overall, ReLU is considered as an excellent default choice of hidden unit and has indeed proved its efficiency on many problems [154, 230]. It is to be noted that ReLU is not differentiable at $t = 0$, and may thus appeared as not eligible in the framework of gradient-based learning. In practice, gradient descent still performs well enough in spite of this indifferentiability, since usually ANN training algorithms do not reach a local minimum of the cost function but instead merely reduce its value significantly. Thus, it is not expected to arrive at a point where the gradient is zero and the fact that the true minima of the cost function correspond to points with undefined gradient is considered as acceptable. The actual drawback of ReLUs rather rests upon the fact that they cannot learn via gradient-based method when their preactivation value is negative.

Various generalisations of ReLU have been proposed to overcome this zero-gradient plateau and guarantee the possibility to learn even from nonpositive preactivation

values. Part of them are based on using a nonzero slope ν when $t < 0$:

$$g(t, \nu) = \max\{0, t\} + \nu \min\{0, t\}, \quad t \in \mathbb{R}. \tag{2.42}$$

Examples of such a strategy include the *absolute value rectification* [114], which fixes ν to -1, *leaky ReLU* [142] which fixes ν to a small value like 0.01, and *parametric ReLU* [91] which treats ν as a learnable parameter. Standard ReLU and its leaky variant are plotted on fig. 2.14b.

So far, we have mainly discussed the activation functions employed for hidden units, but the functions used in the output layer may be different from the former. Such a choice is actually tightly linked to the choice of the cost function and the form of the desired output. The most popular functions encompass *inter alia* linear functions, since nonlinearity is not necessary in the last layer, sigmoid functions which prove particularly useful for binary classification, and the *softmax function*, which can be employed to represent a probability distribution over a discrete variable with k possible values. Formally, the latter is given by:

$$g(t)_k = \text{softmax}(t)_k = \frac{e^{t_k}}{\sum_{j=1}^{d_N} e^{t_j}}, \quad k = 1, \dots, d_N, \quad t \in \mathbb{R}^{d_N}, \tag{2.43}$$

where t is a multi-dimensional vector composed of all the preactivation values of the last layer.

Training an MLP Once the global architecture of the MLP as well as its activation functions have been chosen, one can see the network as a composition of functions connected in chain that outputs $\hat{y}_i = f_\theta(x_i)$, θ denoting the set of all trainable parameters $\{(W^{[1]}, b^{[1]}), \dots, (W^{[N]}, b^{[N]})\}$. For the predicted value $\hat{y}_i$ to be close to the true label y_i, the weights and bias are intended to be trained through an optimisation process, following the procedure explained in section 2.1. Namely, a cost function $\mathcal{L}$ is designed, so that its minimisation

$$\min_\theta \frac{1}{S} \sum_{i=1}^{S} \mathcal{L}(x_i, y_i, f_\theta(x_i)) \tag{2.44}$$

over the provided training set $\mathcal{S} = \{(x_i, y_i) \mid i = 1, \dots, S\}$ leads to an accurate approximation of the mapping between the input vectors and their corresponding outputs. Again, the neural network is not only expected to perform well on the training set, but also and above all to provide a low test error, thus demonstrating good generalisation abilities.

The minimisation of the objective function (2.44) is usually performed by means of gradient-descent techniques and the computation of the gradient w.r.t. the parameters θ, needed at each iteration of such methods, is made possible by the back-propagation algorithm. The latter is described in greater detail in section 2.2.3.

Fundamentally, one of the largest differences between neural networks and traditional machine learning algorithms is that the nonlinearity of ANNs leads most interesting cost functions to become nonconvex. This implies that the gradient-based optimisers used to train ANNs are not guaranteed to converge to the global minimum

and instead merely drive the cost function to a very low value. Furthermore, the success of ANN optimisation process strongly depends on the choice of the initial value of the parameters, as well as some hyperparameters, such as the learning rate. Those challenges and some of the popular methods developed to remedy them are presented in section 2.2.4.

MLPs as universal approximators It has been demonstrated [102] that a feedforward network with a linear output layer and at least one hidden layer of nonlinear units (with a wide ranging choice of 'squashing' functions, such as the logistic sigmoid activation function) can approximate any continuous function on a compact set of $\mathbb{R}^p$. Furthermore, a MLP may also approximate any function mapping from any finite dimensional discrete space to another. As a consequence, MLPs are often referred to as *universal function approximators*.

While the original theorem was first stated for MLPs with activation functions that saturate for both very positive and very negative inputs, the universal approximation theorem has also been demonstrated for a broader class of activation functions, which includes the ReLU function [135].

In short, the universal approximation theorem guarantees that, regardless of the function that is aimed to be learnt, a large MLP will be able to represent this function. Such a theoretical claim, however, is not always easy to translate into practical usefulness, for several reasons. First, there is no guarantee that the learning system will actually find that function. The optimisation algorithm may not converge to the value of the parameters that corresponds to the wanted function, whether because it finds another local minimum, or because it selects an improper function as a result of overfitting. Secondly, although the universal approximation theorem states that there exists an ANN large enough to achieve any degree of complexity, it does not give any information about how large the network needs to be. In the worst-case, up to an exponential number of hidden nodes may be required [9]. As a consequence, although single-layer feedforward networks are theoretically sufficient to approximate any function, their hidden layer may need to be unfeasibly large and may not be concretely trainable. For this reason, deeper MLPs are usually preferred, as they reduce the number of hidden units required in each layer to represent the desired function, as well as the overall number of parameters. A greater depth has furthermore shown better generalisation results in a wide variety of tasks (see [77] and references therein), suggesting that the use of deep architectures does express a useful prior about the kind of function the model learns.

2.2.2.4 Other architectural considerations

So far, we have described ANNs as being simple chains of layers, mainly characterised by their depth and width. However, neural networks are not restricted to this architecture, and actually show a far greater diversity.

In general, connecting the layers in a chain is indeed the most common practice, but many architectures have also explored alternative connections. The *residual ANNS*

[92], for instance, contain a main chain but not only: extra architectural features, such as skip connections going from layer n to layer $n + 2$ or higher have been added with the aim to facilitate the back-propagation of the gradient through the layers of the model.

Many of the architectural innovations have emerged in response to the difficulties of MLPs to solve specific tasks. For example, standard neural networks fail in applications in which the data is sequential, such as text processing, speech recognition or DNA sequences. In such cases, the output strongly depends on all the previous computations, but the static architecture of feedforward ANNs do not allow to take advantage of this information. *Recurrent neural networks* (RNNs) provide a way to overcome this limitation by using feedback cycles, or recurrent connections, which enable them to learn temporal dependencies.

Another key consideration of architecture design relates to the kind of connection that links two consecutive layers. So far, we referred to fully connected neural networks, in which every input unit is connected to every output unit. In many specialised ANNs, the possibility to use fewer connections has been considered, in such a way that each input unit is actually connected to only a small subset of units in the output layer. Inasmuch as they reduce the number of connections, these approaches result in a smaller number of parameters to learn and therefore require much less computations. They are generally highly problem dependent, but have given rise to very efficient models. *Convolutional neural networks* (CNNs), which have revolutionised the field of computer vision by using specialised patterns of sparse connections based on convolutions, are a case in point. A detailed description of their functioning is provided in section 2.3.

2.2.3 Back-propagation algorithm

Training a deep learning system essentially means solving the minimisation problem (2.44) in order to find suitable values for the parameters θ. Most of the time, this is performed thanks to gradient-descent methods, which require computing the gradient of the objective function w.r.t. to θ. In single-layer neural networks, this process proves relatively straightforward as the objective function is built as a direct function of the parameters and thus allows easy gradient computation. In the case of deeper ANNs, however, the loss is a complicated composition function of all the parameters in the hidden layers, making the gradient computation much more arduous. Actually, one of the reasons why neural network research was historically suspended for many years is precisely because no efficient algorithm had been devised to train multi-layer ANNs. The groundbreaking back-propagation algorithm, mainly attributed to David Rumelhart and James McClelland [177], provided a powerful response to that challenging topic.

In this section, we describe such a gradient computation method and illustrate it in the specific case of MLPs.

2.2.3.1 Chain rule of calculus

The training of an ANN is essentially divided into two phases. The first phase consists in computing an output $\hat{y}$, given an input x of the training set. This results in a forward cascade of computations across the layers, based on the current set of parameters θ. The final predicted output can then be quantitatively compared to that of the training instance y thanks to the loss function. This first phase, called *forward propagation*, is formulated in eq. (2.38) and illustrated in fig. 2.13 of section 2.2.2.3. The second phase of the training, called *back-propagation*, is intended to learn the gradients of the cost function w.r.t. the different parameters of the ANN. To do so, the information from the cost function flows backward through the network according to the chain rule of differential calculus. The computed gradients are then used to update the weights and biases of the model.

It is to be mentioned that the back-propagation algorithm, inasmuch as it is a generic method for computing derivatives, is actually not restricted to ANNs and on the contrary, can be applied to a broad range of machine learning tasks.

It essentially rests upon the chain rule of calculus, which states how to compute the derivatives of composite functions. Formally, given $x \in \mathbb{R}^p$, $y \in \mathbb{R}^q$ and two functions $g : \mathbb{R}^p \to \mathbb{R}^q$, $f : \mathbb{R}^q \to \mathbb{R}$, if $y = g(x)$ and $z = f(y) = f(g(x))$, then according to the chain rule:

$$\frac{\partial z}{\partial x_k} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_k}, \quad k = 1, \ldots, p. \tag{2.45}$$

which can be equivalently written in vector notation:

$$\nabla_x z = \left(\frac{\partial y}{\partial x} \right)^T \nabla_y z. \tag{2.46}$$

where $\frac{\partial y}{\partial x}$ is the $q \times p$ Jacobian matrix of g.

The back-propagation algorithm consists in performing such Jacobian-gradient products for each operation in the computational graph associated to the network. The great efficiency of such an algorithm lies in the specific order in which it executes the operations while travelling through the graph. In particular, the fact that it iterates backwards from the output layer towards the input layer enables it to avoid redundant calculations of intermediate terms involved in the chain rule: it is called *dynamic programming*.

2.2.3.2 Illustration of back-propagation in MLPs

In order to get a better intuition of back-propagation, let us illustrate its functioning through the example of an MLP with N layers. For the sake of simplicity, we consider a single training example at a time, denoted $(x, y) \in \mathcal{S}$, although again, one might want to train the learning system on multiple examples simultaneously.

We recall that the system defining the forward propagation is given by:

$$\begin{cases} z^{[n]} = W^{[n]} a^{[n-1]} + b^{[n]}, \\ a^{[n]} = g^{[n]}(z^{[n]}), \qquad \text{for} \quad n = 1, \dots, N. \\ a^{[0]} = x, \quad \hat{y} = a^{[N]}. \end{cases} \tag{2.47}$$

where $z^{[n]}$ is the preactivation value of layer n, and $a^{[n]}$ is its (post)activation value. The predicted output $\hat{y} = f_\theta(x)$ and the corresponding loss $\mathcal{L}(x, y, \hat{y})$ can therefore be computed with algorithm 3.

Algorithm 3 Forward propagation through a typical MLP, and computation of the loss function. For the sake of simplicity, only a single training example $(x, y) \in \mathcal{S}$ is considered, although in practical applications, multiple examples are often processed simultaneously.

Input: $(x, y) \in \mathcal{S}$, training example
N, network depth
$\theta = \{(W^{[1]}, b^{[1]}), \dots, (W^{[N]}, b^{[N]})\}$, network parameters (weights and biases)
$\{g^{[1]}, \dots, g^{[N]}\}$, network activation functions

$a^{[0]} = x$
for n=1,...,N **do**
  $z^{[n]} = W^{[n]} a^{[n-1]} + b^{[n]}$
  $a^{[n]} = g^{[n]}(z^{[n]})$
end for
$\hat{y} = a^{[N]}$
Compute the loss function $\mathcal{L}(x, y, \hat{y})$

Once the loss has been computed, and knowing the preactivation and postactivation values of each hidden layer, one can apply the chain rule of calculus to compute the gradients of the loss w.r.t. the parameters of the network $\nabla_\theta \mathcal{L}(x, y, f_\theta(x))$. The computations are performed backwards, in order to reduce the gradient to a product of straightforward, easy-to-compute derivatives. Thus, the very first step of the backpropagation algorithm consists in computing the gradient of the loss w.r.t. the output of the network $\nabla_{\hat{y}} \mathcal{L}(x, y, f_\theta(x))$. From this point, the gradient of the loss w.r.t. to the weights and biases of a particular hidden layer n is given by propagating the derivatives backwards according to the formulas:

$$\begin{cases} \nabla_{W^{[n]}} \mathcal{L}(x, y, f_\theta(x)) &= \nabla_{z^{[n]}} \mathcal{L}(x, y, f_\theta(x)) {a^{[n-1]}}^T, \\ \nabla_{b^{[n]}} \mathcal{L}(x, y, f_\theta(x)) &= \nabla_{z^{[n]}} \mathcal{L}(x, y, f_\theta(x)), \qquad \text{for } n = N, \dots, 1. \end{cases} \tag{2.48}$$

provided that the derivatives of the loss w.r.t. to the pre- and post-activation values, for $n = N - 1, \dots, 1$, are given by:

$$\begin{cases} \nabla_{z^{[n]}} \mathcal{L}(x, y, f_\theta(x)) &= \nabla_{a^{[n]}} \mathcal{L}(x, y, f_\theta(x)) \odot {g^{[n]}}'(z^{[n]}), \\ \nabla_{a^{[n]}} \mathcal{L}(x, y, f_\theta(x)) &= {W^{[n+1]}}^T \nabla_{z^{[n+1]}} \mathcal{L}(x, y, f_\theta(x)), \\ \nabla_{a^{[N]}} \mathcal{L}(x, y, f_\theta(x)) &= \nabla_{\hat{y}} \mathcal{L}(x, y, f_\theta(x)). \end{cases} \tag{2.49}$$

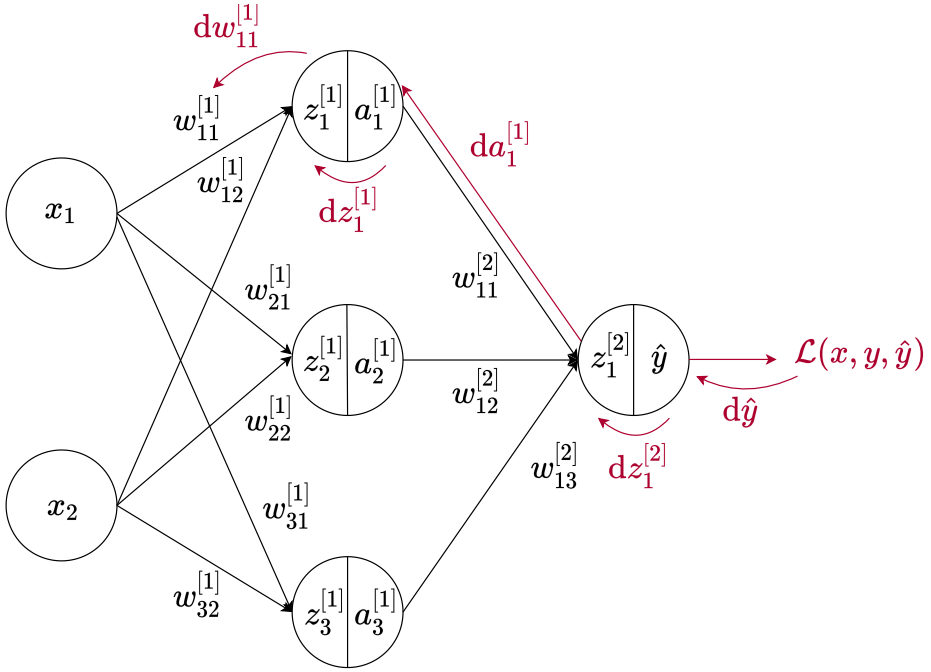


Figure 2.15: Illustration of back-propagation (in red) through the graph of an MLP with 2 layers, in order to compute the gradient of the loss w.r.t its particular weight $w_{11}^{[1]}$. For the sake of simplicity, we denote $\mathrm{d}z$ the derivative of the loss w.r.t. to any variable z: $\mathrm{d}z = \frac{\partial \mathcal{L}(x,y,\hat{y})}{\partial z}$.

where $\odot$ is the element-wise product.

The back-propagation procedure for MLPs is formally reported in algorithm 4. In addition, fig. 2.15 provides an illustration its functioning on a MLP with 2 layers.

The gradients thus computed can be interpreted as an indication of how each parameter should be changed in order to reduce the cost function. We recall that such a parameter modification is then performed by another algorithm, such as the generic ones presented in section 2.1.5 or their variants described in section 2.2.4.

2.2.3.3 General back-propagation

Although they are MLP-specific, algorithms 3 and 4 provide a good illustration of the way back-propagation is performed in ANNs. Abstractly, an ANN can be thought of as a computational graph, composed of nodes that are connected by oriented edges. To obtain the gradient of a variable z w.r.t. one of its ancestors x in the graph, one computes the gradient w.r.t. to each parent of z in the graph by multiplying the current gradient by the Jacobian of the operation that generated z. This process is performed at each node encountered while travelling backwards from z through the graph, until x is reached. For any node that may be reached two or more times while travelling backwards, the respective gradients are simply summed at the node in question.

More formally, each node in the computational graph corresponds to a tensor, that is, a variable that can have any number of dimensions (scalar, vector, matrix, etc.). An oriented edge between two tensors of the graph represents a forward operation

Algorithm 4 Back-propagation for MLPs, associated with the forward algorithm described in algorithm 3. It computes the gradients of the loss function $\nabla_\theta \mathcal{L}(x, y, f_\theta(x))$ w.r.t to the parameters θ of the network. Again, for simplicity we consider only one training sample $(x, y) \in \mathcal{S}$. From [77].

Input: Training example $(x, y) \in \mathcal{S}$
N, network depth
$\theta = \{(W^{[1]}, b^{[1]}), \ldots, (W^{[N]}, b^{[N]})\}$, network parameters (weights and biases)
$\{g^{[1]'}, \ldots, g^{[N]'}\}$, first-order derivatives of the activation functions
$\{(z^{[1]}, a^{[1]}), \ldots, (z^{[N]}, a^{[N]})\}$, preactivation and postactivation values
$\mathcal{L} = \mathcal{L}(x, y, \hat{y})$, loss from the forward step.

Compute the gradient on the output layer: $s = \nabla_{\hat{y}} \mathcal{L}$
for n=N,. . . ,1 **do**
Convert the gradient w.r.t the postactivation value into a gradient w.r.t. the preactivation value:
$s = s \odot g^{[n]'}(z^{[n]})$
Compute the gradient w.r.t. the weights and biases:
$\nabla_{W^{[n]}} \mathcal{L} = s \; a^{[n-1]^T}$
$\nabla_{b^{[n]}} \mathcal{L} = s$
Propagate the gradients w.r.t the previous layer:
$s = W^{[n]^T} s$
end for

`op.forward` which, applied to the first tensor, results in the second one. Each operation is actually also associated with its corresponding backward operation `op.backward`, which computes the Jacobian-vector product as described in eq. (2.46). In other words, each operation is responsible for knowing how to back-propagate information to its ancestors in the graph. In this way, the back-propagation algorithm itself does not need to know any differentiation rule, but only necessitate calling each operation's `op.backward` rules with the appropriate arguments. This is how the back-propagation algorithm is able to achieve great generality.

Specifically, `op.backward(inputs,x,g)` must return:

$$\sum_i \nabla_{\mathtt{x}} \mathtt{op.forward(inputs)}_i \cdot \mathtt{g}_i \tag{2.50}$$

which is simply an implementation of the chain rule. Here, `inputs` is a list of inputs supplied to the operation, `x` is the input with respect to which the gradient is to be computed, and `g` is the gradient w.r.t. the output of the forward operation.

Software libraries, such as Tensorflow or PyTorch, provide a broad range of already-implemented operations, with the forward and corresponding backwards methods. The implementation of ANNs is thus facilitated as most of the operations useful to define a neural network do not need to be explicitly written by the user.

2.2.4 Training challenges and optimisation strategies

Training a neural network ranks among the most difficult optimisation problems involved in deep learning. In this section, we first provide a brief overview of the prominent challenges encountered when training a deep neural network through the minimisation of the generally nonconvex objective function:

$$\mathcal{J}_S(\theta) = \frac{1}{S} \sum_{i=1}^{S} \mathcal{L}(x_i, y_i, f_\theta(x_i)) \tag{2.51}$$

over the provided training set $\mathcal{S} = \{(x_i, y_i) \mid i = 1, \ldots, S\}$. We then describe several of the specialised optimisation strategies developed to overcome the mentioned limitations.

2.2.4.1 Prominent challenges

Ill-conditioning of the Hessian matrix The ill-conditioning of the Hessian matrix of $\mathcal{J}_S$ is actually a general problem that affects most optimisation tasks, convex or otherwise, and the training of neural networks is no exception. An ill-conditioned Hessian matrix may cause stochastic or minibatch GD to get stuck, in the sense that even very small steps may increase the objective function.

Nonconvexity, local minima and saddle points Optimisation in general is an extremely challenging task. Traditional machine learning algorithms usually circumvent the main difficulties of general optimisation by cautiously designing the cost function

in such a way to ensure the convexity of the problem. Although convex optimisation is not devoid of complications, it offers a framework relatively straightforward to deal with. Neural networks, however, do not benefit from such an appealing setup: on the contrary, they suffer from a severe nonconvexity that makes their training all the more difficult.

A first non-desirable consequence of nonconvexity is that the optimisation algorithm may be trapped in sub-optimal local minima. In fact, nearly any deep ANN is guaranteed to have an exceedingly large number of local minima. This could be problematic for gradient-based methods if many of those local minima have a cost significantly higher than the global minimum. It is unclear, however, whether local minima are indeed a major problem when optimising networks of practical interest. Although this still remains an open question, experts tend to think that, for sufficiently large ANNs, most local minima have a low cost value, and that it is acceptable to find one of those local minima instead of the true global minimum [79, 39].

One of the major challenges entailed by nonconvexity in high-dimensional spaces is actually due to another kind of points with zero gradient, namely the saddle points. It has been shown experimentally [53] that the cost function of ANNs contains an extremely large number of high-cost saddle points. Although empirical observations suggest that in many cases, gradient-descent algorithms are capable of escaping them [79], the proliferation of such high-cost critical points remains generally problematic: as they are usually surrounded by flat regions of constant value, they may drastically slow down learning, and give the illusory impression of the existence of a local minimum.

Cliffs of the cost function Besides its flat regions, the cost function $\mathcal{J}_S$ of many neural networks is also characterised by a number of extremely steep regions, resembling cliffs, which result from the multiplication of several large weights together. Such a structure, illustrated in fig. 2.16, is problematic when using gradient-based methods since first-order derivatives do not provide a sufficient amount of information to capture the complexity of the loss surface and may cause the gradient updates to perform in an unanticipated way. Fortunately, most serious consequences of this phenomenon can be circumvented thanks to the *gradient clipping heuristic*, described in section 2.2.4.5.

Poor correspondence between local and global structures Gradient descent, and more generally all learning algorithms that prove effective for training ANNs rest upon small local moves performed in the parameter space. As aforementioned, it may be tricky to compute a local move if, for example, the current point is near a cliff or a saddle point surrounded by a flat region. Those are not, however, the only difficulties that are to be faced when optimising a neural network cost function. In fact, it is possible to overcome the local problems at a single point and still perform poorly at a global level: although the gradient descent direction is locally the best, there is no guarantee it actually points towards regions of globally low cost, especially if they are distant. The local steps performed by the optimisation algorithm can therefore lead along a path that moves downhill, but far from any desirable low-cost region, or along

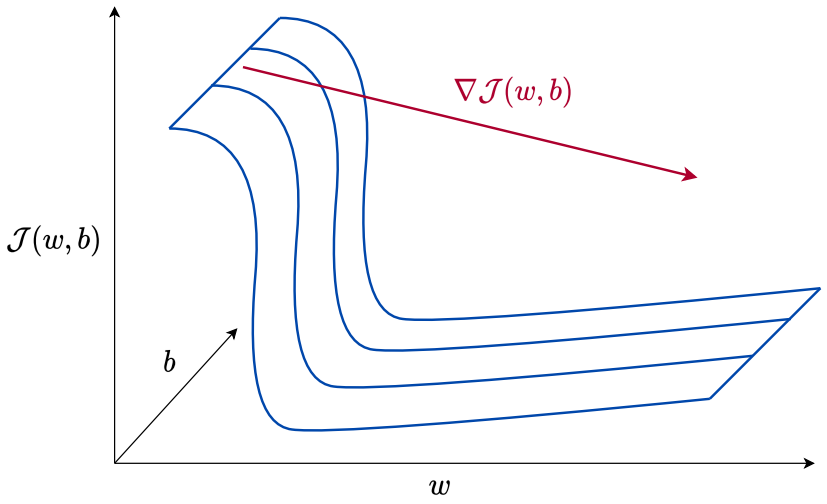


Figure 2.16: Illustrative example of a cliff on the loss surface. The gradient may undershoot if the learning rate is small, and overshoot if the learning rate is large. Illustration inspired from [3].

a trajectory that does lead to an acceptable solution, but at very high computational cost. The several problems caused by the poor correspondence between local and global structures constitute an active area of research. Most of them, however, might be avoided if there exists a region of the parameter space that is connected to an acceptable solution by a path that local descent can follow. This is the reason why many current studies are intended to find good initial points, rather than developing algorithms based on nonlocal moves [77]. Some insight of prominent initialisation methods is given in section 2.2.4.2.

Vanishing and exploding gradients Optimisation algorithms applied to very deep computational graphs may also suffer from the so-called *vanishing* or *exploding gradient* phenomena. The latter appear if the derivatives that are being back-propagated through the graph get either extremely small, or extremely large. Sometimes indeed, gradients may become vanishingly small and prevent the weights from changing their value. In the worst case, this may even completely stop the ANN from further training. This first phenomenon can arise, for example, when using activation functions with a widespread saturation domain, such as the sigmoid functions. It corresponds to flat regions of the cost function. On the contrary, when the activation functions can assume very large derivatives or when the ANN encodes the multiplication of many large weights together, the accumulated gradients may become considerable and result in exceedingly large parameter updates. This manifests for instance in the form of cliff structures, as the ones described earlier. In the overall, exploding gradients make the learning process unstable and are thus a phenomenon to be avoided.

2.2.4.2 Techniques for parameter initialisation

The process of training a neural network usually rests upon iterative algorithms, which require to specify an initial point from which to begin the iterations. The choice of this initialisation is actually determinant for the learning process, and may directly affect the convergence of the algorithm, its convergence rate as well as the quality of the solution it converges to.

The design of advanced initialisation strategies is actually not straightforward, inasmuch as ANN optimisation is not yet well understood. For example, although most initialisation techniques aim at providing desirable properties to the ANN at its initialisation state, it is not clear to what extent the network will keep those beneficial properties after some parameter updates. Furthermore, there is still little understanding of the way initialisation influences generalisation [77].

Empirical methods In practice, most popular initialisation strategies are simple and based on heuristic rules. The main property they are required to possess is to ensure that initial parameters break the symmetry between the different units of the network. This means that if two hidden units with the same activation function are connected to the same inputs, then they must be initialised with different parameters, so that they are not always updated in the same way. This may help not to lose important patterns during the forward or backwards propagations.

The simplest and cheapest way to break symmetry is probably to initialise the weights of the network randomly. Most often, they are initialised to random values drawn from a Gaussian or uniform distribution. Selecting one distribution or the other does not seem to be of great importance but no exhaustive studies have been conducted on this matter. What does have a significant effect on both the result of the optimisation process and the generalisation abilities of the network is the choice of the scale of the initial distribution. Essentially, a good trade-off is to be found in such a way that the initial weights are large enough to sufficiently break symmetry and avoid losing signal during forward or back-propagation through the linear components of each layer, without being too large in order to avert exploding gradients and activation function saturation.

Several heuristic rules have been developed. A first approach, for example, consists in initialising the weights of each neuron with d_{in} inputs and d_{out} outputs by sampling a uniform distribution:

$$W_{i,j} \sim \mathcal{U}\left(-\frac{1}{\sqrt{d_{in}}}, \frac{1}{\sqrt{d_{in}}}\right), \tag{2.52}$$

or similarly, by drawing random values from a Gaussian distribution:

$$W_{i,j} \sim \mathcal{N}\left(0, \frac{1}{\sqrt{d_{in}}}\right). \tag{2.53}$$

More sophisticated rules offer a compromise to initialise all layers with both the same activation variance and the same gradient variance. The *normalised initialisation*,

for instance, proposed in [76], relies on the following sampling:

$$W_{i,j} \sim \mathcal{U}\left(-\sqrt{\frac{6}{d_{in}+d_{out}}}, \sqrt{\frac{6}{d_{in}+d_{out}}}\right). \tag{2.54}$$

Its Gaussian equivalent, known under the name of *Xavier initialisation*, is based on a Gaussian distribution with standard deviation $\sqrt{\frac{2}{d_{in}+d_{out}}}$. In practice, some initialisation strategies are used in preference depending on the activation functions of the hidden units. Strategy (2.54), for example, is particularly powerful when using hyperbolic tangent but performs poorly with ReLU functions. In the latter case, one of the popular choices is given by the He initialisation [91].

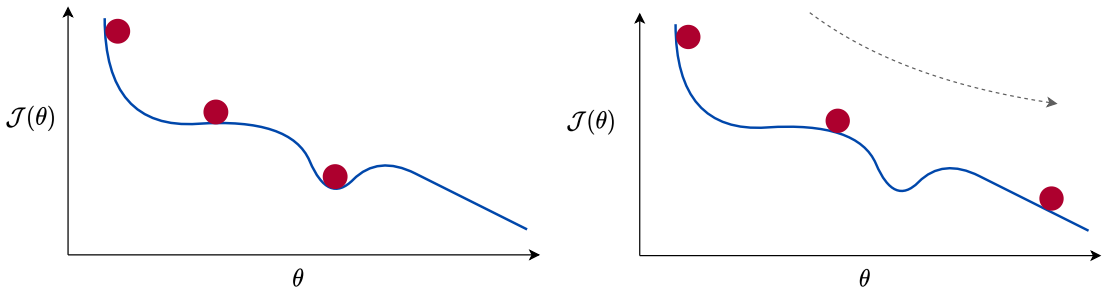
As for the biases, they are typically set to heuristically chosen constants, the selection of these constants being tightly linked to the approach adopted for setting the weights. In most of the cases, however, setting the biases to zero proves compatible with the weight initialisation scheme.

Pretraining Another possibility to find a proper parameter initialisation is referred to as *pretraining*. It mainly consists in using either supervised or unsupervised training on shallow sub-networks of the original ANN with the aim of computing the initial weights. The training of such sub-networks is then performed in a layer-wise fashion, meaning that only one layer is trained at a time. This initialisation strategy may result in a faster convergence and better generalisation abilities. A prominent example of unsupervised pretraining was proposed in [101].

2.2.4.3 Momentum-based learning

In section 2.1.5, we introduced the standard stochastic and minibatch GD algorithms, which make use of the gradient of the cost function, evaluated at the current point, to update the parameters of the learning system. Those techniques are extremely spread in the field of machine learning in general, and remain very popular in most deep learning applications. However, due to the highly nonconvex structure of the cost function in ANNs, standard SGD-like techniques may suffer from a slow convergence rate, partly as a result of their strong oscillating behaviour. Momentum-based learning techniques have thus been developed with the aim to accelerate learning, essentially by updating the parameters in an averaged direction of the last few steps, so that the oscillations in the parameter space are smoothed out.

Momentum The main idea underlying the method of momentum can be illustrated as follows: let us regard the cost function of interest as a rolling terrain, with hills and plains. In order to find the lowest point of this rolling terrain, one may place a ball at a judicious point of the terrain (parameter initialisation), and let it roll downhill (optimisation process). This is broadly what a gradient-based optimisation algorithm is intended to do. Nonetheless, this physical analogy is accurate to a limited extent. When using standard gradient descent techniques, the parameter updates only depend on the current point and on the gradient at that point. The physical ball, however,



(a) Descent without momentum. (b) Descent with momentum.

Figure 2.17: Illustration of the effect of momentum in navigating complex loss surfaces. On the left, the pure gradient descent algorithm slows down in flat regions and gets finally trapped in a local minimum. On the right, the momentum helps the gradient descent algorithm to avoid slowdowns and local basins. Figure inspired from [3].

when it rolls down, benefits in addition from some momentum, which depends on its mass and on its velocity. This momentum is particularly interesting from an optimisation point of view inasmuch as it can prevent the ball (the parameters of the network) from being trapped in a local minimum (see fig. 2.17). The momentum algorithm [166] precisely draws its inspiration from such a physical phenomenon. Formally, it introduces a variable v that plays the role of velocity, meaning that it indicates the speed and direction in which the parameters move through the parameter space. In an optimisation algorithm of the form of algorithm 1, the parameter update is thus replaced by the system:

$$\begin{cases} v^{(k+1)} = \beta v^{(k)} - \alpha g_k(\theta^{(k)}) \\ \theta^{(k+1)} = \theta^{(k)} + v^{(k+1)} \end{cases} \tag{2.55}$$

Where g_k si defined as in eq. (2.22) or eq. (2.24), α is the learning rate, and $\beta \in [0, 1)$ is a hyperparameter that determines how quickly the contributions of previous gradients exponentially decay. The larger β is w.r.t. α, the more previous gradients influence the current direction. This strategy allows to give greater preference to consistent directions over multiple steps, and thus causes the parameters to move in a direction that often points to an optimal solution while mitigating oscillations. The resulting learning process is therefore accelerated, compared to standard SGD.

Nesterov momentum A variant of the momentum algorithm was introduced in [191] under the name of Nesterov momentum. In this case, the update rule reads:

$$\begin{cases} v^{(k+1)} = \beta v^{(k)} - \alpha g_k(\theta^{(k)} + \beta v^{(k)}) \\ \theta^{(k+1)} = \theta^{(k)} + v^{(k+1)} \end{cases} \tag{2.56}$$

As it can be observed, the only difference from the standard momentum method is where the gradient is computed. The fact that the gradient is evaluated after the current velocity is applied can lead to faster convergence. In the previous analogy of

the rolling ball, such a strategy could be informally thought of as starting to apply the brakes on the gradient-descent procedure when the ball gets near the lowest point of the valley, because the lookahead will warn it about the reversal in gradient direction.

2.2.4.4 Adaptive learning rates

Due to its high-dimensional, nonconvex nature, the cost function of deep ANNs may be strongly sensitive to some directions in the parameter space, and insensitive to others. The aforementioned cliffs are an example of such a phenomenon, which is problematic inasmuch as it can bring a harmful instability to the optimisation process. The momentum methods can somewhat alleviate those issues, but another kind of algorithms were developed to provide an even more explicit and efficient way to deal with such a situation. The latter, referred to as adaptive learning rate algorithms, essentially make use of a separate learning rate for each parameter, and adapt those learning rates throughout the optimisation procedure in an automatic fashion.

An early heuristic method, which was proposed in this direction, was the delta-bar-delta method [112]. According to this approach, if the partial derivative of the cost function w.r.t. a given parameter keeps a constant sign, then the learning rate of this parameter should be increased. It is decreased otherwise. Such a strategy, however, cannot be applied to stochastic or minibatch optimisation, as it may magnify the errors of the gradient approximation. Several alternative adaptive learning rate algorithms, compatible with minibatch optimisation, have been proposed more recently, some of which are presented hereinafter.

AdaGrad In the AdaGrad algorithm [61], the learning rates corresponding to all the network parameters are scaled inversely proportional to the square root of the sum of all the previous squared gradients. Therefore, the parameters causing large partial derivatives of the cost function have a learning rate that decreases much faster than parameters resulting in smaller partial derivatives. Mathematically, the parameter update is formulated:

$$\theta^{(k+1)} = \theta^{(k)} - \frac{\alpha}{\delta I + \sqrt{\text{diag}(G_k)}} \odot g_k(\theta^{(k)}). \tag{2.57}$$

Here, we denote I the identity matrix, $\delta > 0$ a smoothing term that avoids division by zero, and G_k is the outer product matrix:

$$G_k = \sum_{t=1}^{k} g_t(\theta^{(t)}) g_t(\theta^{(t)})^T, \tag{2.58}$$

whose diagonal elements are the sum of the squared gradients up to the current iteration k. The implementation of AdaGrad is detailed in algorithm 5.

Fundamentally, this approach encourages the updates to be performed along gently sloped directions of the parameter space and can therefore benefit from some appealing properties. Its main weakness comes from its accumulation of squared gradients in

the denominator: as every added term is positive, the accumulated sum keeps growing throughout the course of training. This may cause an excessive decrease of the learning rates and prematurely slow down, or even stop, the learning process.

Algorithm 5 AdaGrad algorithm, from [77]

Input: $\mathcal{S} = \{(x_i, y_i)| \quad x_i \in X, \quad y_i \in Y, \quad i = 1, \dots, S\}$, training set
$S' < S$, size of the minibatches
$\theta^{(0)}$, initial set of parameters
α, global learning rate
$\delta \sim 10^{-7}$, small constant introduced for numerical stability

Initialise gradient accumulation variable $G_k = 0$
for $k = 0, 1, \dots$ **do**
Sample a minibatch $\mathcal{S}_k = \{(x_i, y_i) \mid i \in \mathcal{I}_k \subset \{1, \dots, S\}, \ \sharp\mathcal{I}_k = S' < S\} \subset \mathcal{S}$
Compute gradient $g_k = \frac{1}{S'} \nabla_\theta \sum_{i \in \mathcal{I}_k} \mathcal{L}(x_i, y_i, f_{\theta^{(k)}}(x_i))$
Accumulate squared gradient $G_k = G_k + g_k \odot g_k$
Compute update $\Delta\theta^{(k)} = -\frac{\alpha}{\delta I + \sqrt{G_k}} \odot g_k$ (operations applied element-wise)
Apply update $\theta^{(k+1)} = \theta^{(k)} + \Delta\theta^{(k)}$
end for

RMSProp algorithm The RMSProp algorithm [100] overcomes the limitations of AdaGrad by converting the gradient accumulation into an exponentially weighted moving average. In this manner, the contributions that are old enough to be considered as irrelevant are discarded, and the algorithm can converge rapidly when it reaches a convex basin. The standard RMSProp method is described in algorithm 6. It can also be coupled with Nesterov momentum. In practice, the RMSProp algorithm has proven efficient in the context of nonconvex functions, and is nowadays one of the most popular optimisation methods used by deep learning practitioners, [77].

Adam Adam algorithm [121], whose name derives from the terms 'adaptive moments', can be regarded as a variant of RMSProp with momentum. It not only makes use of an exponential moving average $v^{(k)}$ of the squared gradients, as does RMSProp, but it also involves an exponential moving average $m^{(k)}$ of the gradients themselves, which can be seen as a momentum term. Formally,

$$\begin{cases} m^{(k+1)} = \rho_1 m^{(k)} + (1 - \rho_1) g_k(\theta^{(k)}) \\ v^{(k+1)} = \rho_2 v^{(k)} + (1 - \rho_2) g_k(\theta^{(k)}) \odot g_k(\theta^{(k)}), \qquad k = 0, \dots, K-1. \\ m^{(0)} = 0, \ v^{(0)} = 0. \end{cases} \tag{2.59}$$

where the hyperparemeters $\rho_1, \rho_2 \in [0, 1)$ determine the exponential decay rates, and m_k and v_k are respectively the first-order moment (the mean) and the second-order raw moment (the uncentered variance) of the gradients. The authors of [121], however,

Algorithm 6 RMSProp algorithm, from [77]

Input: $\mathcal{S} = \{(x_i, y_i)| \quad x_i \in X, \quad y_i \in Y, \quad i = 1, \dots, S\}$, training set
$S' < S$, size of the minibatches
$\theta^{(0)}$, initial set of parameters
α, global learning rate (suggested default: 0.001)
ρ, decay rate (suggested default: 0.9)
$\delta \sim 10^{-6}$, small constant introduced for numerical stability

Initialise gradient accumulation variable $G_k = 0$
for $k = 0, 1, \dots$ **do**
Sample a minibatch $\mathcal{S}_k = \{(x_i, y_i) \mid i \in \mathcal{I}_k \subset \{1, \dots, S\}, \ \sharp\mathcal{I}_k = S' < S\} \subset \mathcal{S}$
Compute gradient $g_k = \frac{1}{S'} \nabla_\theta \sum_{i \in \mathcal{I}_k} \mathcal{L}(x_i, y_i, f_{\theta^{(k)}}(x_i))$
Accumulate squared gradient $G_k = \rho G_k + (1 - \rho) g_k \odot g_k$
Compute update $\Delta\theta^{(k)} = -\frac{\alpha}{\delta I + \sqrt{G_k}} \odot g_k$ (operations applied element-wise)
Apply update $\theta^{(k+1)} = \theta^{(k)} + \Delta\theta^{(k)}$
end for

observed that the moving estimates defined in eq. (2.59), inasmuch as they were initialised with zeros, led to moment estimates that are biased towards zero, in particular over the first iterations of the training. As a consequence, they suggest to compute bias-corrected estimates instead:

$$\begin{cases} \hat{m}^{(k)} = \frac{m^{(k)}}{1 - \rho_1^k} \\ \hat{v}^{(k)} = \frac{v^{(k)}}{1 - \rho_2^k}, \qquad k = 1, \dots, K. \end{cases} \tag{2.60}$$

Based on these corrected moving averages, the parameters are updated according to the rule:

$$\theta^{(k+1)} = \theta^{(k)} - \alpha \frac{\hat{m}^{(k+1)}}{\delta + \sqrt{\hat{v}^{(k+1)}}}. \tag{2.61}$$

Adam offers the advantage of being generally fairly robust to the choice of hyperparameters, and is part of the optimisation algorithms of choice for the training of neural networks. Its implementation details are given in algorithm 7.

2.2.4.5 Gradient clipping

Gradient clipping is another technique employed in cases where partial derivatives along different directions of the parameter space have exceedingly different magnitudes. In some sense, this strategy is based on a similar principle to that of adaptive learning rate algorithms, in that it attempts to make the different components of the partial derivatives more even. Gradient clipping, however, involves only the current values of the gradients and does not keep track of their historical values. This approach stems from the fact that the gradient does not bring any information about the steplength to adopt, it only indicates the optimal direction within an infinitesimal region. When the optimisation algorithm is at a point near a cliff of the cost function, for example, the

Algorithm 7 Adam algorithm, from [121]

Input: $\mathcal{S} = \{(x_i, y_i)| \quad x_i \in X, \quad y_i \in Y, \quad i = 1, \dots, S\}$, training set
$S' < S$, size of the minibatches
$\theta^{(0)}$, initial set of parameters
α, global learning rate (suggested default: 0.001)
$\rho_1, \rho_2,$, exponential decay rates (suggested defaults: 0.9 and 0.999 resp.)
$\delta \sim 10^{-8}$, small constant introduced for numerical stability

Initialise first and second-order moment vectors $m^{(0)} = 0,\ v^{(0)} = 0$
for $k = 1, 2, \dots$ **do**
Sample a minibatch $\mathcal{S}_k = \{(x_i, y_i) \mid i \in \mathcal{I}_k \subset \{1, \dots, S\},\ \sharp\mathcal{I}_k = S' < S\} \subset \mathcal{S}$
Compute gradient $g_k = \frac{1}{S'} \nabla_\theta \sum_{i \in \mathcal{I}_k} \mathcal{L}(x_i, y_i, f_{\theta^{(k-1)}}(x_i))$
Update biased moment estimates
$m^{(k)} = \rho_1 m^{(k-1)} + (1 - \rho_1) g_k$
$v^{(k)} = \rho_2 v^{(k-1)} + (1 - \rho_2) g_k \odot g_k$
Compute bias-correct moments
$\hat{m}^{(k)} = \frac{m^{(k)}}{1-\rho_1^k}$
$\hat{v}^{(k)} = \frac{v^{(k)}}{1-\rho_2^k}$
Compute update $\Delta\theta^{(k)} = -\alpha \frac{\hat{m}^{(k)}}{\delta + \sqrt{\hat{v}^{(k)}}}$ (operations applied element-wise)
Apply update $\theta^{(k)} = \theta^{(k-1)} + \Delta\theta^{(k)}$
end for

computed gradient may be extremely large in at least one direction, and a traditional parameter update may totally overshoot. In those cases, gradient clipping intervenes and scales down the computed gradient, making it more likely to have a reasonable behaviour. The most common forms of gradient clipping are *value-based clipping* and *norm-based clipping*. This technique allows *inter alia* to avoid exploding gradients. A detailed exploration is provided in [161].

2.2.4.6 Batch normalisation

Several further optimisation techniques have been devised in order to improve the training of deep learning models. Here, we only mention *batch normalisation*, since it is highly popular in the field of ANNs, but other methods can be found in [77].

Batch normalisation [110], is actually not an optimisation algorithm, but rather a subroutine of adaptive reparametrisation that can be incorporated into many different algorithms. It may prevent vanishing or exploding gradients but above all, it addresses the problem of *internal covariate shift*. The latter emerges from the fact that all the parameters are updated simultaneously at each iteration of the training algorithm, while the gradients employed to do so are computed under the assumption that the parameters in the neighbouring layers do not change. This may cause any parameter update to bring about unexpected results, and lead to a slower convergence. Batch normalisation offers a solution to this problem by introducing additional normalisation layers between hidden layers. Those additional layers are intended to standardise the mean and variance of each unit and thus results in a more stable learning. It is furthermore possible to incorporate two trainable parameters into each unit of the normalisation layers, so they can regulate the precise level of normalisation.

2.3 Convolutional Neural Networks

One of the most popular and efficacious kinds of deep learning models is undoubtedly the class of convolutional neural networks. The latter are designed to process grid-structured data, such as images or time-series data. *Qua* ANNs, they can be trained through the gradient-descent techniques, the back-propagation algorithm and the optimisation strategies presented in the previous sections. Actually, the main difference between CNNs and MLPs rests upon the fact that CNNs use convolution instead of general matrix multiplication in at least one of their layers. Inasmuch as a convolution can be thought of as a multiplication by a circulant matrix, one can perceive that, on a theoretical level, a CNN is equivalent to a fully connected neural network in which several units would be constrained to be equal to other units. In practice however, the convolution-based architecture of CNNs provides them extremely appealing properties and results in a computationally highly efficient structure that has proven successful in a number of applications. This section describes the functioning and main characteristics of such models, which are a keystone of the research work presented in this thesis.

2.3.1 Historical perspective and neuroscientific inspiration

CNNs have played a substantial role in the history of deep learning. They were actually the first deep models to perform well, long before recent technological advancements led to enhanced performance in other types of architectures. They were also part of the first neural networks to prove successful in a number of important commercial applications and have thus largely contributed to the current intensity of commercial interest in deep learning. The prominence of CNNs, and their extremely broad range of application, show how biologically inspired artificial intelligence models can sometimes lead to ground-breaking results.

The history of convolutional networks indeed started with neuroscientific experiments, conducted by the neuropsychologists David Hubel and Torsten Wiesel on a cat's visual cortex [107]. Their findings helped to characterise many aspects of brain function and had a considerable impact on contemporary deep learning models. In particular, they discovered that the visual cortex is constituted by small regions of cells that are sensitive to specific patterns in the visual field, and totally insensitive to other patterns. In other words, the activation of the cells in the visual cortex depends on the shape and orientation of the objects in the visual field. For example, horizontal edges cause some neuronal cells to be excited, while vertical edges cause different neurons to be activated. Furthermore, the observation that those cells are connected via a layered-architecture suggested that these layers allow the brain to process portions of the image at different levels of abstraction. This phenomenon, also known as *hierarchical feature extraction*, appeared particularly interesting and indeed inspired the design of CNNs. More information about this property is given in section 2.3.3.

Historically, the earliest neural model based upon the above-mentioned biological observations was the *neocognitron* [72]. Although it differs from the current CNNs, *inter alia* in that it does not take advantage of weight sharing, it nonetheless paved the way for the development of the highly efficient, modern convolutional architectures. Amongst the latter, *LeNet-5* [131] was one of the first to draw commercial interest, and soon became a widely spread method to identify hand-written digits on bank checks. Since then, other convolutional architectures have been devised and the number of tasks for which CNNs have proven useful has continued to grow. One of the factors that played a significant role in increasing the prominence of CNNs is the annual *ImageNet* contest, which has been consistently won by CNNs since 2012. As of today, some convolutional networks are even able to exceed human-level performance, an achievement that would have appeared impossible to most experts in computer vision few decades ago.

2.3.2 Structural description

Convolutional networks are designed to process multi-dimensional grid-structured inputs that have strong spatial dependencies in local regions of the grid. As such, they are particularly well-suited for imaging problems, since images do exhibit spatial dependencies (adjacent pixels often have similar colour values). In the remainder of this

chapter, we will mostly illustrate the concepts underlying CNNs through imaging examples. Convolutional networks can, however, be applied to other kinds of data, and are indeed also used in video recognition, natural language processing and financial time series, for instance.

The functioning of a CNN is very similar to that of a traditional feed-forward ANN, except that the operations in its layers are spatially organised with sparse connections between layers. This architecture allows a drastic reduction in terms of trainable parameters, and thus contributes to the appealing computational efficiency of CNNs.

Typically, a CNN involves three types of operations, which are interleaved in order to increase the expressive power of the network: convolution, ReLU and pooling. In addition, it is not uncommon to append a final set of fully connected layers, according to the task under consideration. Below, we introduce those layers and the related operations.

2.3.2.1 Convolutional layer and ReLU

As its name suggests, a convolutional layer fundamentally rests upon convolution computations. Informally, a convolution is an operation taking two functions as arguments, and outputting a third function that expresses how the shape of one is modified by the other. Such an operation is usually denoted by an asterisk $*$ between the two functions it is applied to. Its mathematical 1-D definition is reported below.

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

$$(f * h)(t) = \int_{-\infty}^{\infty} f(t - t')h(t')dt' = \int_{-\infty}^{\infty} f(t')h(t - t')dt'. \tag{2.62}$$

A way to interpret $f*h$ is to regard it as a weighted mean of the values of f, in which the weights are defined by the values of h. In other words, computing $f * h$ for a given t amounts to shifting f by t, multiplying it by h, and integrating the product over the real line. The resulting function is thus a transformed representation of f, whose shape depends on the features of h. Although there are some differences between the 1-D mathematical definition in eq. (2.62) and the practical implementation of convolutions in CNNs, this interpretation gives a first good insight of the principles underlying a convolutional layer.

In practice, the data to which a CNN is applied is digital, meaning that the functions of interest are not continuous, but discrete. The mathematical definition of 1-D discrete convolution is given by:

$$(f * h)(t) = \sum_{t'=-\infty}^{\infty} f_{t-t'}h_{t'} = \sum_{t'=-\infty}^{\infty} f_{t'}h_{t-t'} \tag{2.63}$$

As it can be observed, the mathematical convolution is commutative. This property arises because one of the arguments is 'flipped', meaning that as the index of the sum increases, the index of one of the arguments increases as well, while the index of the flipped one decreases. Although the commutative property is helpful for writing proofs,

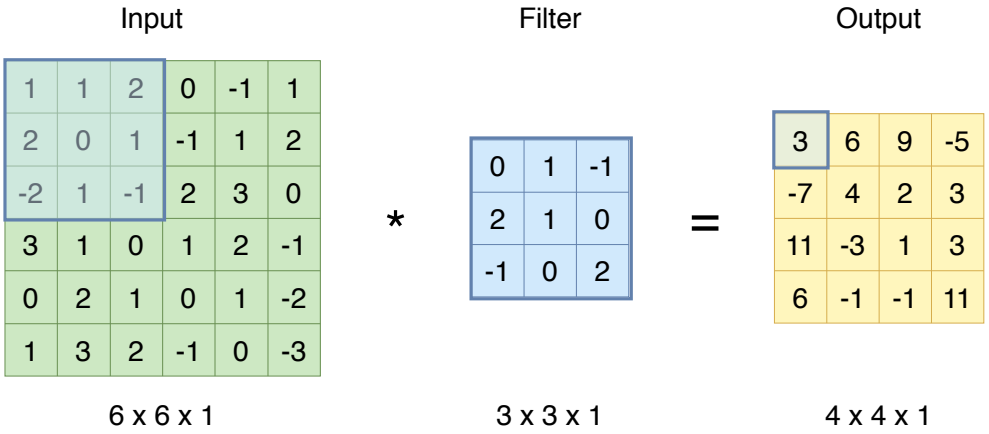


Figure 2.18: Illustration of a 2-D convolution between a $6 \times 6 \times 1$-input and a $3 \times 3 \times 1$-filter. Each coefficient of the $4 \times 4 \times 1$-output is equal to the dot product between the entries of the filter and the corresponding entries of the input. For instance, the top-left coefficient of the output is equal to: $1 \times 0 + 1 \times 1 + 2 \times (-1) + 2 \times 2 + 0 \times 1 + 1 \times 0 - 2 \times (-1) + 1 \times 0 - 1 \times 2 = 3$. In this example, no padding is used and the stride is equal to 1.

it is not usually a necessary property for the implementation of neural networks. This is the reason why most ANN libraries actually implement a related function, which is essentially a convolution without flipping any of the arguments. Thus, in 1-dimension, the term *convolution* in machine learning actually corresponds to the operation:

$$(f * h)(t) = \sum_{t'=-\infty}^{\infty} f_{t'} h_{t+t'} \tag{2.64}$$

In convolutional network terminology, the first argument is often referred to as the *input* and the second one as the *filter*, or *kernel*. As for the output of the convolution, it is usually called *feature map*. As all the elements of the input and kernel are variables to be stored, it is usually assumed that these functions are zero everywhere but in the finite set of points whose values are stored. This means that in practice, the infinite summation is actually implemented as a summation over a finite number of elements.

Most frequently, convolutions in CNNs are employed over more than one axis at the time and involve 3-D variables[1], called *tensors*, which have a *width*, a *height* and a *depth*. Typically, the inputs are 3-D arrays of data (such as multi-channel images), and the kernels are 3-D arrays of parameters that are to be trained by a learning algorithm. Computing a convolution comes down to placing the kernel at each possible position in the image (or hidden layer) so that the filter fully overlaps with the image, and performing a dot product between the parameters in the filter, and the matching grid in the input volume. The number of alignments between the filter and the image (or

[1]In practice, software implementations process several inputs in parallel. This means that the variables actually have four dimensions, where the fourth one indexes the different examples of the *batch*. For simplicity, we omit the batch axis in our description.

hidden layer) thus defines the spatial height and width of the next hidden layer. As for its depth, it is defined by the number of filters applied to the current hidden layer.

More formally, a convolution in a CNN is defined as follows. We denote $W^{[p,q]} = \left(w_{ijk}^{[p,q]}\right)$ the tensor of the trainable parameters in the p^{th} filter of the q^{th} layer, where the indices i, j, and k indicate the positions along the height, width and depth of the filter. We assume that this filter is of dimensions $F_q \times F_q \times d_q$. Furthermore, we denote $Z^{[q]} = \left(z_{ijk}^{[q]}\right)$ the tensor containing the feature maps in the q^{th} layer, and we assume it is of size $H_q \times L_q \times d_q$. Then, the convolution between the feature maps and the p^{th} filter at the q^{th} layer outputs the p^{th} feature map of the $(q+1)^{\text{th}}$ layer according to the formula:

$$z_{ijp}^{[q+1]} = \sum_{r=1}^{F_q} \sum_{s=1}^{F_q} \sum_{k=1}^{d_q} w_{rsk}^{[p,q]} z_{i+r-1,j+s-1,k}^{[q]} \tag{2.65}$$

for all $i \in \{1, \ldots, H_q - F_q + 1\}$, $j \in \{1, \ldots, L_q - F_q + 1\}$ and $p \in \{1, \ldots, d_{q+1}\}$. Although equation (2.65) might appear somewhat complex from a notational point of view, it is nothing but the expression of the dot-product between the feature maps and the filter, which is repeated over all the spatial positions (i, j) of the current hidden layer. The set of convolutions using one given filter thus produces a new 2-D feature map. By employing several filters and stacking the results along the depth dimension, one obtains the 3-D volume of all the feature maps of the next hidden layer. This procedure is illustrated in the 2-D case in fig. 2.18 and in the 3-D case in fig. 2.19. It is not uncommon, furthermore, to add a bias term to each 2-D feature map computed. Most of the time, the filters are chosen to be spatially small. Their number, on the contrary, is usually very large (from few hundreds to several thousands). The larger the number of filters, the larger the number of trainable parameters, and thus the higher the capacity of the network.

The convolution, when performed as explained above, reduces the size of the $(q+1)^{\text{th}}$ layer in comparison to the size of the q^{th} layer. This kind of size reduction is not desirable in that it causes a loss of information along the borders of the image or internal feature maps and, in the case of deep CNNs, may result in extremely small hidden layers. Such a phenomenon can be bypassed by using *padding*. This technique consists in adding external rows and columns of zeros to the input of the convolution, in order to make it wider. It allows the filter to 'stick out' from the borders of the layer during the convolution operation: the dot product is then computed only on the portion of the layer where the values are defined, since the padded portions do not contribute to the final dot product as their values are 0. One of the advantages of padding is that the size of the output of the convolutions can be easily tuned, in function of the number of zeros added all around the borders of the input (cf eq. (2.66)). There exist different types of padding implemented in the commonly used libraries. One is the extreme case in which no padding is used and the convolutional filter visits only positions where it is entirely contained within the image: this is the *valid-padding*. Another type, amongst the most frequent, is called *same-padding*: in that case, just enough zeros are added so that the size of the output is equal to the size of the input. An example of same-padding is given in fig. 4.7 of chapter 4.

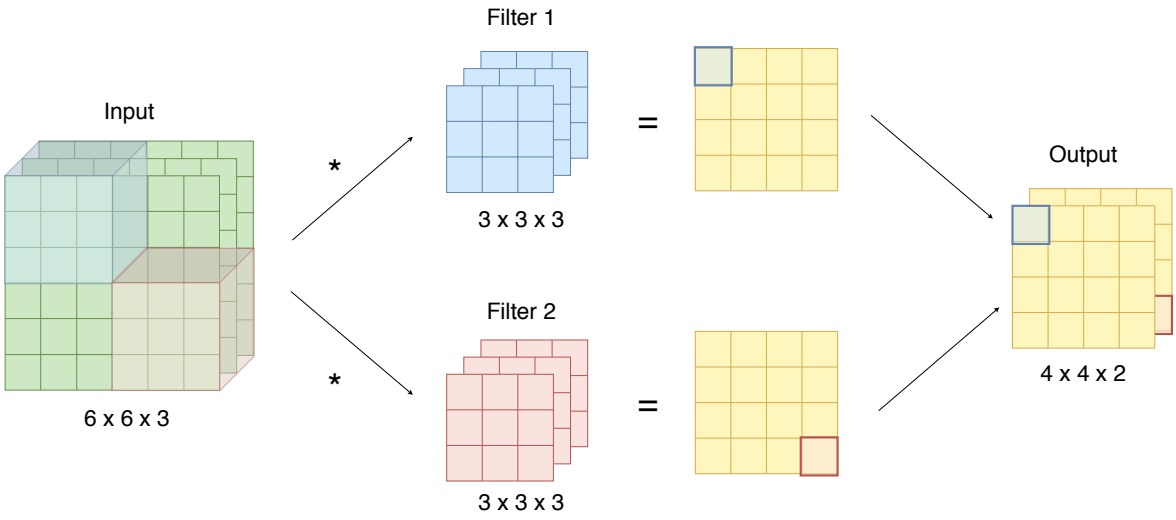


Figure 2.19: Illustration of 3-D convolution between a $6 \times 6 \times 3$-input and two $3 \times 3 \times 3$-filters. Each filter, when convolved with the input, produces a 2-D feature map. The two resulting feature maps are then reassembled and the final output is a $4 \times 4 \times 2$-tensor. In this example, no padding is used and the stride is equal to 1.

There are other ways to modify the size of the output of a convolution. It is possible, for example, to tune the so-called *stride*. The stride defines the amount of pixels by which the filter is shifted after each operation. So far, we have implicitly assumed that the slide is equal to 1, but it is possible, and not uncommon, to use a larger stride. In general, larger strides can be useful, for example, to reduce overfitting if the spatial resolution is needlessly high. As strides greater than 1 result in a reduction of the spatial dimension of the hidden layers, they may also prove helpful in memory-constrained setups.

If we denote S_q the stride and P_q the padding, at the q^{th} layer, the dimensions of the output of a convolution are equal to:

$$H_{q+1} = \frac{H_q - F_q + 2P_q}{S_q} + 1, \qquad L_{q+1} = \frac{L_q - F_q + 2P_q}{S_q} + 1. \tag{2.66}$$

The use of the convolutional operator instead of a standard matrix multiplication as it is the case in fully connected ANNs, gives CNNs appealing properties. In particular, it allows the network to use each parameter for more than one function in the model: this is called *parameter sharing*. Such a characteristic not only entails a drastic reduction in the number of trainable parameters, but it also gives CNNs the property of *equivariance* to linear translations. In a network, equivariance essentially means that if the input image is translated, the activations of the network in each layer will translate the same way. This is advantageous when we know that some function of a small number of neighbouring pixels is to be applied to multiple input locations. For instance, in image processing, if one filter of the CNN learns how to detect horizontal edges, it is practical to share this filter across the entire image, which very likely will contain many horizontal edges in different locations. In the overall, the properties of

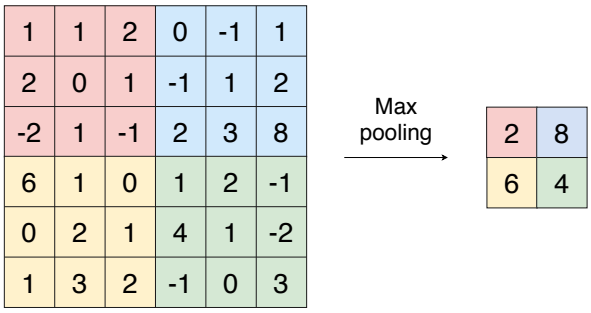


Figure 2.20: Example of max-pooling operation on a single depth slice with stride $S = 3$ and $F = 3$. The output is composed of the maximum pixel value of each spatial region of size 3×3 in the input.

the convolution operation significantly reduce the memory requirements of CNNs in comparison to fully connected ANNs, and impart them a higher statistical efficiency.

The output of a convolutional layer is essentially a linear combination of the input values. Therefore, it is important to apply some nonlinear function to those values before transmitting them to the next layer, just as in the framework of MLPs. This function is applied to all the feature maps computed, and does not change their dimensions as it is a simple one-to-one mapping of activation values. Although in the early years, saturating activation functions like sigmoid ones were used, it has been shown [128] that ReLU has considerable advantages over the latter, both in terms of speed and accuracy, and it has thus become the default option in CNNs.

2.3.2.2 Pooling layer

Most commonly in CNN architectures, convolutional layers are interleaved with pooling layers. The latter differ from the former for two main reasons. First, pooling layers do not contain any trainable parameters. Secondly, unlike convolutional operations, which simultaneously use all the d_q feature maps in combination with a filter to generate a single value, the pooling function is independently applied to each feature map to generate another feature map. Therefore, pooling is applied to spatial regions of each hidden layer, and produces another layer with the same depth as the current one.

Essentially, the function of pooling layers is to reduce the spatial size of the feature maps, without loosing too much information. In practice, the pooling operation outputs a summary statistic of each small spatial region it is applied to. For example, the *max pooling* function [235] returns the maximum pixel value within a rectangular neighbourhood, as illustrated in fig. 2.20. Other popular pooling operations encompass the average of a rectangular neighbourhood (*average pooling*), the ℓ_2-norm, or a weighted average based on the distance from the central pixel. Some guidance, based on theoretical considerations, about which kind of pooling should be employed in various situations, is given in [26].

Either way, a pooling layer performs a downsampling operation along the spatial axis, while the depth dimension remains unchanged. The pooling operation is described by two hyperparameters: the stride S_q, and the side F_q of the square neighbourhood in which to apply the downsampling operation (or the length and width if the neighbourhood of interest is assumed to be rectangular). The output dimensions of a pooling layer, applied to a $H_q \times L_q$ feature map then read:

$$H_{q+1} = \frac{H_q - F_q}{S_q} + 1, \qquad L_{q+1} = \frac{L_q - F_q}{S_q} + 1. \tag{2.67}$$

Since it reduces the spatial sizes of the feature maps, the pooling operation allows a reduction of the amount of parameters and computations needed in the next layers. It thus enhances the computational efficiency of the network while reducing its memory requirements for storing the parameters. Furthermore, pooling make it possible for a same network to handle inputs of varying size. In the case of image classification, for instance, the input of the classification layer must have a fixed size, independently of the size of the input image. It is then possible to define the final pooling layer in such a way that it outputs four set of summary statistics, one for each quadrant of the feature map, regardless of the image size.

Another important characteristic of pooling is that it helps to make the representation *invariant* to translation, since slightly shifting the input will not change the values of most of the pooled outputs. This property proves helpful if, given the task under consideration, it appears more important to know whether some feature is present in the image rather than identifying its exact location. For instance, when determining if an image contains a bird, there is no need to know the location of the beak and wings with pixel-perfect accuracy, it is just sufficient to detect their presence in the image.

Finally, pooling increases the size of the receptive field across the layers and thus allows the network to capture larger and larger regions of the image. It therefore highly contributes to the hierarchical feature extraction abilities of CNNs, as explained in section 2.3.3.

2.3.2.3 Fully connected layers

It it possible to append several layers of fully connected neurons to the final layers of a CNN. They function in the exact same way as a traditional feed-forward network. As they are densely connected, those layers may actually contain the vast majority of the CNN parameters. By way of example, if each of two fully connected layers has 4096 hidden units (which corresponds to no more than the flattened pixels of a 64×64 feature map), then the connections between them have more than 16 million weights. Whether to add fully connected layers to a CNN as well as their number and widths is sensitive to the application at hand. When the output of the CNN must be another image, for example, no such layers are employed.

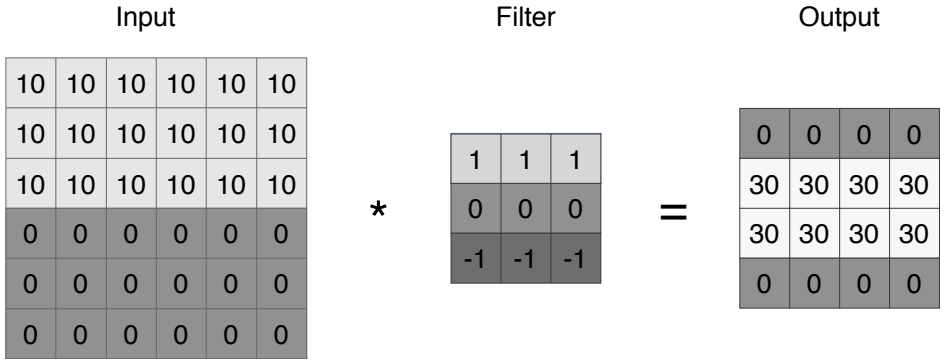


Figure 2.21: Example of horizontal edge detection by means of convolution. We consider a 6×6 input, whose three upper rows have high intensity, whereas the three lower rows have low intensity. Such an input can be thought of as a simple image, composed of two regions of different colours delimited by a horizontal edge. By applying a wisely designed convolutional filter, such as the one presented in the image, it is possible to detect the horizontal edge: the output of the convolution has indeed two non-zero value rows, whose location corresponds to that of the discontinuity in the image.

2.3.3 Hierarchical feature extraction

It is interesting to observe the activations of trained filters when applied to real-world images. Generally, the activations in the early layers are low-level features, such as edges, while those in later layers are more and more complex features. In fact, each layer puts together the features learnt in earlier layers in a semantically coherent way in order to learn increasingly complex and interpretable visual features.

In order to get an intuition about this process, let us first give an illustration about how early layers are able to detect edges, and then how this kind of features can be exploited by later layers. Fig. 2.21 shows how a carefully designed filter can detect an horizontal edge on a grey-scale image with one channel. As it can be seen, the resulting feature has high activation at the positions where a horizontal edge is detected. If that same filter were applied to an image containing perfectly vertical edges, those would give zero activation, while a tilted edge would result in intermediate activations. Thus, the application of horizontal and vertical edge detector filters across an entire image generates very different activation feature maps, locating all the horizontal and vertical edges respectively. The fact that small portions of an image activate hidden layers in a different way depending on the features they contain is reminiscent of the biological model of Hubel and Wiesel in which different shapes in the visual field activate different neurons.

The next layer can then make use of those primitive features to detect more complex features. For instance, fig. 2.22 illustrates how the second layer can assemble the edges detected in the first one, in order to identify a rectangle. A later layer may then combine the rectangles of the second layer to detect a brick wall, and so on.

This hierarchical feature extraction ability is *inter alia* made possible by the fact that both the convolution operation and the pooling function increase the receptive

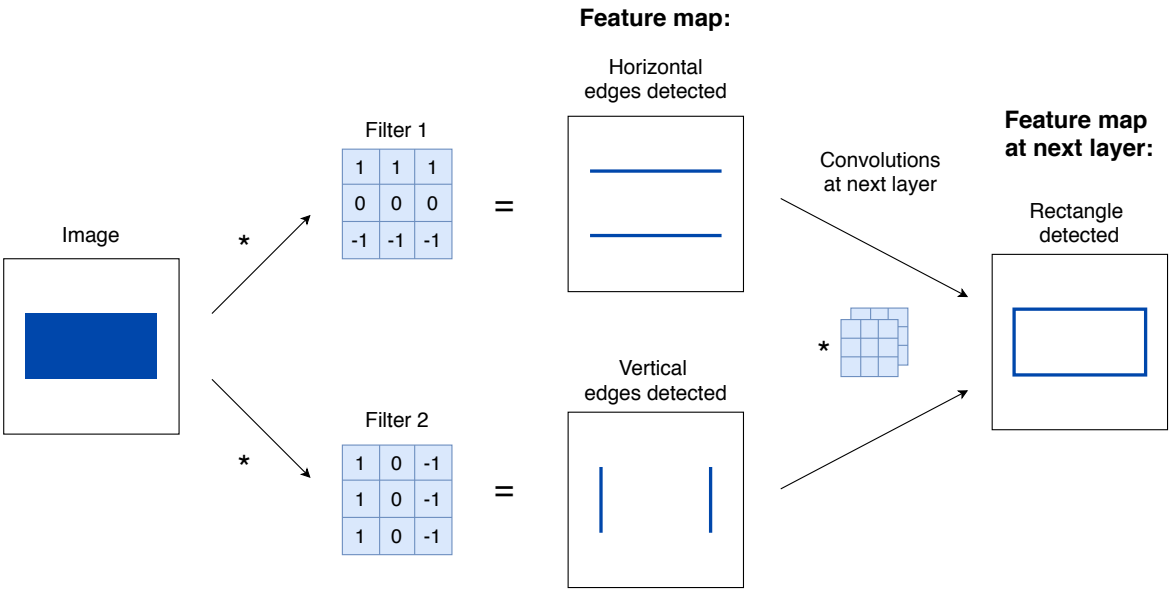


Figure 2.22: Illustration of how filters can detect edges and combine them to create a rectangle. Figure inspired from [3].

field of the features across the network. In other words, each feature in a layer q captures a larger spatial region in the layer $(q-1)$. For example, if we apply a simple 3×3 filter convolution to three successive layers, the activations thereof will respectively capture pixel regions of size 3×3, 5×5 and 7×7 in the original input image. The same phenomenon occurs within a pooling layer. In this way, later layers can capture the characteristics of the image over larger and larger spatial regions, and a sufficiently deep CNN can thus combine enough complex patterns to process a whole real-world image containing very sophisticated shapes.

The values of the filter parameters are determined through a learning process that allies back-propagation and gradient-descent optimisation as explained in 2.2. They may heavily depend on the task at hand. For instance, if one is interested in training a CNN to classify images as containing a car or not, the first layers may learn how to put together arcs to detect a circle, and the successive ones might combine circles with other shape to create a car wheel.

Like in the case of MLPs, the initialisation of the convolutional filters can significantly impact the quality of the results. A first strategy consists in initialising them with random values. Several works empirically demonstrated how this kind of initialisation work surprisingly well in CNNs [114, 164, 47]. In particular, [183] shows that convolutional layers interleaved with pooling ones naturally become frequency selective and translation invariant when assigned random weights. A second initialisation strategy is to design the filters by hand. Those two first approaches are investigated in the work presented in this thesis. Finally, one can initialise the filters using a greedy layer-wise pretraining or learn them through an unsupervised process.

2.3.4 Conclusion

In this section, we have described the fundamental principles underlying convolutional neural networks. Most popular CNN architectures make use of the aforementioned operations and strategies, and mainly differ in the arrangement of those operations in the network. In fact, the main changes from the seminal *LeNet-5* architecture [131] to modern models come essentially from the use of ReLU activation, the explosion of depth, and the training efficiency made possible by modern computers and optimisation enhancements. Nowadays, CNNs are increasingly deep and are built according to a range of computational, architectural and hardware tricks that make their training possible and efficient. By way of example, prominent CNN models for image classification include *AlexNet* [128] and *VGG* [188], which are composed of several convolutional layers interleaved with max-pooling layers, and two additional fully connected layers. More advanced architectures have also provided remarkable results, such as the *GoogleNet* [194], which has proposed the concept of *inception architecture*, that is, a network within a network. Finally, noteworthy CNN design possibilities encompass the use of *skip connections*, which allow feature maps to be copied from one layer to a next one and thus introduce an *iterative view* of feature engineering (as opposed to *hierarchical view*). This approach was *inter alia* investigated in *ResNet* [92].

The notions discussed in this chapter are fundamental for a clear understanding of the research work presented in the two following chapters. The proposed model, ΨDONet, is a CNN; it makes use of the aforementioned convolutional and pooling operations and is furthermore trained through the back-propagation and optimisation algorithms described in this chapter. However, its global architecture differs from many popular CNN architectures in that it does not make use of an arbitrary number of filters and layers, but instead is designed according to the unfolding of a variational method (ISTA). The advantages of such an approach are multiple: first, this gives ΨDONet more explainability and reliability, inasmuch as its structure can be justified from a mathematical point of view. Secondly, this means that its convolutional filters can be initialised to values computed beforehand, based on the physical understanding of the operators involved. An efficient training of such filters can then lead to an increasingly good quality of the reconstructions.

3 From ISTA to ΨDONet: theoretical results

The deep learning techniques presented in chapter 2 offer an extremely powerful framework which, *inter alia*, has already provided remarkable results for a number of inverse problems. In particular, CNNs have become a prominent tool in the field of computer vision and imaging [232, 222, 59, 115]. However, DNNs and CNNs present the drawback of a questionable explainability and reliability when used as mere black-boxes. In medical applications in particular, where it is crucial for the model to be both explainable and dependable, algorithms that rest upon the exclusive use of deep neural networks may not be considered as a satisfactory option.

In this doctoral thesis, we present an approach that takes advantage of the performance of the deep learning framework, while benefiting from the trustworthiness of a more traditional, variational method. Our model, ΨDONet, can thus incorporate both the physical understanding of the problem under consideration and a *prior* leveraged for the regularisation of the solution, while making use of a learning process to extract meaningful information from a set of training examples.

ΨDONet is a CNN inspired from the well-known ISTA [52], and it is designed to learn convolutional ΨDOs or FIOs in the broad context of linear inverse problems. In particular, it can be applied to the case of LA-CT, whose essential operator belongs indeed to the class of FIOs. This application, which is the main focus of this thesis, will be extensively investigated in chapter 4.

In the current chapter, we aim at presenting the theoretical principles underlying the proposed algorithm ΨDONet. In order to provide a general description of the abilities and characteristics of our approach, we will present the following results in the broad context of linear inverse problems, where we call A the general operator defining the inverse problem. It is clear, however, that all the convergence results and observations noted in this chapter apply to the particular case of LA-CT, where $A = R_\Gamma$.

First, section 3.1 is dedicated to the formulation of the general inverse problem and the theoretical background of sparsity promoting regularisation. In section 3.2, we detail the key idea subtending our approach, namely the convolutional interpretation of ISTA using the wavelet transform. Finally, in section 3.3, we present the general

design of the proposed CNN, ΨDONet, and the main theoretical results thereof.

3.1 Inverse problem and sparsity promoting regularisation

3.1.1 Preamble: wavelets in 2D

Before entering the details of the problem formulation and its sparsity promoting regularisation, we provide a quick reminder of important characteristics of the wavelet basis in $\mathbb{R}^2$. The notions presented here are useful for the understanding of the results provided in this chapter. Further details about the functioning of wavelets are provided in Appendix B.

A wavelet basis in $\mathbb{R}^2$ can be fully described by means of two real functions ϕ and φ, respectively defined as the *mother wavelet* and the *scaling function*, whose support is in $[0,1]$. The different elements ψ_I of the basis are then defined as the multiplication of those two functions, applied to a scaled, translated input. Formally, we identify any ψ_I by its scale j, its translation $k \in \mathbb{N}_0^2$ and its type $(t) \in \{(v),(h),(d),(f)\}$, standing for the vertical, horizontal, diagonal and low-pass filters respectively. We denote $\psi_I(x)$ as $\psi_{j,k}^{(t)}(x) = 2^j \psi^{(t)}(2^j x - k)$, $x \in [0,1]^2$, where:

$$\psi^{(v)}(x_1,x_2) = \varphi(x_1)\phi(x_2), \qquad \psi^{(h)}(x_1,x_2) = \phi(x_1)\varphi(x_2),$$
$$\psi^{(d)}(x_1,x_2) = \varphi(x_1)\varphi(x_2), \qquad \psi^{(f)}(x_1,x_2) = \phi(x_1)\phi(x_2).$$

For a given J, representing the maximum scale under consideration, and $J_0 < J$ the coarsest scale, we can define a wavelet basis of $p = 2^{2J}$ elements as follows: take $j \in \{J_0, \dots, J_1 = J-1\}$; for each $j \neq J_0$, consider wavelets of the types (v), (h) and (d), whereas for $j = J_0$ include also the type (f). For each level j and type (t), consider offsets $k = (k_1, k_2)$, $k_1 = 0, \dots, 2^j - 1$, $k_2 = 0, \dots, 2^j - 1$.

The wavelets basis functions can thus be grouped in $3(J-J_0)+1$ different subbands, each of which is identified by a scale j and a type (t).

3.1.2 Problem formulation

We consider the inverse problem of determining $u^\dagger \in X$ from the measurements:

$$m = Au^\dagger + \epsilon, \tag{3.1}$$

where $A : X \to Y$ is a linear bounded operator between the Hilbert spaces X and Y, and $\epsilon \in Y$ is a perturbation such that $\|\epsilon\|_Y \leq \delta$. In the particular case of LA-CT, the operator A stands for the Radon transform R_Γ, which is a continuous linear operator that maps the set $X = L^2(\Omega)$, where $\Omega \subset \mathbb{R}^2$, onto $Y = L^2([-\Gamma,\Gamma] \times [-S,S])$, where $[-\Gamma,\Gamma]$ defines the limited angular range and $[-S,S]$ is the range of signed distances from the origin.

In order to find an adequate solution to the ill-posed problem arising from eq. (3.1), the sought signal can be approximated by the minimiser of a regularised objective function, expressed as the sum of a data-fidelity term, which measures the faithfulness of the solution to the observation model, and an additional regularisation term, which orients the reconstruction process towards a solution with some desired characteristics and improves stability to noise. In particular, one of the simplest and surprisingly effective way to remove noise is to expand the signal in an orthogonal basis and to enforce the sparsity of such expansion.

Here, we consider the orthogonal wavelet basis $\{\psi_I\}_{I\in\mathbb{N}}$ in X. Let $W : X \to \ell^2(\mathbb{N})$ denote the transform mapping any $u \in X$ to the sequence of its components w.r.t. the wavelet basis: $(Wu)_I = \langle u, \psi_I\rangle_X$, where $\langle\cdot,\cdot\rangle_X$ denotes the inner product in X. We assume to know *a priori* that the exact solution $u^\dagger$ is sparse w.r.t. the wavelet basis $\{\psi_I\}_I$:

$$Wu^\dagger = w^\dagger \in \ell^0(\mathbb{N}). \tag{3.2}$$

The justification of such an approach is that in the wavelet domain, the signal gets usually concentrated in a few significant coefficients while the noise is spread out in smaller and diffuse values that can be more easily suppressed. However, the ℓ^0-pseudonorm can prove very troublesome to deal with from an optimisation point of view. Instead, it is common practice to encode the *a priori* information regarding the sparsity of $w^\dagger$ through the ℓ^1-norm. Such a convex relaxation results in the following variational minimisation problem:

$$\min_{w\in\ell^1(\mathbb{N})} \|AW^*w - m\|_Y^2 + \lambda\|w\|_{\ell^1}, \tag{3.3}$$

where $\lambda > 0$ is a scalar defining the trade-off between the data-fidelity term and the regularisation term.

Problem (3.3) can equivalently be formulated in the image domain:

$$\min_{u\in Z} \|Au - m\|_Y^2 + \lambda\|u\|_Z, \tag{3.4}$$

where $Z \subset X$ is defined such that $Z = \{u \in X : Wu \in \ell^1(\mathbb{N})\}$. In particular, in the case of LA-CT, it is possible to show that the ℓ^1-norm $\|w\|_{\ell^1}$ of the components of the wavelet representation of a $L^2(\Omega)$ function u is equivalent to the Besov norm $B^1_{1,1}(\Omega)$ of u, $\|u\|_{B^1_{1,1}}$ (see, *e.g.*, [52, section 1.4.1]). As problems (3.3) and (3.4) are mathematically equivalent, the solution of one of the two problems provides an immediate and unambiguous solution to the second one. In the remainder of this document, we will focus on the formulation in (3.3), without loss of generality.

The ℓ_1-norm relaxation in the minimisation problem (3.3) shows appealing properties, not only in terms of convexity, but also in terms of consistence of the solution w.r.t. the noise level. In particular, if the noise level δ tends to 0, there exists a suitable choice of the regularisation parameter $\lambda = \lambda(\delta)$ that ensures the convergence of the solution w^δ_λ of (3.3) to the exact wavelet coefficients $w^\dagger$. Furthermore, it can be proven that such convergence occurs with linear rate, as reported in Proposition 3.1.

Proposition 3.1. *Let $w^\dagger$ satisfy* (3.2)*, and suppose $A : X \to Y$ is an injective sequentially weak*-to-weak continuous bounded linear operator. Define w_δ^λ a solution of problem* (3.3) *associated with a regularisation parameter λ and a noise level δ. For sufficiently small δ, provided that λ is chosen such that $\lambda = c_0\delta$, then there exists a positive constant $c_1 = c_1(c_0, A, \|w^\dagger\|_{\ell^0})$ such that*

$$\|w^\dagger - w_\delta^\lambda\|_{\ell^1} \leq c_1\delta. \tag{3.5}$$

This proposition is an immediate consequence of [66, Corollary 2] which, interestingly, does not require $w^\dagger$ to satisfy a classical source condition, but instead mainly relies on the sparsity assumption (3.2) and on the injectivity of the operator A. Such property may be restrictive in some applications and consequently, numerous alternative results require some weaker assumptions, such as the Restricted Isometry Property (RIP). In the tomographic application, however, the injectivity of the Radon transform is verified, even in the limited-angle case.

From now on, we assume that λ is chosen as a linear function of δ and, for the sake of simplicity, we omit it in the notations: u_δ^λ is denoted u_δ and w_δ^λ is denoted w_δ.

We now introduce a finite-dimensional approximation of the regularised problem (3.3). We consider the subspaces $X_p = \text{span}\{\psi_I\}_{I=1}^p \subset X$ and $Y_q = \text{span}\{\varphi_j\}_{j=1}^q \subset Y$, where $\{\varphi_j\}_{j=1}^q$ is an orthogonal wavelet basis of Y. The finite-dimensional wavelet transform W_p is defined such that $W_p = \{w \in \mathbb{R}^{\mathbb{N}} \mid w_I = 0,\ \forall I > p\}$ and we call $\mathbb{P}_p$ the orthogonal projection of $\ell^2(\mathbb{N})$ onto W_p. Furthermore, we denote the orthogonal projection of X onto X_p by $\widetilde{\mathbb{P}}_p = W^*\mathbb{P}_pW$ and the orthogonal projection of Y onto Y_q by $\mathbb{P}_q$. Finally, let $A_{p,q}$ be the representation of the operator A in the subspaces X_p, Y_q for any choice of $p, q > 0$, that is: $A_{p,q} = \mathbb{P}_qA\widetilde{\mathbb{P}}_p^*$. The finite-dimensional minimisation problem then reads:

$$\min_{w \in W_p} \|A_{p,q}W^*w - \mathbb{P}_qm\|_Y^2 + \lambda\|w\|_{\ell^1}. \tag{3.6}$$

The solution of (3.6), denoted $w_{\delta,p,q}$, benefits from the convergence results formulated in Proposition 3.2.

Proposition 3.2. *Let $w^\dagger$ satisfy* (3.2) *and A be an injective operator. Suppose moreover that for a suitable choice of p, q it is possible to ensure that $\|w^\dagger - \mathbb{P}_pw^\dagger\|_{\ell^2} \leq c_p\delta$ and $\|(I - \mathbb{P}_q)A\|_{X\to Y} \leq c_q\delta$. Then, provided that λ is chosen as $\lambda = c_0\delta$, there exists a positive constant c_2 (depending on $\|A\|, \|w^\dagger\|_{\ell^1}$, on the choice of $\{\psi_I\}, \{\varphi_j\}$ and on the constants c_0, c_1, c_p, c_q) such that:*

$$\|w_{\delta,p,q} - w^\dagger\|_{\ell^1} \leq c_2\delta. \tag{3.7}$$

The proof, which is inspired from an application of the variational source condition reported in [66, Section 3], is reported in Appendix C.

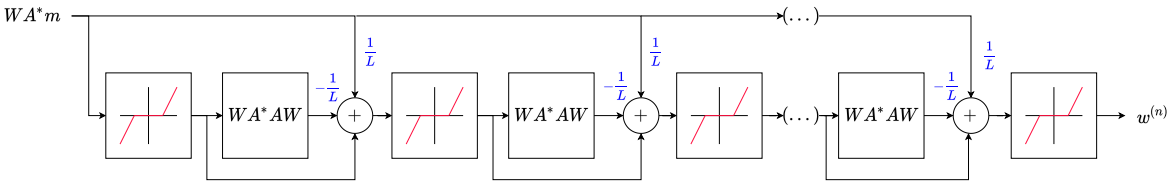


Figure 3.1: Block diagram illustrating the functioning of ISTA. The corresponding implementation is given in algorithm 8.

3.1.3 Iterative Soft-Thresholding Algorithm

A well-known technique to solve the minimisation problem (3.6) is the Iterative Soft-Thresholding Algorithm (ISTA), proposed by Daubechies *et al.* in 2004 [52]. It essentially consists in selecting an initial guess $w^{(0)} \in \mathbb{R}^p (\cong W_p)$ and in iteratively building the sequence $\{w^{(n)}\}_{n=1}^N$ as follows:

$$w^{(n)} = \mathcal{S}_{\lambda/L}\left(w^{(n-1)} - \frac{1}{L} W A_{p,q}^* A_{p,q} W^* w^{(n-1)} + \frac{1}{L} W A_{p,q}^* m\right), \tag{3.8}$$

where $\frac{1}{L} > 0$ can be interpreted as a (fictitious) time step, and $S_\beta(w)$ is the component-wise soft-thresholding operator, defined for any $\beta > 0$ as:

$$[\mathcal{S}_\beta(w)]_I = S_\beta(w_I), \qquad S_\beta(w_I) = \begin{cases} w_I + \beta & \text{if } w_I < -\beta \\ 0 & \text{if } |w_I| \leq \beta \\ w_I - \beta & \text{if } w_I > \beta \end{cases}.$$

The detailed implementation of ISTA is given in algorithm 8 and fig. 3.1 shows a block diagram illustrating its functioning.

Algorithm 8 Standard ISTA

Input: m, acquired measurements
λ, regularisation parameter
L, positive constant
$w^{(0)}$, initial guess
N, maximum number of iterations
tol, tolerance

for $n = 1, \ldots, N$ **do**
  Compute intermediate iterate $r^{(n)} = w^{(n-1)} - \frac{1}{L} W A_{p,q}^* A_{p,q} W^* w^{(n-1)} + \frac{1}{L} W A_{p,q}^* m$
  Apply soft-thresholding: $w^{(n)} = \mathcal{S}_{\lambda/L}\left(r^{(n)}\right)$
  if $\|w^{(n)} - w^{(n-1)}\|_2^2 / \|w^{(n-1)}\|_2^2 < tol$ **then**
    End computations
  end if
end for

The convergence of the sequence $\{w^{(n)}\}_{n=1}^N$ to a minimiser $w_{\delta,p,q}$ of (3.6) has been studied in an infinite dimensional context, in [27]. As for the discrete setting, convergence results have been proposed in [21], which can be translated in our case as follows:

Proposition 3.3. *If L is chosen such that $L \geq \|WA_{p,q}^* A_{p,q} W^*\|/2$ then the sequence $\{w^{(n)}\}_{n=1}^N$ generated via (3.8) by any $w^{(0)} \in \mathbb{R}^p$ converges in ℓ^2 to the solution $w_{\delta,p,q}$ of (3.6). Moreover, there exist $c_3 > 0$ and $0 \leq a < 1$ (both depending on $A_{p,q}$, L and $\|w^\dagger\|_{\ell^2}$) such that*

$$\|w^{(N)} - w_{\delta,p,q}\|_{\ell^2} \leq c_3 a^N. \tag{3.9}$$

3.1.4 A perturbation of ISTA

In order to best apprehend the convolutional interpretation of ISTA detailed in next section, we also present a modification of ISTA and the consequences thereof in terms of convergence.

Let us consider an operator $Z : \ell^2(\mathbb{N}) \to \ell^2(\mathbb{N})$ such that:

$$\|WA_{p,q}^* A_{p,q} W^* - Z\|_{\ell^2 \to \ell^2} \leq \rho. \tag{3.10}$$

We then replace $WA_{p,q}^* A_{p,q} W^*$ with Z in eq. (3.8). Such a perturbation of ISTA results in a slightly different sequence $\{w_\rho^{(n)}\}$, obtained by selecting $w_\rho^{(0)} \in \mathbb{R}^p$ and by iterating:

$$w_\rho^{(n)} = \mathcal{S}_{\lambda/L}\left(w_\rho^{(n-1)} - \frac{1}{L} Z w_\rho^{(n-1)} + \frac{1}{L} WA_{p,q}^* m\right). \tag{3.11}$$

It is first possible to show that the convergence of the sequence $\{w_\rho^{(n)}\}$ to the minimiser $w_{\delta,p,q}$ depends on the magnitude of the perturbation ρ. The proof of such a result is reported in Appendix C.

Proposition 3.4. *Let $w^{(0)} = w_\rho^{(0)}$, $L \geq \|WA_{p,q}^* A_{p,q} W^*\|$ and consider $N_0, \eta_0 > 0$. Then there exists a constant $\tilde{c}_4$, depending on $L, A, w^{(0)}, \|w^\dagger\|_{\ell^2}$ and on N_0, η_0, such that if $N \geq N_0$ and $\rho N \leq \eta_0$ then*

$$\|w_\rho^{(N)} - w_{\delta,p,q}\|_{\ell^2} \leq c_3 a^N + \tilde{c}_4 \rho N. \tag{3.12}$$

If, moreover, N, ρ are chosen as $N > \frac{\ln(\delta^{-1})}{\ln(a^{-1})}$ and $\rho < \frac{\delta}{N}$, then (for $c_4 = c_3 + \tilde{c}_4$):

$$\|w_\rho^{(N)} - w_{\delta,p,q}\|_{\ell^2} \leq c_4 \delta. \tag{3.13}$$

Finally, by combining the results obtained in Proposition 3.2 and Proposition 3.4, we obtain the following convergence estimate:

Theorem 3.1. *Let $w^\dagger$ satisfy (3.2) and let A be injective. For sufficiently small δ, select a regularisation parameter $\lambda = c_0\delta$. Select p, q s.t. $\|w^\dagger - \mathbb{P}_p w^\dagger\| \leq c_p \delta$ and $\|(I - \mathbb{P}_q)A\|_{X \to Y} \leq c_q \delta$. Let $L \geq \|WA_{p,q}^* A_{p,q} W^*\|$ and consider the perturbed ISTA iterations (3.11), where the operator Z satisfies (3.10), $N = \log_a \delta$ and $\rho = \frac{\delta}{N}$. Then, there exists a positive constant c_5 (depending on the previously introduced constants $c_0, c_1, c_2, c_3, c_4, c_p, c_q$) such that, for sufficiently small δ,*

$$\|w_\rho^{(N)} - w^\dagger\|_{\ell^2} \leq c_5 \delta. \tag{3.14}$$

3.2 ISTA as a convolutional neural network

3.2.1 ISTA interpreted as a neural network

It is already well-known in the literature that the unfolded iterations of ISTA can be considered as the layers of a neural network (see, *e.g.* [81]). Mathematically, the n^{th} iteration of ISTA can indeed be written as:

$$w^{(n)} = \mathcal{S}_{\frac{\lambda}{L}}\left(w^{(n-1)} - \frac{1}{L}K^{(n)}w^{(n-1)} + \frac{1}{L}b^{(n)}\right), \quad n = \{1, \cdots, N\}, \tag{3.15}$$

where $K^{(n)} = WA^*_{p,q}A_{p,q}W^*$ and $b^{(n)} = WA^*_{p,q}m$. It can be observed that eq. (3.15) exhibits a structure similar to that of an MLP layer (see eq. (2.38)). It thus appears clearly that the n^{th} iteration of ISTA can be interpreted as the n^{th} layer of a fully connected neural network, where $K^{(n)}$ stands for the matrix of weights and $b^{(n)}$ is the bias vector. More precisely, since $K^{(n)}$ and $b^{(n)}$ are actually independent of n in eq. (3.15), such an ANN falls into the class of recurrent neural networks. As for the soft-thresholding operator $\mathcal{S}_\beta$, it plays the role of the nonlinear activation function. In particular, the analogy between $\mathcal{S}_\beta$ and more traditional activation functions, such as the ReLU (eq. (2.41)) or sigmoid functions, can be highlighted with the observation that the soft-thresholding operator can be written in terms of rectified linear units:

$$\begin{aligned}\mathcal{S}_\beta(x) &= \max(0, x-\beta) - \max(0, -x-\beta) &(3.16)\\ &= \mathrm{ReLU}(x-\beta) - \mathrm{ReLU}(-x-\beta), &(3.17)\end{aligned}$$

for any $x \in \mathbb{R}$ (it is applied component-wise if x is a multi-dimensional vector).

This similarity between the iterations in eq. (3.15) and the structure of traditional ANNs suggests that deep learning strategies can be incorporated to ISTA in order to improve the performance thereof. To do so, it is possible, for example, to replace the entries of $K^{(n)} = WA^*_{p,q}A_{p,q}W^*$ with trainable parameters. In addition, one may also want to learn further parameters, such as the regularisation parameter λ or the steplength L. Formally, let $\theta \in \Theta$ denote the vector containing all the trainable parameters of the neural network, and $f_\theta : Y \to \ell^1(\mathbb{N})$ the map parameterised by $\theta \in \Theta$, which takes as input $m \in Y_q$ and computes N iterations according to (3.15), where, for each n, $K^{(n)} \in \mathbb{R}^{p\times p}$ is specified in θ and $b^{(n)} = WA^*_{p,q}m$. The resulting neural network, which is therefore composed of N layers corresponding to the N first iterations of ISTA, can then be trained through an optimisation process, such as the ones described in chapter 2.

The strength of such a strategy is that it combines the mathematically justified structure of a variational method while allowing the incorporation of meaningful information extracted from a dataset of examples. Furthermore, for any p, q and N, it is possible to choose a particular value θ_0 of the parameters such that each layer of $f_{\theta_0}(m)$ is exactly equivalent to the ISTA iterations associated with the measurements m. This is particularly interesting inasmuch as such parameters θ_0, which can easily be calculated, may serve as initial parameters for the optimisation process. From that

point, the learning algorithm will then be able to gradually modify the entries of $K^{(n)}$ as well as the additional trainable parameters in order for the neural network to provide an ever improving image quality in the reconstructions.

However, such an approach still presents a significant drawback: depending of the value of p, the matrix $K^{(n)}$ at each layer n may be exceedingly large, and the training of the resulting neural network is therefore likely to rapidly become impractical. It is possible to circumvent this problem by interpreting the unfolded iterations of ISTA in eq. (3.15) no longer as the layers of a fully connected neural network, but as the layers of a CNN. The advantages of such a modification are multiple, as discussed in section 2.3. In particular, using a CNN instead of a fully connected ANN brings about a drastic reduction in terms of memory requirements, a higher computational efficiency, and furthermore imparts the model appealing properties such as parameter sharing and equivariance.

In the next subsections, we provide a detailed description of the operations that allow to translate the unfolded iterations of ISTA in terms of layers of a CNN. We thus provide a convolutional architecture that can reproduce the exact behaviour of ISTA (or a perturbation of the kind described by (3.10)), for a specific choice of the parameters.

From now on, we focus on the case $X = L^2(\Omega)$. Furthermore, although our approach is sufficiently general to handle higher-dimensional spaces, we will focus on the two-dimensional case, that is, $\Omega \subset \mathbb{R}^2$ (*e.g.*, $\Omega = [0,1]^2$).

3.2.2 Convolutional interpretation of ISTA

Encoding the iterations of ISTA as a CNN essentially rests upon the convolutional nature of the normal operator A^*A under consideration. From now on, we therefore assume that A^*A is a convolutional kernel operator, *i.e.*,

$$\mathbf{K}_{I,I'} = (WA^*AW^*)_{I,I'} = \langle A^*A\psi_I, \psi_{I'}\rangle_X = \int_{\mathbb{R}^2}\int_{\mathbb{R}^2} K(x,x')\psi_I(x)\psi_{I'}(x')\mathrm{d}x\mathrm{d}x', \quad (3.18)$$

where $K(x,x') = K(x-x')$. The fact that the wavelet basis can be split in subbands, each of which is identified by a couple j,(t) (see section 3.1.1), implies that the matrix $\mathbf{K}$ representing A^*A in the wavelet domain can be seen as a block matrix. We now aim at describing the application of each block $\mathbf{K}_{j\to j'}^{(t)\to(t')} w_j^{(t)}$ by means of the following operations:

1. **Discrete convolution.** Let $B \in \mathbb{R}^{b\times b}$, $C \in \mathbb{R}^{(2b-1)\times(2b-1)}$, and denote the elements of C with indices i,j , being $i = -b+1,\dots,0,\dots,b-1$, $j = -b+1,\dots,0,\dots,b-1$. Then, $C * B \in \mathbb{R}^{b\times b}$:

$$(C*B)_{k,l} = \sum_{i=0}^{b-1}\sum_{j=0}^{b-1} C_{k-i,l-j}B_{i,j} \quad (3.19)$$

2. **Upsampling.** Let $B \in \mathbb{R}^{b\times b}$; then, $\mathscr{U}(B) \in \mathbb{R}^{2b\times 2b}$ satisfies:

$$\mathscr{U}(B)[2k:2k+1, 2l:2l+1] = \begin{bmatrix} B_{k,l} & 0 \\ 0 & 0 \end{bmatrix} \quad \forall k,l = 0,\dots,b-1, \quad (3.20)$$

where the notation $\mathscr{U}(B)[2k:2k+1, 2l:2l+1]$ is used to denote a submatrix of $\mathscr{U}(B)$ containing the rows from $2k$ to $2k+1$ and all the columns from $2l$ to $2l+1$. We call $\mathscr{U}^\eta$ the iterated application of $\mathscr{U}$: $\mathscr{U}^\eta = \mathscr{U} \circ \ldots \circ \mathscr{U}$ (η times).

3. **Downsampling.** Let $B \in \mathbb{R}^{2b\times 2b}$; then, $\mathscr{D}(B) \in \mathbb{R}^{b\times b}$ satisfies:

$$\mathscr{D}(B)_{k,l} = B_{2k,2l} \quad \forall k,l = 0,\ldots,b-1. \tag{3.21}$$

We call $\mathscr{D}^\eta$ the iterated application of $\mathscr{D}$: $\mathscr{D}^\eta = \mathscr{D} \circ \ldots \circ \mathscr{D}$ (η times).

As it can be observed, such operations are very similar to the traditional CNN operations described in section 2.3.2. In particular, the downsampling operation is nothing but a form of pooling. As for the upsampling, it can be interpreted as an 'unpooling' operation, whose implementation is straightforward and completely suitable for the CNN framework. The key idea of our approach is precisely to translate the application of the matrix $\mathbf{K}$ exclusively in terms of the aforementioned convolution, upsampling and downsampling operations, in such a way to make the unfolded ISTA iterations compatible with the CNN technology. Such a procedure is described in proposition 3.5, which provides the full description of the convolutional interpretation of the matrix representing A^*A in the wavelet domain. The presented result can be compared to the ones already known in literature (see *e.g.* [50, Formula (4.2)]), although the more complicated structure of the wavelet basis implies some substantial differences.

Proposition 3.5. *Let $\mathbf{K} \in \mathbb{R}^{p\times p}$ be the matrix representing an operator A^*A satisfying (3.18) in a 2D wavelet basis $\{\psi_I\}_{I=1}^p$. For a vector $w \in \mathbb{R}^p$, let $w_j^{(t)}$ be the vector of the wavelet components related to basis functions of scale j and type (t). Let $\mathbf{K}_{j\to j'}^{(t)\to(t')}$ denote the block of $\mathbf{K}$ corresponding to the $j,(t)$ subset of the column indices and the $j',(t')$ subset of the row indices. Then:*

$$\mathbf{K}_{j\to j'}^{(t)\to(t')} w_j^{(t)} = \begin{cases} \mathscr{D}^\delta(\widetilde{\mathbf{K}}_{j\to j'}^{(t)\to(t')} * W_j^{(t)}) & \text{if } j > j', \\ \widetilde{\mathbf{K}}_{j\to j'}^{(t)\to(t')} * W_j^{(t)} & \text{if } j = j', \\ \widetilde{\mathbf{K}}_{j\to j'}^{(t)\to(t')} * \mathscr{U}^\delta(W_j^{(t)}) & \text{if } j < j', \end{cases} \tag{3.22}$$

where $\delta = |j'-j|$, and $\widetilde{\mathbf{K}}_{j\to j'}^{(t)\to(t')} \in \mathbb{R}^{(2^{\hat{j}+1}-1)\times(2^{\hat{j}+1}-1)}$ (with $\hat{j} = \max(j,j')$) defined such that:

$$\left[\widetilde{\mathbf{K}}_{j\to j'}^{(t)\to(t')}\right]_d = \int_{\mathbb{R}^2}\int_{\mathbb{R}^2} K(x-x'-2^{-\hat{j}}d)\ \psi_{j,0}^{(t)}(x)\ \psi_{j',0}^{(t')}(x')\mathrm{d}x\mathrm{d}x', \tag{3.23}$$

for $d = (d_1,d_2)$; $\quad d_1,d_2 = \{-2^{\hat{j}}+1,\ldots,0,\ldots,2^{\hat{j}}-1\}$. The matrix $W_j^{(t)} \in \mathbb{R}^{2^j\times 2^j}$ is obtained by reshaping the vector $w_j^{(t)} \in \mathbb{R}^{2^{2j}}$ so that $[W_j^{(t)}]_d$ is the component w_I whose index is identified by $(j,(t),d)$.

Proof. Let I, I' be identified by $(j,(t),k)$ and $(j',(t'),k')$, respectively. Then,

$$\begin{aligned}
[\mathbf{K}]_{I',I} &= \int_{\mathbb{R}^2}\int_{\mathbb{R}^2} K(x-x')\ \psi^{(t')}_{j,k'}(x')\ \psi^{(t)}_{j,k}(x)dxdx' \\
&= \int_{\mathbb{R}^2}\int_{\mathbb{R}^2} K(x-x')\ \psi^{(t')}_{j,0}(x'-2^{-j'}k')\ \psi^{(t)}_{j,0}(x-2^{-j}k)dxdx' \\
&= \int_{\mathbb{R}^2}\int_{\mathbb{R}^2} K(x+2^{-j}k-x'-2^{-j'}k')\psi^{(t')}_{j,0}(x)\psi^{(t)}_{j,0}(x)dxdx' \\
&= \int_{\mathbb{R}^2}\int_{\mathbb{R}^2} K(x-x'-2^{-\hat{j}}(2^{\delta^-}k'-2^{\delta^+}k))\psi^{(t')}_{j,0}(x)\psi^{(t)}_{j,0}(x)dxdx' = \left[\widetilde{\mathbf{K}}^{(t)\to(t')}_{j\to j'}\right]_d,
\end{aligned}$$

where $\delta^+ = \max(0, j-j')$, $\delta^- = \max(0, j'-j)$, and $d = 2^{\delta^-}k' - 2^{\delta^+}k$. For the sake of simplicity, we use $\mathbf{K}$ instead of $\mathbf{K}^{(t)\to(t')}_{j\to j'}$, $\widetilde{\mathbf{K}}$ instead of $\widetilde{\mathbf{K}}^{(t)\to(t')}_{j\to j'}$, w instead of $w^{(t)}_j$, W instead of $W^{(t)}_j$. Moreover, we call $\mathcal{I}$ the set of indices $\mathcal{I} \subset \{1,\dots,p\}$ belonging to the wavelet scale j and type (t).

First, consider the case $j = j'$. Then $\delta = \delta^+ = \delta^- = 0$, and it holds

$$[\mathbf{K}]_{I',I} = \left[\widetilde{\mathbf{K}}\right]_d, \qquad d = k' - k.$$

Therefore,

$$\begin{aligned}
[\mathbf{K}w]_{I'} &= \sum_{I\in\mathcal{I}} [\mathbf{K}]_{I',I}\, w_I = \sum_{I\in\mathcal{I}} \left[\widetilde{\mathbf{K}}\right]_{k'-k(I)} w_I \\
&= \sum_{k_1=-2^j}^{2^j}\sum_{k_2=-2^j}^{2^j} \left[\widetilde{\mathbf{K}}\right]_{k'_1-k_1,k'_2-k_2} W_{k_1,k_2} = [\mathbf{K} * W]_{I'}.
\end{aligned}$$

Let now $j < j'$. Then $\delta = \delta^+ > 0$, $\delta^- = 0$, and

$$\begin{aligned}
[\mathbf{K}w]_{I'} &= \sum_{I\in\mathcal{I}} [\mathbf{K}]_{I',I}\, w_I = \sum_{I\in\mathcal{I}} \left[\widetilde{\mathbf{K}}\right]_{k'-2^{\delta^+}k(I)} w_I \\
&= \sum_{k_1=-2^{j'}}^{2^{j'}}\sum_{k_2=-2^{j'}}^{2^{j'}} \left[\widetilde{\mathbf{K}}\right]_{k'_1-2^{\delta^+}k_1,k'_2-2^{\delta^+}k_2} \mathscr{U}^{\delta^+}(W)_{2^{\delta^+}k_1,2^{\delta^+}k_2} = [\mathbf{K} * \mathscr{U}^{\delta}W]_{I'}.
\end{aligned}$$

Finally, let $j > j'$. Then $\delta^+ = 0$, $\delta = \delta^- > 0$, and

$$\begin{aligned}
[\mathbf{K}w]_{I'} &= \sum_{I\in\mathcal{I}} [\mathbf{K}]_{I',I}\, w_I = \sum_{I\in\mathcal{I}} \left[\widetilde{\mathbf{K}}\right]_{2^{\delta^-}k'-k(I)} w_I \\
&= \sum_{k_1=-2^j}^{2^j}\sum_{k_2=-2^j}^{2^j} \left[\widetilde{\mathbf{K}}\right]_{2^{\delta^-}k'_1-k_1,2^{\delta^-}k'_2-k_2} W_{k_1,k_2} = [\mathscr{D}^{\delta}(\mathbf{K} * W)]_{I'}.
\end{aligned}$$

□

Thanks to the convolutional interpretation detailed in Proposition 3.5, one can substitute the multiplication $K^{(n)}w^{(n-1)}$ in the neural network architecture proposed in eq. (3.15) by the operations encoded in eq. (3.22), namely: decomposition of $w^{(n-1)}$ in wavelet subbands, upscaling, application of convolutional filters, downscaling. In other words, we have demonstrated that it is possible to build a CNN that, for a special choice of its parameters θ_0, is able to reproduce the exact application of ISTA.

The possibility to translate the iterations of ISTA into such a CNN architecture opens up new interesting perspectives for solving linear inverse problems with sparse-promoting regularisation. In the same way as mentioned in section 3.2.1, one can consider converting some of the coefficients involved in the convolutional interpretation of ISTA into learnable parameters, and train them through a learning process that uses θ_0 as initialisation point. The major asset of such a CNN w.r.t. the fully connected neural network previously described is the considerable reduction in the number of parameters it involves. Specifically, the fully connected ANN described in section 3.2.1 rests upon the standard, finite-dimensional representation of A^*A, which, inasmuch as it is a matrix of $\mathbb{R}^{p\times p}$, necessitates $O(p^2)$ coefficients. The convolutional representation of the operator A^*A presented in Proposition 3.5, in contrast, requires only $O(p)$ elements.

The approach proposed in this doctoral thesis precisely takes advantage of this appealing framework. Before presenting the details of its underlying principles in section 3.3, we illustrate the convolutional representation of ISTA reported in (3.22) by means of a short example, and, in a second time, we study the extent to which such representation is affected if we employ convolutional filters smaller than the ones described in eq. (3.23).

3.2.3 Illustrative example

We provide an illustrative example of the convolutional representation of ISTA reported in (3.23). Let us consider the case of 64×64 images, thus corresponding to $J = 6$ and $p = 2^{12}$. Create a wavelet basis consisting of three scales of wavelets, from $J_0 = 3$ to $J_1 = 5$. The resulting basis $\{\psi_I\}_{I=1}^{p}$ can be therefore split into $3(6-3)+1 = 10$ subbands: 4 associated to the scale $j = 3$ (types: (h), (v), (d) and (f)), 3 associated to the scales $j = 4$ (types: (h), (v), (d)) and 3 with $j = 5$. Each subband is composed of 2^{2j} elements.

According to section 3.2.2, if we call $\mathbf{K} \in \mathbb{R}^{p\times p}$ the matrix representing the operator A^*A in the wavelet basis $\{\psi_I\}$, then, following the procedure described below is equivalent to applying the matrix $\mathbf{K}$ to a vector $w \in \mathbb{R}^p$ (representing the wavelet transform of an image).

1. The first step consists in splitting the vector w into its 10 wavelet subbands, each of which identified by a scale j and a type (t). This process is illustrated in fig. 3.2. In practice, the vector $w_j^{(t)} \in \mathbb{R}^{2j}$ can be interpreted as a matrix $W_j^{(t)} \in \mathbb{R}^{j\times j}$. Each element $[W_j^{(t)}]_d = [W_j^{(t)}]_{(d_1,d_2)}$ is the component associated to the basis function $\psi_{j,d}^{(t)}(x) = 2^j\psi^{(t)}(2^j x_1 - d_1, 2^j x_2 - d_2)$.

2. Then, for each subband $j,(t)$, compute the 10 vectors $\mathbf{K}_{j\to j'}^{(t)\to(t')} w_j^{(t)}$, where each matrix $\mathbf{K}_{j\to j'}^{(t)\to(t')}$ is a $2^{2j'} \times 2^{2j}$ block of the matrix $\mathbf{K}$. Those vectors represent the contributions of $w_j^{(t)}$ on the subband $j',(t')$ of the vector $\mathbf{K}w$. As shown in (3.22), these computations can be performed via downsampling, upsampling and convolution. Consider the case $j = J_0 = 3$:

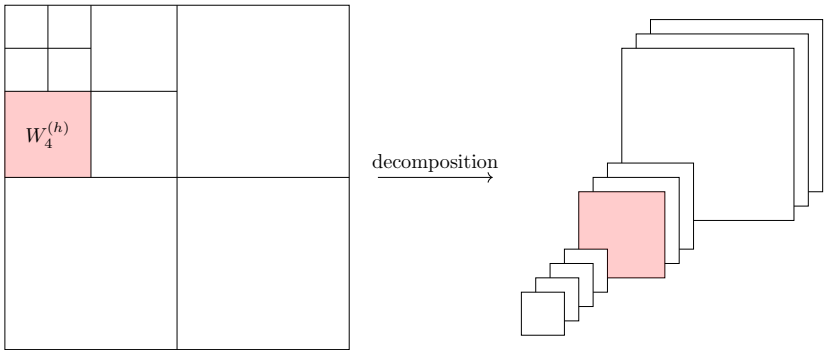


Figure 3.2: Interpretation of (3.22). Step 1: decomposition of the wavelet representation in subbands.

- if $j' = 3$, then $\hat{j} = 3$ and $\delta = 0$. Thus, by computing the convolution of the 8×8 matrix $W_3^{(t)}$ with the 15×15 filter $\widetilde{\mathbf{K}}_{3\to 3}^{(t)\to(t')}$, we obtain a 8×8 matrix representing the vector $\mathbf{K}_{3\to 3}^{(t)\to(t')} w_3^{(t)} \in \mathbb{R}^{64}$.
- if $j' = 4$, then $\hat{j} = 4$ and $\delta = 1$. Since $j < j'$, we shall apply the third variant in formula (3.22), to compute the 16×16 matrix associated with $\mathbf{K}_{3\to 4}^{(t)\to(t')} w_3^{(t)}$. This means that we must first upsample the matrix $W_3^{(t)}$, and then convolve it with the 31×31 filter $\widetilde{\mathbf{K}}_{3\to 4}^{(t)\to(t')}$.
- if $j' = 5$, then $\hat{j} = 5$ and $\delta = 2$. Again, $j < j'$, meaning that we make use of the third variant of (3.22). Here, the matrix $W_3^{(t)}$ must be upsampled twice before being convolved with the 63×63 filter $\widetilde{\mathbf{K}}_{3\to 5}^{(t)\to(t')}$.

Consider now the case $j = 4$:

- if $j' = 3$, then $\hat{j} = 4$ and $\delta = 1$. Since $j > j'$, we shall employ the first variant in (3.22), meaning that we first compute the convolution between the 16×16 matrix $W_4^{(t)}$ and the 31×31 filter $\widetilde{\mathbf{K}}_{4\to 3}^{(t)\to(t')}$, and then downscale the result to recover the 8×8 matrix describing $\mathbf{K}_{4\to 3}^{(t)\to(t')} w_4^{(t)}$.
- the case $j' = 4$ is analogous to $3 \to 3$, using 31×31 filters $\widetilde{\mathbf{K}}_{4\to 4}^{(t)\to(t')}$.
- the case $j' = 5$ is analogous to $3 \to 4$: we first perform upsampling and then convolution.

Finally, in the case $j = J_1 = 5$,

- if $j' = 3$, then we first compute the convolution between the 32×32 matrix $W_5^{(t)}$ and the 63×63 filter $\widetilde{\mathbf{K}}_{5\to 3}^{(t)\to(t')}$, and then downsample twice.
- if $j' = 4$, we only downsample once, as in the case $4 \to 3$.
- if $j' = 5$, we only apply convolution, as in the cases $3 \to 3$ and $4 \to 4$, but with 63×63 filters.

These operations are depicted in fig. 3.3.

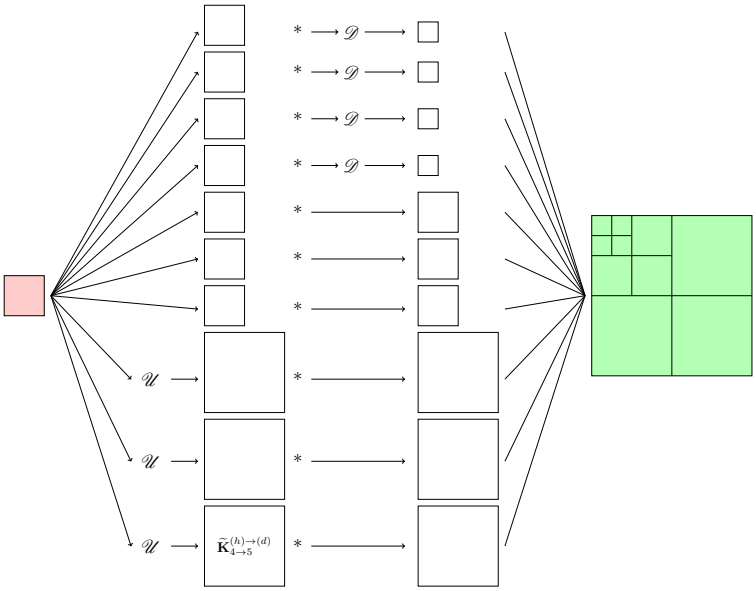


Figure 3.3: Interpretation of (3.22). Step 2: downsampling, upsampling and convolution.

3. The last step amounts to computing each $j', (t')$ subband of the vector $\mathbf{K}w$, by summing up all the related contributions coming from the vectors $w_j^{(t)}$. Each subband thus computed is a $2^{j'} \times 2^{j'}$ matrix. An illustration of this operation is provided in fig. 3.4.

3.2.4 Smaller convolutional filters

When designing a CNN, it is common practice to make use of a large number of small-sized convolutional filters. The architecture defined by (3.22) and (3.23), on the contrary, is based on a rather small number of filters that are usually bigger than the wavelet subbands they are applied to. More precisely, it employs $(3(J-J_0)+1)^2$ filters, each of which being of dimensions $(2^{\hat{j}+1}-1)^2$. Furthermore, each part of the vector $w^{(n-1)}$ interacts only with $(3(J-J_0)+1)$ of the filters. Although this design is still totally compatible with the CNN technology, it is possible that in practice, the use of such large filters brings about some speed reduction in the training process. This is due to the fact that, to the best of our knowledge, convolutions with filters that are larger than the input have not been optimised yet in the existing deep learning libraries. In this section, we investigate the possibility of substituting such large filters with smaller ones, and analyse the effect of such a modification in terms of convergence results.

We consider filters of dimensions $\tau \times \tau$, where $\tau = 2\xi + 1$ and $\xi > 1$, obtained by extracting the central elements of the large filters $\widetilde{\mathbf{K}}_{j\to j'}^{(t)\to(t')}$. We define $\widetilde{\mathbf{K}}^{\tau} = (\widetilde{\mathbf{K}}_{j\to j'}^{(t)\to(t')})_{\tau}$ such that:

$$\left[\widetilde{\mathbf{K}}^{\tau}\right]_d = \begin{cases} \left[\widetilde{\mathbf{K}}_{j\to j'}^{(t)\to(t')}\right]_d & if\ \|d\|_\infty \leq \xi, \\ 0 & if\ \|d\|_\infty > \xi. \end{cases} \tag{3.24}$$

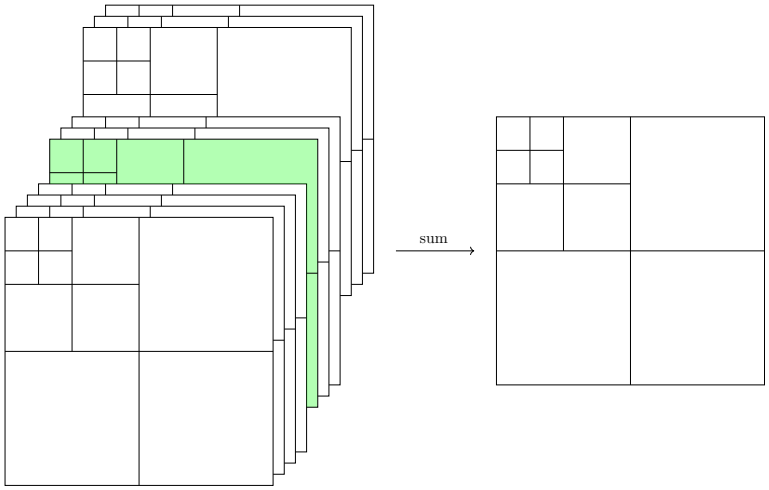


Figure 3.4: Interpretation of (3.22). Step 3: collecting all the wavelet subbands.

We claim that this modification amounts to performing a perturbation of ISTA, such as the one described in Proposition 3.4, where the parameter ρ is a suitable function of τ. This can be demonstrated by providing a bound on the coefficients of the filters that are discarded due to (3.24).

It is possible to obtain such an estimate by making further assumptions on the operator A. More precisely, suppose that A^*A is a convolutional operator of kernel K, as in (3.18). Assume furthermore that for $x \neq x'$, the kernel $K(x, x')$ is smooth and such that:

$$K(x,x') \leq \frac{C}{|x-x'|}, \quad \text{and} \quad |\nabla_x K(x,x')| + |\nabla_{x'} K(x,x')| \leq \frac{C}{|x-x'|^2}. \tag{3.25}$$

Equation (3.27) is in particular satisfied whenever A^*A is a ΨDO of order -1 with constant coefficients, that is

$$A^*Af = \mathcal{F}^{-1}\left\{a(\xi)\mathcal{F}\left\{f\right\}(\xi)\right\}, \qquad a(\xi) \sim \frac{1}{|\xi|} \; as \; \xi \to 0.$$

In addition, equation (3.27) is also surely satisfied by the Radon transform R_Γ, since its normal operator $R_\Gamma^* R_\Gamma$ is associated with the kernel $K(x,x') = \chi_\Gamma(x-x')/|x-x'|$, as mentioned in section 1.2.

We furthermore assume first-order vanishing moment property for wavelet basis functions:

$$\int_{\mathbb{R}^2} \psi_I(x)dx = 0. \tag{3.26}$$

This property is for example verified by the 2D Haar wavelets, except for the subbands of type (f).

Proposition 3.6. *Let the operator A satisfy* (3.18) *and* (3.25). *Let the indices I, I' denote two wavelets of scales j, j', type $(t), (t')$ and offsets k, k'. Let ψ_I and $\psi_{I'}$ meet the condition* (3.26) *and let $d_{I,I'}$ be the distance between the supports of ψ_I and $\psi_{I'}$.*

Whenever $d_{I,I'} > 0$, it holds:

$$\mathbf{K}_{I,I'} = (A^*A\psi_I, \psi_{I'})_X \leq c\frac{2^{-2(j+j')}}{d_{I,I'}^3} \tag{3.27}$$

It can be noted that the decay reported in (3.27) is reminiscent of formula (9.4.5) in [49], for the specific choice of $n = 2$, $\tilde{d} = 1$, $r = 2t = -1$, as well as formula (4.26) in [18], for $M = 2$.

Proof. According to (3.25), and to (3.26), for any $x_0 \in \operatorname{supp}\psi_I$, $x_0' \in \operatorname{supp}\psi_{I'}$, there exist two points ξ, ξ' respectively belonging to the same supports such that:

$$\begin{aligned}\mathbf{K}_{I,I'} &= \int_{\mathbb{R}^2}\int_{\mathbb{R}^2}(K(x,x') - K(x,x_0'))\psi_I(x)\psi_{I'}(x')\mathrm{d}x\mathrm{d}x' \\ &\leq \int_{\mathbb{R}^2}\int_{\mathbb{R}^2}|\nabla_{x'}K(x,\xi')||x'-x_0'|\psi_I(x)\psi_{I'}(x')\mathrm{d}x\mathrm{d}x' \\ &\leq C\int_{\mathbb{R}^2}\int_{\mathbb{R}^2}\frac{|x'-x_0'|}{|x-\xi'|^2}\psi_I(x)\psi_{I'}(x')\mathrm{d}x\mathrm{d}x' \leq C\int_{\mathbb{R}^2}\int_{\mathbb{R}^2}\frac{|x-x_0||x'-x_0'|}{|\xi-\xi'|^3}\psi_I(x)\psi_{I'}(x')\mathrm{d}x\mathrm{d}x'.\end{aligned}$$

The quantity $|\xi - \xi'|$ is bounded from below by $d_{I,I'}$ by definition. Moreover, $|x - x_0| \leq \operatorname{diam}(\operatorname{supp}\psi_I) = c2^{-j}$, and finally $\int_{\mathbb{R}^2}\psi_I(x)\mathrm{d}x \leq 2^j|\operatorname{supp}\psi_I| = 2^{-j}$ (analogous arguments hold on I'). □

From (3.27) and (3.23), one can infer a bound on the elements of the convolutional filters:

$$\left[\widetilde{\mathbf{K}}_{j\to j'}^{(t)\to(t')}\right]_d \leq c\frac{2^{-\hat{j}}}{(\|d\|_\infty - 1)^3}, \tag{3.28}$$

provided that $\|d\|_\infty > 1$. Based on this result as well as on (3.22), it is possible to obtain an explicit bound of the type of (3.13), on the perturbation induced by the thresholding (3.24).

3.3 ΨDONet: theoretical principles

The theoretical results presented in sections 3.1 and 3.2 pave the way to a novel convolutional network architecture, called ΨDONet. Here, we report the general idea inspiring such a reconstruction algorithm for sparsity-promoting regularisation. We furthermore provide a theoretical result ensuring the convergence of the proposed algorithm.

3.3.1 ΨDONet: a CNN to learn pseudodifferential operators

As mentioned in section 3.2.2, whenever the operator A^*A is of convolutional type, it is possible to design a reconstruction algorithm based on a CNN architecture with N layers, each of which is described in (3.15). More specifically, the bias vectors appearing in (3.15) are $b^{(n)} = WA_{p,q}^*m$, whereas the linear operators $K^{(n)}$ are interpreted as a combination of downscaling, upscaling, and convolution, as shown in (3.22). As

demonstrated in Proposition 3.5, if the entries of the convolutional filters are chosen as in (3.23), such a CNN is equivalent to performing N iterations of ISTA.

The key idea of the proposed algorithm is to incorporate deep learning processes in such a convolutional structure, by converting some of the filter coefficients into trainable parameters. To do so, we split each one of the $3(J - J_0) + 1$ filters involved in each layer into two parts: a central $\tau \times \tau$ submatrix (where τ is a predefined hyperparameter of the algorithm) and the outer frame. We then consider the entries of the external frame as fixed, that is, non learnable, and we assume that their values are specified by (3.23). Conversely, we define the central filter coefficients as trainable parameters, meaning that their values will be gradually modified throughout a learning process. Splitting the filter entries into trainable and non-trainable coefficients as it is suggested presents several advantages. In particular, it offers the possibility to control the trade-off between the complexity of the CNN model, that is, the number of parameters to be learnt, and the level of likeness between the network and standard ISTA, thanks to the hyperparameter τ. We will see furthermore that learning only the central part of the filters is particularly judicious in the case of ΨDOs and FIOs.

The trainable part of the filters involved in each layer n are collected in a vector ζ_n, and ultimately stored in the global vector θ, possibly together with additional learnable parameters. The resulting network is denoted f_θ^τ. The goal of the proposed CNN-based algorithm is then to find a parameter θ such that the network f_θ^τ is a good approximation of the map that, given some measurements m, outputs the solution $w^\dagger = Wu^\dagger$ that satisfies eq. (3.1).

It is clear that, among all the possible choices of the optimal parameter, the network could select the vector θ_0 which exactly replicates the ISTA iterations, that is, the vector such that also the central entries of each filter are specified by (3.23). However, if the optimal choice of θ differs from θ_0, this implies that the network has extracted helpful information from the training examples it could observe, and has learnt 'something more' than the ISTA iterations associated with the operator A^*A. Such an improvement can be apprehended as follows: the aforementioned splitting between the central part of the convolutional filters and their outer frame in the network f_θ^τ allows to regard the kernel K of the operator under consideration as the sum of two kernels $K_0 + K_1$, where K_0 is the known part of the model, and K_1 is the kernel of an operator to be learnt. Due to its trainable nature, K_1 thus provides the potential to add information to the known (fixed) part of the model K_0 and as such, can be seen as an adjunct for the direct betterment of the normal operator A^*A within the reconstruction process. Since the difference between standard ISTA and the trained network will only occur in the central entries of the convolutional filters, and according to the analysis of section 3.2.4, we can argue that the learnt operator is a suitable approximation of a *pseudodifferential* operator, hence the name we propose for this novel CNN-based reconstruction algorithm: **ΨDONet**.

There are multiple reasons for which the learning process could lead to a better performance than that of ISTA. First, a better choice of the parameters enables to lower the numerical errors caused by the discrete representation of A^*A, which might have a significant effect due to the error propagation amongst the iterations. Secondly,

we may also reduce model errors that come from the definition of the operator A itself. Finally, this perturbation could be seen as providing a representation of A^*A w.r.t. a slightly different basis, which may allow to better satisfy the sparsity assumption on the solutions.

For such reasons, the proposed algorithm ΨDONet is specifically recommended whenever the original operator A^*A is a ΨDO itself: its kernel representation by means of convolutional filters indeed contains all its most significant entries in the centre of its filters. Focusing the learning process on those particular coefficients thus ensures that the learnt corrections will affect the most important elements defining the operator while allowing the reduction in the number of trainable parameters via the tuning of τ.

The use of ΨDONet is also highly recommended if the normal operator A^*A is a FIO: in such event, the largest entries of its convolutional filters are situated in the centre and along some lines, possibly stretching away from the centre. This is the case of the limited-angle Radon transform, whose kernel reads:

$$K(x, y) = \frac{1}{|x - y|}\chi_\Gamma(x - y),$$

where χ_Γ is the indicator function of the cone in $\mathbb{R}^2$ between the angles $-\Gamma$ and Γ. As it is numerically verified in chapter 4, the convolutional filters associated with this operator exhibit large values only in their central coefficients and along two lines having the same slope of the ones delimiting the cone. This results in a somewhat curious shape for the filters, which evokes that of a *bowtie*. We will show that the application of ΨDONet to this operator, which brings learnt corrections only to the centre of the bowties, is still remarkably effective.

3.3.2 Convergence results of ΨDONet

In this subsection, we present a theoretical result that holds true for ΨDONet, regardless of its particular implementation.

We first introduce the following probabilistic framework, in analogy with the approach in [54]: let $\mathcal{B} = \{u \in X_p : \ Wu \in \ell^1(\mathbb{N}); \|Wu\|_{\ell^1} \leq C_\mathcal{B}\}$ and u be a random variable with a probability distribution μ on the space $\mathcal{B}$, where μ represents some prior information on the solution of the inverse problem. We define furthermore ϵ as a random variable in Y_q with distribution ν, modelling the measurement error. We suppose that u and ϵ are independent. Then, the measurement $m = A_{p,q}u + \epsilon$ is also a random variable, defined on the product space $X_p \times Y_q$ with density $A_*\mu \otimes \nu$, where $A_*\mu$ denotes the pushforward of μ to Y via the linear map A.

As detailed in section 2.1.2, the performance of a learning system can be measured by means of a loss function. In the present case, we propose to define the loss function associated to the network f_θ^τ as:

$$\mathcal{L}(\theta; \mu, \nu) = \mathbb{E}_{u\sim\mu,\epsilon\sim\nu}\left[\|f_\theta^\tau(A_{p,q}u + \epsilon) - Wu\|_{\ell^2}^2\right] \tag{3.29}$$

Then, the optimal neural network is the one described by the parameters θ^* that satisfy:

$$\theta^* = \underset{\theta\in\Theta}{\operatorname{argmin}}\, \mathcal{L}(\theta;\mu,\nu). \tag{3.30}$$

In order to provide some convergence results of the optimal network $f^\tau_{\theta^*}$, it is convenient to first analyse the properties of $f^\tau_{\theta_0}$, where θ_0 is such that the network is equivalent to performing N iterations of (modified) ISTA. In particular, we will make use of the following rough estimate:

Lemma 3.1. *There exist two constants $k_1, k_2 > 0$ (depending on $C_\mathcal{B}, L, \rho$, $\|A_{p,q}\|$, $w^{(0)}$,N) such that, for all $u \in \mathcal{B}$ and $\epsilon \in Y_q$,*

$$\|f^\tau_{\theta_0}(A_{p,q}u+\epsilon) - Wu\|_{\ell^2} \le k_1 + k_2\|\epsilon\|_{Y_q} \tag{3.31}$$

Proof. According to (3.15), defining $\kappa = 1 + \frac{\|A^*_{p,q}A_{p,q}\|+\rho}{L}$, we get

$$\begin{aligned}\|f^\tau_{\theta_0}(A_{p,q}u+\epsilon) - Wu\|_{\ell^2} &\le \|f^\tau_{\theta_0}(A_{p,q}u+\epsilon)\|_{\ell^2} + \|u\|_{X_p}\\ &\le \kappa^N\|w^{(0)}\| + \left(1+\kappa+\ldots+\kappa^{N-1}\right)\|A_{p,q}u+\epsilon\|_{Y_q}+C_\mathcal{B}\\ &\le \kappa^N\|w^{(0)}\| + C_\mathcal{B} + \frac{\kappa^N-1}{\kappa-1}(\|A_{p,q}\|C_\mathcal{B}+\|\epsilon\|_{Y_q})\end{aligned}$$

□

We now focus on the special case in which the noise ϵ is a Gaussian random vector, meaning that $\nu = \mathcal{N}(0,\sigma^2 I_q)$, where I_q the identity matrix in $\mathbb{R}^{q\times q}$. We recall that in such event:

$$\mathbb{E}[\|\epsilon\|^2_{Y_q}] = q\sigma^2, \qquad \mathbb{E}[\|\epsilon\|^4_{Y_q}] \le 3q^2\sigma^4. \tag{3.32}$$

From this point, we can rely both on Lemma 3.1 and on Theorem 3.1, introduced in section 3.1.4, to provide a more refined estimate. We can indeed observe that the convergence result reported in (3.14) is independent of the choice of $\epsilon = m - Au$, as long as $\|\epsilon\| \le \delta$. Furthermore, the constant c_5 appearing in (3.14) can depend on u, but only through an upper bound on $\|w^\dagger\|_{\ell^1}$. This enables to come to the following conclusion:

Lemma 3.2. *Suppose $\epsilon \sim \mathcal{N}(0,\sigma^2 I_q)$ and let $\delta = \sigma^{1/\eta}$, being $\eta > 1$. There exists $\sigma_0 > 0$ such that, for $\sigma < \sigma_0$, then for every $u \in \mathcal{B}$*

$$\mathbb{E}_{\epsilon\sim\nu}\left[\|f^\tau_{\theta_0}(Au+\epsilon) - Wu\|^2_{\ell^2}\right] \le c_5^2\delta^2 + 2\sqrt{2}k_1^2\delta^{\eta-1} + 2\sqrt{6}k_2^2 q\delta^{3\eta-1}.$$

If, moreover, $\eta = 3$ and $\sigma < \min\{\sigma_0, q^{-1/2}\}$, then there exists a constant c^* (depending on c_5, k_1, k_2) such that

$$\mathbb{E}_{\epsilon\sim\nu}\left[\|f^\tau_{\theta_0}(Au+\epsilon) - Wu\|^2_{\ell^2}\right] \le c^*\delta^2. \tag{3.33}$$

Proof. We start by considering that

$$\mathbb{E}_{\epsilon\sim\nu}\left[\|f^\tau_{\theta_0}(Au+\epsilon)-Wu\|^2_{\ell^2}\right]=\int_{Y_q}\|f^\tau_{\theta_0}(Au+\epsilon)-Wu\|^2_{\ell^2}d_\nu(\epsilon)$$
$$=\int_{\|\epsilon\|<\delta}\|f^\tau_{\theta_0}(Au+\epsilon)-Wu\|^2_{\ell^2}d_\nu(\epsilon)+\int_{\|\epsilon\|>\delta}\|f^\tau_{\theta_0}(Au+\epsilon)-Wu\|^2_{\ell^2}d_\nu(\epsilon).$$

We now employ (3.14) on the first term and Hölder inequality on the second term. Moreover, in view of Chebyshev's inequality, $\nu(\{\|\epsilon\|>\delta\})\leq\frac{\sigma^2}{\delta^2}$. Therefore,

$$\mathbb{E}_{\epsilon\sim\nu}\Big[\|f^\tau_{\theta_0}(Au+\epsilon)-Wu\|^2_{\ell^2}\Big]\leq c_5^2\delta^2\left(1-\frac{\sigma^2}{\delta^2}\right)+\frac{\sigma}{\delta}\left(\mathbb{E}_{\epsilon\sim\nu}\Big[\|f^\tau_{\theta_0}(Au+\epsilon)-Wu\|^4_{\ell^2}\Big]\right)^{\frac{1}{2}}$$
$$\leq c_5^2\delta^2+\frac{\sigma}{\delta}\left(8k_1^4+8k_2^4\mathbb{E}\Big[\|\epsilon\|^4_{Y_q}\Big]\right)^{\frac{1}{2}}.$$

By (3.32) and using the fact that $\sigma=\delta^\eta$, we immediately verify the first thesis, and imposing $\eta=3$ and $\delta^{2\eta}q<1$ we get (3.33) with $c^*=c_5^2+2\sqrt{2}k_1^2+2\sqrt{6}k_2^2$. □

On the basis of Lemma 3.2, we can easily deduce the following convergence result concerning the optimal network $f^\tau_{\theta^*}$:

Proposition 3.7. *Consider $\epsilon\sim\mathcal{N}(0,\sigma^2I_q)$ with $\delta=\sigma^{1/3}$ and let θ^* satisfy* (3.30). *There exists $\sigma_1>0$ such that, for $\sigma\leq\min\{\sigma_1,q^{1/2}\}$, it holds*

$$\mathcal{L}(\theta^*;\mu,\nu)\leq c^*\delta^2. \tag{3.34}$$

In particular, this implies that the random variable $f^\tau_{\theta^*}(A_{p,q}u+\epsilon)$ converges to Wu in mean as $\delta\to0$.

Proof. By definition of θ^* and by Lemma 3.2,

$$\mathcal{L}(\theta^*;\mu,\nu)\leq\mathcal{L}(\theta_0;\mu,\nu)=\int_{\mathcal{B}}\int_{Y_q}\|f^\tau_{\theta_0}(A_{p,q}u+\epsilon)-Wu\|^2_{\ell^2}d_\nu(\epsilon)d_\mu(u)$$
$$\leq\int_{\mathcal{B}}c^*\delta^2d_\mu(u)=c^*\delta^2.$$

□

Thus, the optimal network $f^\tau_{\theta^*}$ can provide a good approximation of the solution map underlying the inverse problem of interest. However, in the same way as explained in section 2.1.2, solving the minimisation problem stated in (3.30) is impracticable inasmuch as the probability distributions μ and ϵ are unknown in practice. Instead, the available information on which the training can be performed consists of a set of i.i.d samples of the random variables u and ϵ drawn from their respective distributions μ and ν. The learning task is therefore addressed by minimising a discretised loss functional, and eventually leads to a parameter θ that corresponds to a local minimum of such a cost function. At the end of the training process, the quality of the trained network can be assessed by analysing its generalisation abilities (see section 2.1.3). Such an analysis has been proposed in detail in [54], and can be extended, in spite of some differences in the assumptions w.r.t. the ones in this work, to the problem under consideration.

As a proof of concept, the concrete application of ΨDONet is investigated in next chapter, in the particular case of limited-angle tomography.

4 ΨDoNet in practice: application to the case of LA-CT

In this chapter, we explore the implementational aspects of ΨDONet when applied to the special case of LA-CT with discrete setting. The practical design of the proposed architectures directly derives from the theoretical results presented in chapter 3, which hold true when the general operator $A_{p,q}$ in eq. (3.1) is interpreted as the limited-angle Radon transform R_Γ.

Formally, we assume that, after suitable discretisation, we are given some measurements $\mathbf{m} \in \mathbb{R}^q$, also called sinogram, such that:

$$\mathbf{m} = \mathbf{R}_\mathbf{\Gamma}\mathbf{u}^\dagger + \epsilon, \tag{4.1}$$

where $\mathbf{u}^\dagger \in \mathbb{R}^p$ denotes the (unknown) discrete and vectorised image, $\mathbf{R}_\mathbf{\Gamma} \in \mathbb{R}^{q\times p}$ describes a discretised version of the Radon transform where the angles are limited in the arc specified by Γ, and $\epsilon \in \mathbb{R}^q$ models the measurement noise. We call $\mathbf{w}^\dagger$ the $\mathbb{R}^p$-vector such that $\mathbf{W}\mathbf{u}^\dagger = \mathbf{w}^\dagger$ where $\mathbf{W} \in \mathbb{R}^{p\times p}$ represents a discretisation of the wavelet transform. Thus, the regularised minimisation problem under consideration reads:

$$\min_{\mathbf{w}\in\mathbb{R}^p} \|\mathbf{R}_\mathbf{\Gamma}\mathbf{W}^*\mathbf{w} - \mathbf{m}\|_2^2 + \lambda\|\mathbf{w}\|_1 \tag{4.2}$$

As detailed in chapter 3, the proposed reconstruction algorithm ΨDONet rests upon the unfolding of ISTA, in which each iteration is converted into the layer of a trainable CNN. Implementing such a CNN first requires to numerically verify the theoretical results of section 3.2.2. Section 4.1 thus provides the convolutional filter building procedure and the result of their application to a simple phantom. In the second part of that section, we propose a detailed description of two different implementations of ΨDONet, namely Filter-Based ΨDONet (ΨDONet-F) and Operator-based ΨDONet (ΨDONet-O). Those architectures are then tested on different sets of simulated data: the testing setups as well as the corresponding numerical results are presented in section 4.2.

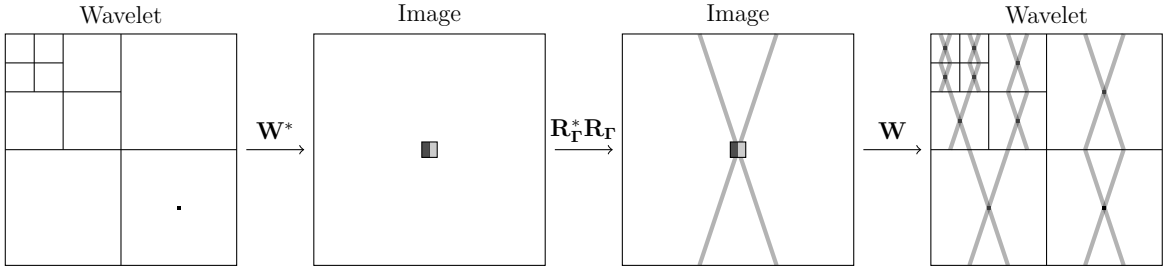


Figure 4.1: Illustration of the proposed way to compute the filters of the convolutional kernel operator K in the LA-CT case. The initial object (*left*) is generated in the wavelet domain by setting all its entries but one to zero. The only non-zero entry is situated at the centre of one of its wavelet subbands. By applying the operator $\mathbf{W}\mathbf{R}_{\mathbf{\Gamma}}^{*}\mathbf{R}_{\mathbf{\Gamma}}\mathbf{W}^{*}$ to this initial object, one obtains a new object in the wavelet domain, whose subbands exhibit a bowtie-shaped structure. Those bowtie subbands constitute a first set of convolutional filters. The complete collection of convolutional filters required for the approximation of $\mathbf{W}\mathbf{R}_{\mathbf{\Gamma}}^{*}\mathbf{R}_{\mathbf{\Gamma}}\mathbf{W}^{*}$ is formed by reiterating such procedure until the central entry of all the wavelet subbands in the initial object has been visited. In this illustration, we have considered a three-scale wavelet decomposition ($J_1 = J_0 + 2$), meaning that the total number of convolutional filters amounts to $((J_1 - J_0 + 1)^2 + 1)^2 = (3^2 + 1)^2 = 100$.

4.1 Implementational description

4.1.1 Convolutional interpretation of the LA-CT normal operator

The design of a CNN-based algorithm that is able to reproduce the behaviour of ISTA for a special choice of its parameters rests upon the demonstration that a convolutional, normal operator can be translated in terms of upscaling, downscaling and convolution operations, as described in Proposition 3.5. In this section, we provide a numerical verification of such theoretical results in the case of the limited-angle Radon transform. For this purpose, we first need to identify the various blocks of the matrix $\mathbf{K}$ in eq. (3.18), representing the back-projection operator in the wavelet domain. In other words, we wish to determine the convolutional filters described in eq. (3.23), which, once applied as defined in eq. (3.22), provide a reliable approximation of the operator $\mathbf{W}\mathbf{R}_{\mathbf{\Gamma}}^{*}\mathbf{R}_{\mathbf{\Gamma}}\mathbf{W}^{*}$.

One way to compute such convolutional filters that proves to be a numerically advantageous alternative to eq. (3.23), is illustrated in fig. 4.1. Let us consider an object whose representation in the wavelet domain has only one non-zero entry, situated at the centre of one of its wavelet subbands. Applying the operator $\mathbf{W}\mathbf{R}_{\mathbf{\Gamma}}^{*}\mathbf{R}_{\mathbf{\Gamma}}\mathbf{W}^{*}$ to this initial object results in a new object, whose subbands exhibit a bowtie-shaped structure. Those bowtie subbands form a first set of convolutional filters. The entire collection of convolutional filters necessary for the approximation of $\mathbf{W}\mathbf{R}_{\mathbf{\Gamma}}^{*}\mathbf{R}_{\mathbf{\Gamma}}\mathbf{W}^{*}$ can then be obtained by reiterating this process until the central pixel of all the wavelet

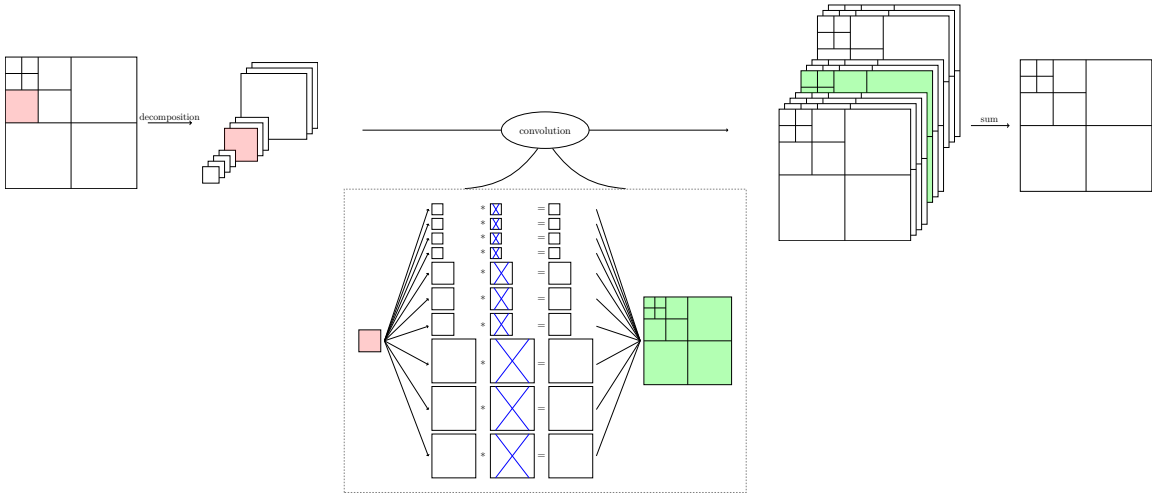


Figure 4.2: Illustration of the way the convolutional filters, computed as described in fig. 4.1, are applied to each wavelet subband of the initial object, after up- and down-sampling operations, in order to approximate the operator $\mathbf{W}\mathbf{R}_{\mathbf{\Gamma}}^{*}\mathbf{R}_{\mathbf{\Gamma}}\mathbf{W}^{*}$.

subbands in the initial object has been visited. A numerical example of convolutional filter is provided in fig. 4.4.

Once computed, such convolutional filters can be applied to the wavelet subbands of any object as illustrated in fig. 4.2, in order to imitate the behaviour of the operator $\mathbf{W}\mathbf{R}_{\mathbf{\Gamma}}^{*}\mathbf{R}_{\mathbf{\Gamma}}\mathbf{W}^{*}$. First, each wavelet subband of the object under consideration is replicated $3(J_1 - J_0 + 1) + 1$ times. Those replicas are then either downsampled, or upsampled, or kept with the same dimensions, depending on the scale of the filter they are to be convolved with. The set of convolutional filters employed on the replicas of a wavelet subband of scale j and type (t) is the set of filters beforehand generated by applying the operator $\mathbf{W}\mathbf{R}_{\mathbf{\Gamma}}^{*}\mathbf{R}_{\mathbf{\Gamma}}\mathbf{W}^{*}$ to an object whose only non-zero pixel is located at the centre of its wavelet subband of exact same scale j and type (t). After the convolutions between the replicas and the filters have been carried out, the resulting subbands are reassembled to constitute the wavelet representation of a new object. This operation is repeated for all the subbands of the initial object and ultimately, the $3(J_1 - J_0 + 1) + 1$ resulting items are summed. The final outcome is an approximation of the operator $\mathbf{W}\mathbf{R}_{\mathbf{\Gamma}}^{*}\mathbf{R}_{\mathbf{\Gamma}}\mathbf{W}^{*}$ applied to the initial object. Fig. 4.3 illustrates such a procedure when the object under consideration is a circular phantom, and fig. 4.5 shows the numerical results thus obtained, in the image domain.

Two observations are worth mentioning with regards to the creation and application of the above-defined convolutional filters. First, our practical implementation very slightly differs from the theory presented in eq. (3.22) as far as the downsampling is concerned. In our codes, downsampling is indeed applied before convolving the filter and the wavelet subband replica, and not after, as it is indicated in the theory. This choice is motivated by the diminution in terms of storage needs and running time such a change enables while preserving the correctness of the approximation. Secondly, both the theoretical analysis and the experimental tests proved that the dimensions

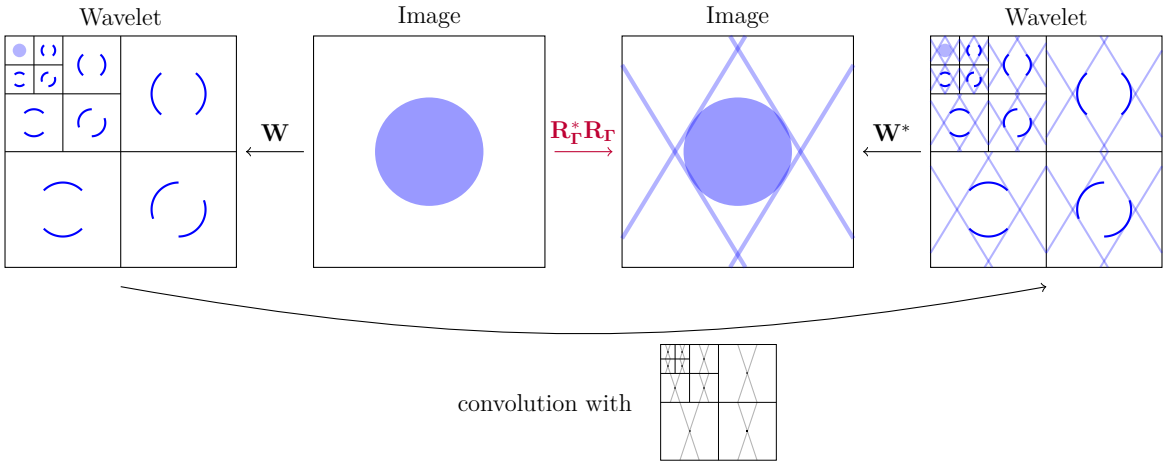


Figure 4.3: Figure illustrating the effect of the convolutional filters, computed as described in (fig. 4.1), when applied to a circular phantom (*second from the left*). As theoretically demonstrated in Proposition 3.5 and numerically verified in fig. 4.5, the application of $\mathbf{R}_\mathbf{\Gamma}^*\mathbf{R}_\mathbf{\Gamma}$ to the phantom can be approximated by the following procedure: first, decompose the image in the wavelet domain (*left*), then apply the convolutional filters to the wavelet subbands (according to fig. 4.2) and use the subbands thus computed (*right*) to recompose the final image. The latter, which exhibits bowtie-shaped artefacts, offers a reliable approximation of the image obtained by directly applying the normal operator $\mathbf{R}_\mathbf{\Gamma}^*\mathbf{R}_\mathbf{\Gamma}$ to the initial phantom.

of the convolutional filters employed for the approximation of $\mathbf{W}\mathbf{R}_\mathbf{\Gamma}^*\mathbf{R}_\mathbf{\Gamma}\mathbf{W}^*$ do affect the accuracy of the results. Initially, we assumed that the convolutional filters should be generated from only-one-non-zero-pixel objects with the same size $2^J \times 2^J$ as the image of interest (recall that $p = 2^{2J}$). However, we reached the conclusion that they actually have to be generated from twice bigger objects, that is of size $2^{J+1} \times 2^{J+1}$, in order to obtain an accurate approximation of the operator $\mathbf{W}\mathbf{R}_\mathbf{\Gamma}^*\mathbf{R}_\mathbf{\Gamma}\mathbf{W}^*$. The effects of the filter dimensions are shown in fig. 4.5.

4.1.2 Proposed implementations

The above described method for generating and applying the filters of the kernel operator $\mathbf{K}$ makes the concrete implementation of a convolutional algorithm that imitates the behaviour of ISTA possible. Mathematically, the convolutional implementation of ISTA, result-wise equivalent to the standard one, can be deduced both from eq. (3.15) and eq. (3.22), and reads:

$$\mathbf{w}^{(n+1)} = \mathcal{S}_{\frac{\lambda}{L}}\left(\mathbf{w}^{(n)} + \frac{1}{L}\left(\mathbf{W}\mathbf{R}_\mathbf{\Gamma}^*\mathbf{m} - \mathbf{K}\mathbf{w}^{(n)}\right)\right), \qquad n = \{0, \dots, N\}, \tag{4.3}$$

where $\mathbf{K}$ stands for the combination of downscaling, upscaling and convolution operations summarised in fig. 4.2. As previously mentioned, the interest of such an algorithm is that it presents the merits of the iterative, model-based method ISTA, while allowing the incorporation of data-driven approaches. We thus propose to take full advantage of

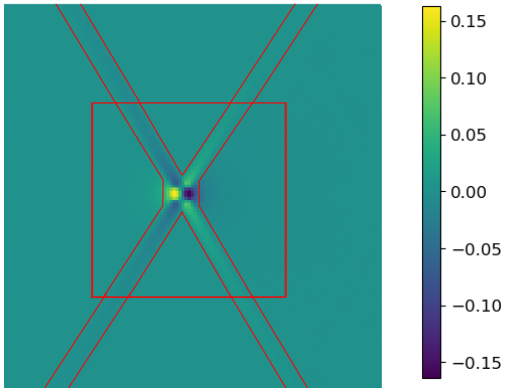


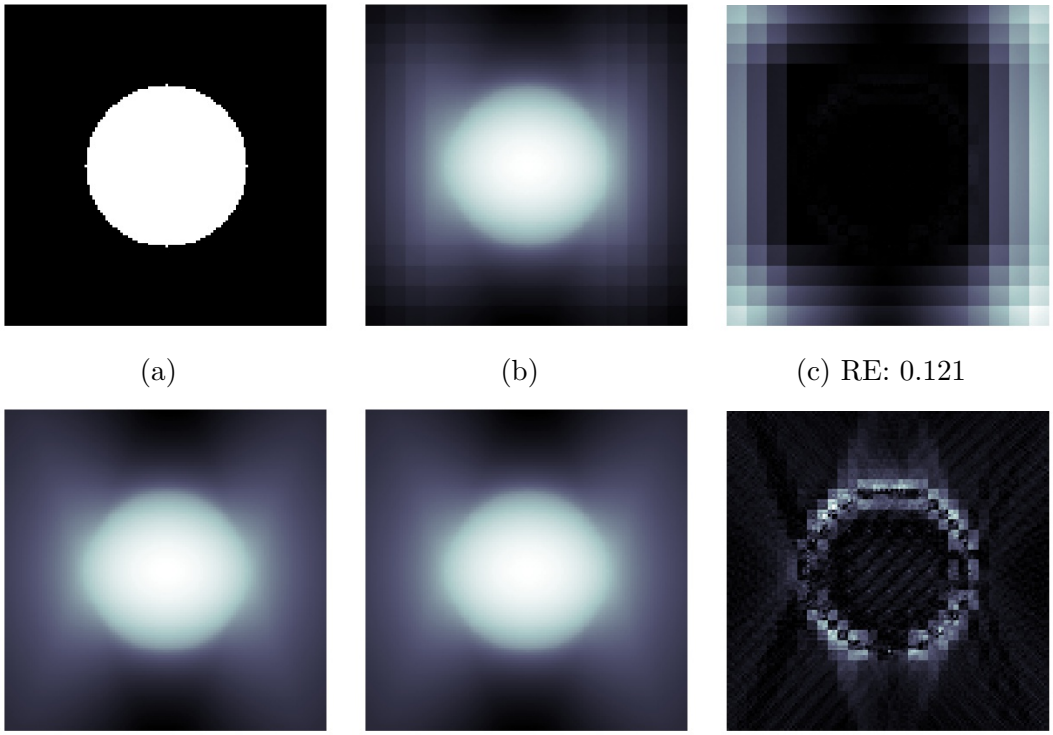
Figure 4.4: Example of a bowtie subband that can be employed as a convolutional filter of the kernel operator K. It was generated from a 256×256 initial object, according to the procedure detailed in section 4.1.1 and illustrated in fig. 4.1. Theory suggests that the pixels with highest intensities are spread according to a bowtie-shaped structure. In practice, they are even more condensed: most of the energy is concentrated along two diagonal lines that intersect in the center and whose inclination is defined by the limited angle: 95.8% of the ℓ_2-norm of the filter is concentrated along those two lines, from which 94.8% are inside the central red square.

this compatibility and profit from the remarkable potentials of deep neural networks by converting the hitherto fixed operator $\mathbf{K}$ into a partially trainable CNN. More specifically, we now consider the centre of the convolutional filters so far precomputed with the deterministic method presented in section 4.1.1 as parameters to be learnt from data, while the outer frame of the filters keeps unchanged. The theoretical considerations regarding such an architectural choice are provided in section 3.3.1. From a more computational point of view, learning only the central part of the convolutional filters of $\mathbf{K}$ rather than the whole filters offers the possibility to reduce the model complexity which, in the latter case, makes the training process burdensome if not impractical.

In practice, we propose to split the operator $\mathbf{K}$ into two operators: a first one, $\mathbf{K}_0$, standing for the fixed, known part of the model; and a second, trainable operator referred to as $\mathbf{K}_1 := \Lambda_\zeta^\tau$, where τ is a tunable hyperparameter and ζ represents the set of parameters to be learnt. In order to further improve reconstruction performance, we also decide to learn the soft-thresholding parameter as well as the ISTA time step, so far set to $1/L$. The convolutional network thus defined is none other than our proposed algorithm ΨDONet applied to the discretised LA-CT case. In section 4.1.2.1 and section 4.1.2.2, we investigate two different implementations of ΨDONet, which prove to be result-wise similar as it can be observed in section 4.2.2, but differ in their computational needs.

4.1.2.1 ΨDONet-F

The most natural way to implement ΨDONet consists in building two operators $\mathbf{K}_0$ and $\mathbf{K}_1$ with the same architecture as the convolutional operator $\mathbf{K}$, and whose filters



(a) (b) (c) RE: 0.121

(d) (e) (f) RE: 0.003

Figure 4.5: Effect of the standard back-projection operator and of its approximations with convolutional filters of different sizes. (a) shows the ground truth $\mathbf{u}^\dagger$ under consideration and (d) its standard back-projection $\mathbf{R}_{\mathbf{\Gamma}}^\star \mathbf{R}_{\mathbf{\Gamma}} \mathbf{u}^\dagger$, computed with the basic functions of the Python package `scikit-image`. (b) (resp. (e)) represents the approximation $\mathbf{K}\mathbf{u}^\dagger$ obtained with the convolutional filters beforehand generated from $2^J \times 2^J$ (resp. $2^{J+1} \times 2^{J+1}$) only-one-non-zero-pixel object. (c) and (f) show the absolute differences between the approximation of the back-projection operator and the expected value (d). The dynamic range of the plot is modified for better contrast.

are such that their sum, before any training, is strictly equivalent to the filters of $\mathbf{K}$: $\mathbf{K} = \mathbf{K}_0 + \mathbf{K}_1$. The operator $\mathbf{K}_0 := \breve{\mathbf{K}}^\tau$ is non trainable and its filters are a copy of the filters of $\mathbf{K}$, with the exception that the $\tau \times \tau$ central entries of each filter are set to zero. The operator $\mathbf{K}_1 := \Lambda_\zeta^\tau$, in contrast, involves $\tau \times \tau$-trainable filters that are initialised with the $\tau \times \tau$ central part of the filters of $\mathbf{K}$ (see fig. 4.6), and then gradually modified during to training process. This first implementation of ΨDONet, referred to as Filter-Based ΨDONet or ΨDONet-F, is formulated as:

$$\mathbf{w}^{(n+1)} = \mathcal{S}_{\gamma_n}\left(\mathbf{w}^{(n)} + \alpha_n\left(\mathbf{W}\mathbf{R}_{\mathbf{\Gamma}}^*\mathbf{m} - \beta_n\left(\breve{\mathbf{K}}^\tau \mathbf{w}^{(n)} + \Lambda_{\zeta_n}^\tau \mathbf{w}^{(n)}\right)\right)\right), \tag{4.4}$$

for $n = \{0, \ldots, N\}$, where the parameters to be learnt are $\{\gamma_0, \alpha_0, \beta_0, \zeta_0, \ldots, \gamma_N, \alpha_N, \beta_N, \zeta_N\}$. The parameters $\{\beta_0, \ldots, \beta_N\}$ have been added in such a way that the influence of the fixed operator $\breve{\mathbf{K}}^\tau$ w.r.t. the constant term $\mathbf{W}\mathbf{R}_{\mathbf{\Gamma}}^*\mathbf{m}$ can be adjusted in order to maximise the accuracy of the results. It is worth mentioning that for the particular

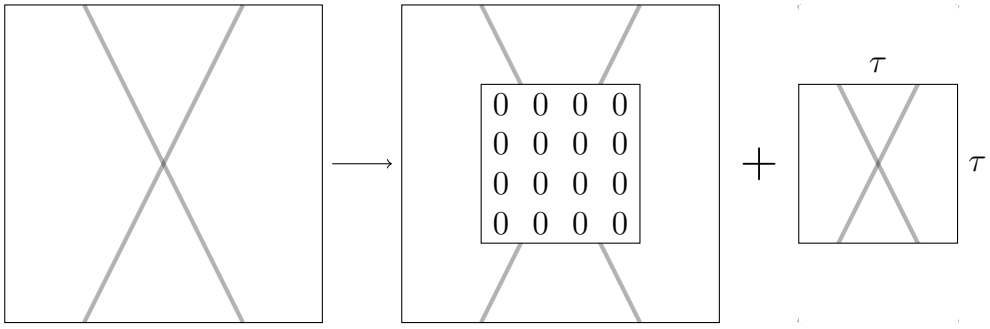


Figure 4.6: Figure illustrating how the convolutional filters of $\mathbf{K}$ are to be split to generate the filters of the operators $\mathbf{K}0 := \breve{\mathbf{K}}^\tau$ and $\mathbf{K}_1 := \Lambda_\zeta^\tau$ in ΨDONet-F. Each filter of $\mathbf{K}$ is partitioned into two filters whose sum is equivalent to the initial one. The filter of $\breve{\mathbf{K}}^\tau$ is a copy of the filter of $\mathbf{K}$ with the exception that the $\tau \times \tau$ central weights are set to zero. The filter of Λ_ζ^τ has dimensions $\tau \times \tau$ and is initialised with the central $\tau \times \tau$ central weights of the filter of $\mathbf{K}$.

choice of $\gamma_n = \frac{\lambda}{L}, \alpha_n = \frac{1}{L}, \beta_n = 1$, for any $n = \{0, \ldots, N\}$, this model before any training is exactly equivalent to standard ISTA.

The trade-off between the number of parameters that can be improved through the learning process and the trainability of the model is controlled by τ. For a sound choice of such a hyperparameter, the complexity of the model is sufficiently reduced to enable the convergence of the learning algorithm while allowing the enhancement of a significant number of weights in the filters.

This implementation has the merit of offering a clear interpretation of the role and meaning of the convolutional filters belonging to $\breve{\mathbf{K}}^\tau$ and Λ_ζ^τ. Those filters are indeed initialised with the filters of the operator $\mathbf{K}$, which reproduces the behaviour of $\mathbf{W}\mathbf{R}_\mathbf{\Gamma}^*\mathbf{R}_\mathbf{\Gamma}\mathbf{W}^*$. Thus, modifying their weights through the training process can be regarded as a direct improvement of the back-projection operator in the wavelet domain.

ΨDONet-F has led to very satisfactory preliminary results, presented in section 4.2. However, training such a model on big images may quickly become extremely onerous in terms of running time and storage requirements. Such problems may arise while training ΨDONet-F on images of dimensions greater or equal to 256×256. Unlike typical CNNs, which usually make use of small-sized convolutional filters, the filters of $\breve{\mathbf{K}}^\tau$ in our proposed algorithm are much bigger than the wavelet subbands they are convolved with. This uncommon procedure implies the padding, *i.e.*, the addition of many extra pixels to the edge of each wavelet subband, as illustrated in fig. 4.7. As, to the best of our knowledge, convolutions with filters that are significantly larger than the input have not been optimised yet in the existing software libraries, such computations may cause a severe speed reduction in the training process as well as the necessity of a substantial memory space. The alternative implementation of ΨDONet, described in section 4.1.2.2 , addresses these shortcomings.

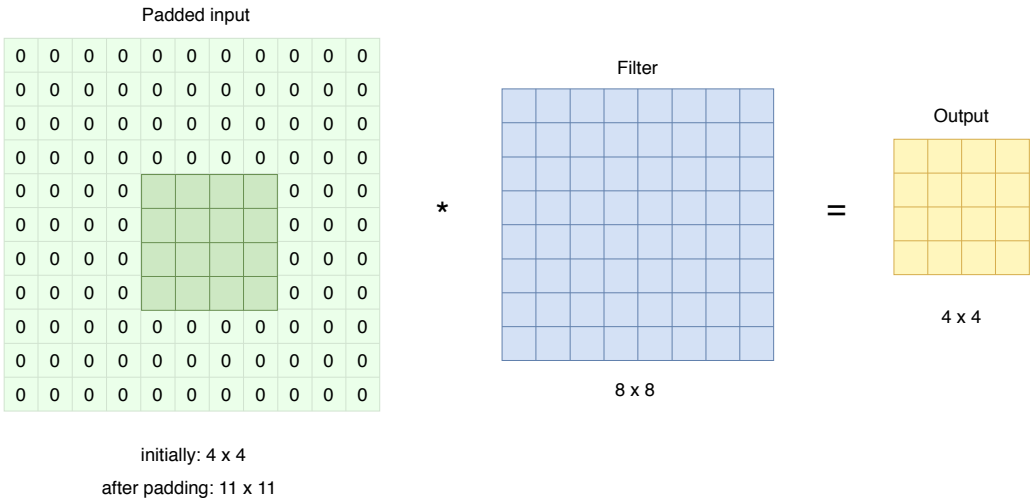


Figure 4.7: Convolving an $F \times F$ input with a $2F \times 2F$ filter, as it is needed in ΨDONet-F, implies the addition of $2F - 1$ external rows (resp. columns) of zeros to the input: such process is called *padding*. This figure illustrates the padding required if $F = 4$ (for the sake of simplicity, we consider only a single depth slice). The numerous zero pixels that have to be added when convolving with filters that are significantly larger than the input may bring about some speed reduction in the training process as well as the necessity of a substantial memory space.

4.1.2.2 ΨDONet-O

The main flaw of ΨDONet-F rests upon the use of the operator $\breve{\mathbf{K}}^{\tau}$, which implies numerous burdensome convolutions. This issue is worked around in ΨDONet-O, as $\breve{\mathbf{K}}^{\tau}$ is not involved anymore. Here, the back-projection operator is not approximated, meaning that $\mathbf{W}\mathbf{R}_{\mathbf{\Gamma}}^{*}\mathbf{R}_{\mathbf{\Gamma}}\mathbf{W}^{*}$ is indeed implemented as the succession of the inverse wavelet, Radon, inverse Radon and direct wavelet transforms applied to the iterate $\mathbf{w}^{(n)}$. As for the learnable part of the network, a trainable, convolutional operator $\Lambda_{\zeta_n}^{\tau}$ is added to each unfolded ISTA iteration, just as in ΨDONet-F. Conceptually, this second implementation of ΨDONet can still be interpreted as the splitting of the kernel operator $\mathbf{K}$ into two operators: the first one, $\mathbf{K}_0 := \mathbf{W}\mathbf{R}_{\mathbf{\Gamma}}^{*}\mathbf{R}_{\mathbf{\Gamma}}\mathbf{W}^{*}$, is the exact back-projection operator in the wavelet domain and stands for the known part of the model; the second operator, $\mathbf{K}_1 := \Lambda_{\zeta_n}^{\tau}$, whose architecture is the same as $\mathbf{K}$, provides the potential to improve the reconstruction process. Fig. 4.8 shows how this splitting can be thought of. We however insist on the fact that it only illustrates the theoretical concept underlying ΨDONet-O, and not its actual implementation as the fixed operator $\mathbf{K}_0$ does not involve convolutional filters anymore, but instead makes use of the classic transforms.

The second implementation of ΨDONet, named Operator-Based ΨDONet or ΨDONet-O, reads:

$$\mathbf{w}^{(n+1)} = \mathcal{S}_{\gamma_n}\left(\mathbf{w}^{(n)} + \alpha_n\left(\mathbf{W}\mathbf{R}_{\mathbf{\Gamma}}^{*}\mathbf{m} - \mathbf{W}\mathbf{R}_{\mathbf{\Gamma}}^{*}\mathbf{R}_{\mathbf{\Gamma}}\mathbf{W}^{*}\mathbf{w}^{(n)}\right) + \beta_n\Lambda_{\zeta_n}^{\tau}\mathbf{w}^{(n)}\right), \tag{4.5}$$

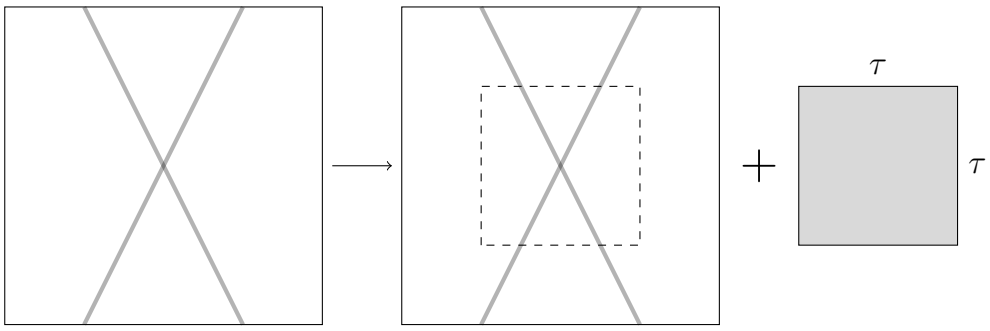


Figure 4.8: Conceptual illustration of the splitting of $\mathbf{K}$ in ΨDONet-O into two operators: $\mathbf{K}0 := \mathbf{W}\mathbf{R}_{\mathbf{\Gamma}}^{*}\mathbf{R}_{\mathbf{\Gamma}}\mathbf{W}^{*}$ and $\mathbf{K}_1 := \Lambda_{\zeta}^{\tau}$. Each filter of $\mathbf{K}$ can be thought of as the sum of a fixed filter, representing the known part of the model, and a trainable filter with dimensions $\tau \times \tau$, interpreted as an adjunct for improving the back-projection operator. It is crucial to note, however, that this figure is only intended to provide a theoretical comprehension of the concept underlying ΨDONet-O, but does not represent the actual implementation thereof, since the operator $\mathbf{K}_0$ does not involve convolutional filters, but instead makes use of the classic wavelet and Radon transforms.

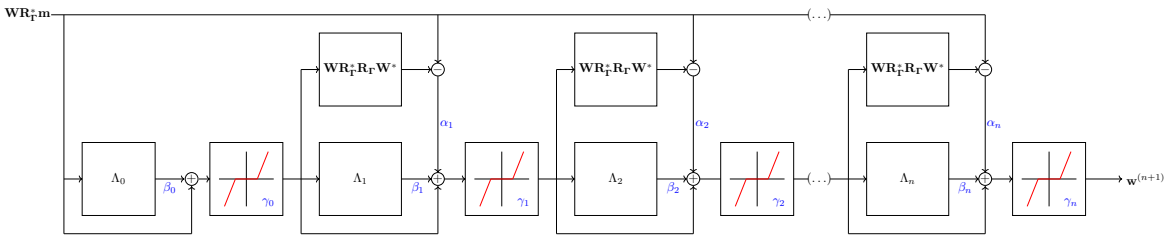
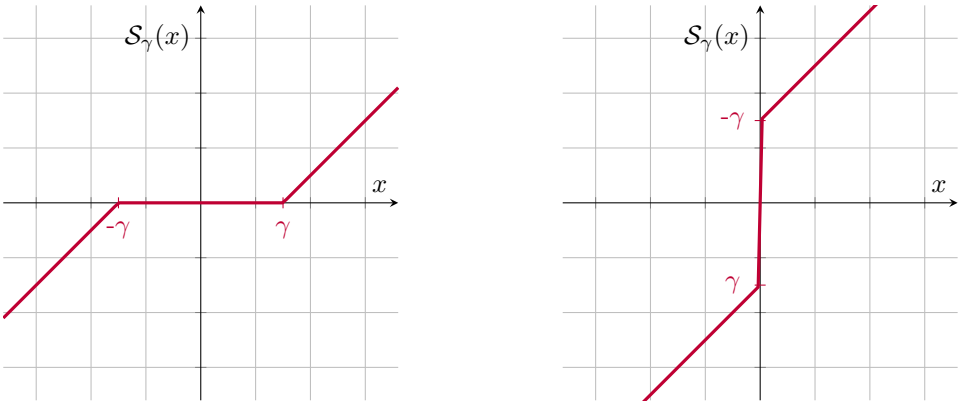


Figure 4.9: Block diagram of the proposed model (4.5). The contribution of the trainable operators $\Lambda_{\zeta_1}^{\tau}, \ldots, \Lambda_{\zeta_N}^{\tau}$ can be highlighted by comparing this architecture with that of standard ISTA, illustrated in fig. 3.1.

for $n = \{0, \ldots, N\}$, where the parameters to be learnt are $\{\gamma_0, \alpha_0, \beta_0, \zeta_0, \ldots, \gamma_N, \alpha_N, \beta_N, \zeta_N\}$. The operator $\Lambda_{\zeta_n}^{\tau}$ is initialised with random values, according to the Xavier initialisation presented in section 2.2.4.2. The block diagram of the algorithm is represented in fig. 4.9. For the special choice of $\gamma_n = \frac{\lambda}{L}, \alpha_n = \frac{1}{L}, \beta_n = 0$, for any $n = \{0, \ldots, N\}$, this model is exactly equivalent to standard ISTA. The only convolutions involved in this alternative implementation are the ones composing the CNN $\Lambda_{\zeta_n}^{\tau}$, whose filters can be chosen to be small enough to avoid any running time or storage issue. In that sense, ΨDONet-O offers an implementation numerically preferable to ΨDO-Net-F, while retaining the same properties on a theoretical level. Furthermore, such a model keeps offering a clear interpretation of its post-processing abilities since $\Lambda_{\zeta_n}^{\tau}$, on account of its architecture, can still be seen as an adjunct for improving the back-projection operator.



(a) Soft-thresholding function with non-negative parameter: $\gamma \geq 0$.

(b) Soft-thresholding function with negative parameter: $\gamma < 0$.

Figure 4.10: Plots of the soft-thresholding function as defined in section 4.1.2.3.

4.1.2.3 Note on soft-thresholding parameters

From a theoretical point of view, the parameters $\gamma_0, \ldots, \gamma_N$ in ΨDONet-F and ΨDONet-O have to be non-negative, as they represent the soft-thresholding parameters. In order to stick to the operator originally involved in standard ISTA, it is possible to enforce the positivity of the coefficient by replacing each γ_n by $10^{\tilde{\gamma}_n}$, where $\tilde{\gamma}_n$ becomes the actual trainable parameter. However, with the aim to allow a greater degree of freedom in the learning process, we propose to implement the operator $\mathcal{S}_{\gamma_n}$ in such a way that it is also interpretable for negative values of its parameter γ_n. In such a case, we define the function $\mathcal{S}_{\gamma_n<0}$ as the symmetric of the soft-thresholding curve w.r.t. $y = x$, while for non-negative values of γ_n, $\mathcal{S}_{\gamma_n\geq 0}$ is exactly equivalent to the soft-thresholding operator. Formally, $\mathcal{S}_{\gamma_n}$ becomes:

$$\text{For } \gamma_n \geq 0: \quad \mathcal{S}_{\gamma_n}(x) = \begin{cases} x - \gamma_n, & \text{if } x \geq \gamma_n \\ 0, & \text{if } |x| < \gamma_n \\ x + \gamma_n, & \text{if } x \leq -\gamma_n \end{cases}$$

$$\text{For } \gamma_n < 0: \quad \mathcal{S}_{\gamma_n}(x) = \begin{cases} x - \gamma_n, & \text{if } x \geq 0 \\ x + \gamma_n, & \text{if } x < 0 \end{cases}$$

The resulting functions $\mathcal{S}_{\gamma_n\geq 0}$ and $\mathcal{S}_{\gamma_n<0}$ are plotted in fig. 4.10. The two implementations ΨDONet-F and ΨDONet-O are tested with and without the positivity constraint on γ (see results in section 4.2.2).

4.1.3 Supervised Learning

We denominate f_θ^τ the N-layer CNN that, given $\mathbf{m}$, computes the final output $\mathbf{w}^{(N+1)}$ according to one of the two proposed architectures. We aim at learning the optimal high-dimensional vector $\theta = \{\gamma_0, \alpha_0, \beta_0, \zeta_0, \ldots, \gamma_N, \alpha_N, \beta_N, \zeta_N\}$ that ideally satisfies the relation:

$$f_\theta^\tau(\mathbf{m}) \approx \mathbf{W}\mathbf{u}^\dagger \tag{4.6}$$

More formally, we regard the tuple $\left(\mathbf{m}, \mathbf{u}^\dagger\right) \in \mathbb{R}^q \times \mathbb{R}^p$ as a random variable with a joint probability distribution Ξ, as detailed in section 3.3.2. Ideally, we would like to find a parameter vector θ^* minimising the expected risk:

$$\min_\theta \left(\mathbb{E}_{(\mathbf{m},\mathbf{u}^\dagger)\sim\Xi} \| f_\theta^\tau(\mathbf{m}) - \mathbf{W}\mathbf{u}^\dagger \|_2^2 \right) \tag{4.7}$$

Other loss functions, such as the weighted l_2-norm, where the wavelet coefficients are weighted depending on their scale, have been tested and lead to results similar to the non-weighted l_2-norm. For the sake of brevity, we will stick to the basic form of (4.7).

In practice, computing the expectation w.r.t. Ξ is not possible (see section 2.1.2.3). Instead, we are given a finite set of independent drawings $(\mathbf{m}_1, \mathbf{u}_1^\dagger), \ldots, (\mathbf{m}_S, \mathbf{u}_S^\dagger)$ and we consider the minimisation of the empirical risk:

$$\min_\theta \frac{1}{S} \sum_{i=1}^{S} \| f_\theta^\tau(\mathbf{m}_i) - \mathbf{W}\mathbf{u}_i^\dagger \|_2^2 \tag{4.8}$$

We propose to solve the optimisation problem (4.8) by means of minibatch gradient descent (see section 2.1.5.2), where the gradients are computed via backpropagation (see section 2.2.3). The final performance (*i.e.*, the generalisation ability) of the trained network f_θ^τ is evaluated on a separate set of independent drawings, the *test set*, that has not been used during the learning process.

The two implementations ΨDONet-F and ΨDONet-O have been trained and tested on different datasets of simulated data. The testing setups and the obtained numerical results are presented in section 4.2.

4.2 Experiments and results

In this section, we evaluate the performance of the proposed reconstruction schemes and compare it with that of standard ISTA.

4.2.1 Preliminaries

To begin with, we describe the considered experimental scenario, the implementation of the used operators and the training procedure.

4.2.1.1 Datasets

Different sets of simulated data have been used to test the proposed algorithm ΨDONet. They are divided into two categories, called *Ellipses* and *Apples*. Within each category, several setups have been considered, for different image resolutions and different missing wedges. Table 4.1 summarises the configurations of interest, as well as the number of images for the training, validation and testing sets in each category. The next paragraphs describe the composition of the different datasets as far as the groundtruth images are concerned.

dataset	*Ellipses*			*Apples*	
resolution	128×128	256×256	512×512	128×128	256×256
missing wedge	$60°$	$80°$	$80°$	$60°$	$80°$
train/val/test	10000/500/500			25319/3259/3773	

Table 4.1: Table summarising the experimental setups considered. The last row specifies the number of images contained in the training set, the validation set and the testing set respectively.

Ellipses The *Ellipses*-dataset consists of 11000 synthetic images of ellipses, generated with Matlab. In each image, the number, locations, sizes and intensity gradients of the ellipses are chosen randomly.

Apples The *Apples*-dataset contains 32351 tomographic scans that belong to the dataset of the Apples-CT challenge of the Code Sprint 2020 [43] They correspond to the slices of 52 different apples. The original 972×972 images have been resized to 128×128 and 256×256 images.

For all the aforementioned setups, the projections are simulated thanks to the Matlab's routine `radon`. The measurements are first computed at a higher resolution and then downsampled to their actual resolution, in such a way to avoid *inverse crime* [152]. We furthermore corrupt all the simulated sinograms with white Gaussian noise with zero mean and a variance equal to 1% of the sinogram maximum value.

4.2.1.2 Operators

For the implementation of the discrete limited angle operator $\mathbf{R}_\Gamma$ we use the `radon` routine of the Python package `scikit-image` [203], and the 2D parallel beam geometry of the Operator Discretization Library (ODL) library [1], which is based on the Astra toolbox [202]. The former is employed for generating the back-projections $\mathbf{W}\mathbf{R}_\Gamma^*\mathbf{m}$ provided as inputs to ΨDONet-F and ΨDONet-O, while the latter is used for the implementation of $\mathbf{W}\mathbf{R}_\Gamma^*\mathbf{R}_\Gamma\mathbf{W}^*$ in ΨDONet-O. The direct and inverse Radon transform operators are multiplied by a constant so that their norm is equal to one. This constant is computed through the Von Mises iteration [149], with a number of iterations equal to 10. Regarding the wavelet transform, we make use of the Python package `pywt` [132] to decompose the inputs before providing them to the network, and to recompose the outputs produced by the model. The wavelet transforms involved in the network itself, however, are implemented thanks to a modified version of the package `tf-wavelets` [89]. In all our experiments, we use Haar wavelets and consider the case $J_0 = J_1 + 2$, implying that the wavelet decomposition $\mathbf{W}\mathbf{u}$ has 10 subbands.

4.2.1.3 Network structure and training

The two schemes ΨDONet-F and ΨDONet-O have been implemented and tested on 128×128 images. As mentioned in section 4.1.2, however, the training of ΨDONet-F becomes rapidly impractical for 'big' images; thus, only the operator-based implementation has been tested on 256×256 and 512×512 images.

Regardless of the choice of ΨDONet's implementation, the architecture of the network is determined by a set of hyperparameters, which are to be manually tuned before starting the training process. The hyperparameter settings are summarised in table 4.2.

For example, the hyperparameter τ determines the size of the filters of the trainable operator $\Lambda^{\tau}_{\zeta_n}$ (see section 4.1.2). Note that according to theory, τ is supposed to be odd, however, in practice we prefer it to be even. This very slight modification has no effect on the results. Another important hyperparameter is the number N of layers in the network, which can be interpreted as the unrolling of the first N iterations of ISTA. In practice, with the purpose of reducing the number of parameters to be learnt, we choose to use N_s ($< N$) different sets of trainable parameters $\{\zeta_n, \gamma_n, \alpha_n, \beta_n\}$, each of which are used over n_r consecutive layers of the network. Therefore, the number of layers N is actually determined by the two hyperparameters N_s and n_r, such that $N = N_s \times n_r$.

The aforementioned hyperparameters concern the structure of the network. Other hyperparameters that do not define the network architecture but rather the course of the training process are also of great importance. They include *e.g.* the learning rate, the number of epochs and the minibatch size. We recall that the learning rate represents the steplength in the gradient descent algorithm employed to minimise the cost function (see section 2.1.5). The initial learning rates, reported in table 4.2, are scaled down by a factor of 0.9 at each epoch. The number of epochs is the number of times all the examples of the training set are shown to the network, and the minibatch size is the number of samples that are simultaneously provided to the network at each iteration of the training process. More information about these concepts are given in chapter 2.

The implementation and the training of our algorithms have been performed using Tensorflow with an Adam optimiser (see section 2.2.4.4). The trainings have been run on a NVIDIA Quadro P6000 GPU. The running times for each configuration are reported in table 4.2. All the codes are available at `https://github.com/megalinier/PsiDONet`.

4.2.1.4 Compared Methods

We compare the results achieved with the proposed architectures to the reconstructions provided by ISTA. In the implementation of the latter, we make use of the formula introduced in [52] and the number of iterations for ISTA is determined by the stopping criterion:

$$\|\mathbf{u}^{(n+1)} - \mathbf{u}^{(n)}\|_2^2 / \|\mathbf{u}^{(n)}\|_2^2 < tol \tag{4.9}$$

resolution	128×128				256×256		512×512
dataset	*Ellipses*		*Apples*		*Ellipses*	*Apples*	*Ellipses*
scheme	F	O	F	O	O		O
τ	32				64		64
N_{s}	40				30		25
n_{r}	3				4		4
learning rate	0.005	0.005	0.005	0.001	0.001		0.001
epochs	3		2		3	2	1
minibatch size	25				15		5
running time	12h	15h	20h	24h	34h	50h	146h

Table 4.2: Table summarising the network hyperparameters employed for the experimental tests of ΨDONet and the corresponding running times (in hours) required by the training processes. The 'scheme' row specifies which of the two implementations has been tested: the letter F stands for ΨDONet-F and the letter O stands for ΨDONet-O. In practice, the training of ΨDONet-F becomes rapidly impractical for 'big' images; this is the reason why only the operator-based implementation has been tested on 256×256 and 512×512 images. The hyperparameter τ defines the size of the $\tau \times \tau$ filters of the trainable operator $\Lambda^{\tau}_{\zeta_n}$ (see section 4.1.2) in both implementations. In all the setups, the network is built as a succession of N layers, which can be interpreted as unrolled iterations of ISTA. In order to reduce the number of parameters to be learnt, we choose to use N_s ($< N$) different sets of trainable parameters $\{\zeta_n, \gamma_n, \alpha_n, \beta_n\}$, each of which being used over n_r consecutive layers of the network. Thus, N_s and n_r are such that $N = N_s \times n_r$. The 'learning rate' row provides the values of the initial learning rate for each training, which are then scaled by a factor of 0.9 at each epoch.

resolution	128×128		256×256		512×512
dataset	*Ellipses*	*Apples*	*Ellipses*	*Apples*	*Ellipses*
λ	2.10^{-6}				1.10^{-6}
L	5				2
tol	2.10^{-4}				8.10^{-4}

Table 4.3: Table summarising the values of the regularisation parameter λ, the constant L and the tolerance *tol*, chosen when applying ISTA to each dataset under consideration.

The values of the regularisation parameter λ, the constant L and the tolerance *tol* chosen for each configuration are summarised in table 4.3.

Below, we give a list of the abbreviations henceforth used for the different recovery methods.

$\mathbf{u_{ista}}$ Standard ISTA reconstruction.

$\mathbf{u_{FBP}}$ Standard filtered back-projection with the 'ramp' filter of `scikit-image`.

$\mathbf{u^{+}_{\Psi do\text{-}F}}$ Solution provided by ΨDONet-F with positivity-constraint on the soft-thresholding parameter ($\gamma_n = 10^{\tilde{\gamma}_n}, \forall n$).

$\mathbf{u_{\Psi do\text{-}F}}$ Solution provided by ΨDONet-F without positivity-constraint on the soft-thresholding parameter.

$\mathbf{u^{+}_{\Psi do\text{-}O}}$ Solution provided by ΨDONet-O with positivity-constraint on the soft-thresholding parameter ($\gamma_n = 10^{\tilde{\gamma}_n}, \forall n$).

$\mathbf{u_{\Psi do\text{-}O}}$ Solution provided by ΨDONet-O without positivity-constraint on the soft-thresholding parameter.

4.2.1.5 Similarity Measures

For the assessment of image quality, we use the following quantitative measures (where $\mathbf{u}^\dagger$ denotes the reference image and $\mathbf{u}$ its reconstruction):

- the relative error (RE):
$$RE(\mathbf{u}^\dagger, \mathbf{u}) = \frac{\|\mathbf{u}^\dagger - \mathbf{u}\|_2}{\|\mathbf{u}^\dagger\|_2} \tag{4.10}$$

- the peak signal-to-noise ratio (PSNR):
$$PSNR(\mathbf{u}^\dagger, \mathbf{u}) = 10 \log_{10}\left(\frac{d^2}{MSE(\mathbf{u}^\dagger, \mathbf{u})}\right), \tag{4.11}$$
where d is the maximum possible pixel value of the image (1 in our case), and MSE is the mean squared error defined as:
$$MSE(\mathbf{u}^\dagger, \mathbf{u}) = \frac{1}{p}\sum_{i=1}^{p}(u_i^\dagger - u_i)^2 \tag{4.12}$$

- the structured similarity index (SSIM) [212]:
$$SSIM(\mathbf{u}^\dagger, \mathbf{u}) = \frac{(2\mu_u\mu_{u^\dagger} + c_1)(2\sigma_u\sigma_{u^\dagger} + c_2)(2\,\mathrm{cov}_{uu^\dagger} + c_3)}{(\mu_u^2 + \mu_{u^\dagger}^2 + c_1)(\sigma_u^2 + \sigma_{u^\dagger}^2 + c_2)(\sigma_u\sigma_{u^\dagger} + c_3)} \tag{4.13}$$
where $(\mu_{u^\dagger}, \sigma_{u^\dagger})$ and (μ_u, σ_u) are the mean and standard deviation of $\mathbf{u}^\dagger$ and $\mathbf{u}$ respectively, $\mathrm{cov}_{uu^\dagger}$ is the cross-variance of $\mathbf{u}^\dagger$ and $\mathbf{u}$ and c_1, c_2 and c_3 are constants. SSIM offers a measure of the perceived visual quality, based on how the human eye extracts structural information from an image. It is thus more discriminating with regards to artefacts than the mean square error for instance.

- the Haar wavelet-based perceptual similarity index (Haar-PSI), recently proposed in [170], which also offers to assess the perceptual similarity between two images w.r.t. a human viewer.

4.2.2 Numerical results

In the following, we report and discuss the results of our numerical experiments. For the sake of clarity, we will distinguish the different datasets by their category (*Ellipses*, *Apples*) and their resolution (128×128, 256×256 or 512×512). For example, the dataset of *Ellipses* composed by 128×128 images will be denoted by *Ellipses-128*. This name is sufficient to uniquely determine the dataset configuration (see table 4.1).

The average image quality measures of the reconstructed test images are reported in tables 4.4 to 4.8, for each setup under consideration. Furthermore, examples of reconstructions obtained with the different methods of interest are shown in figures 4.11 to 4.15. The measurements $\mathbf{m}$, or sinograms, are represented with black strips on their left and right sides, corresponding to the missing angular range. In addition to the obtained reconstructions, we also provide the visualisation of the absolute difference of each reconstructed image and its ground truth, in order to better evaluate the performance of the proposed method.

As it can be seen on the aforementioned figures, due to the large missing angle ($60°$ and $80°$), the FBP images are contaminated with contrast changes and streaking artefacts (see fig. 4.11c for instance). The latter are reminiscent of the bowtie-shaped artefacts illustrated in 4.3, and indeed stem directly from the very nature of the normal operator $\mathbf{R}_{\mathbf{\Gamma}}^{*}\mathbf{R}_{\mathbf{\Gamma}}$. Although ISTA offers better quality reconstructions (see *e.g.* fig. 4.11d), it still leads to streaking artefacts, and impurities due to the noise in the measurements can be noticed in the reconstructions. Besides, ISTA images are toned down, meaning that for the most part, the intensity of the pixels remain significantly lower than the expected values.

With the proposed implementations, whether with positivity constraint on the soft-thresholding parameter or without, it is possible to substantially reduce those artefacts and contrast issues. As it can be observed in figs. 4.11 to 4.15, the proposed methods lead to undeniably enhanced reconstructions, with a meaningful diminution of the relative error. As reported in tables 4.4 to 4.8, the ΨDONet reconstructions also show significantly better PSNR, SSIM and HaarPSI values when compared to ISTA and FBP. Furthermore, it can be remarked that the reconstructions achieved with the proposed methods are less corrupted by the measurement noise. However, in some cases, as it can be seen on fig. 4.14, the ΨDONet images exhibit a somewhat 'blocky' structure. This can be explained by the fact that the wavelet basis employed for the decomposition of the images is the Haar wavelet basis. Choosing a smoother kind of wavelets, such as the Daubechies wavelets with some number of vanishing moments, may soften this 'blocky' effect. Exploring such a possibility is left to future works.

Although the training of the operator-based models requires less computational resources than the filter-based ones, the two kinds of implementation provide similar reconstruction qualities (see figs. 4.11 and 4.14 and tables 4.4 and 4.5), and in that

Method	RE	PSNR	SSIM	HaarPSI
$\mathbf{u}_{\text{ista}}$	0.40	23.43	0.50	0.40
$\mathbf{u}_{\text{FBP}}$	0.52	21.31	0.45	0.36
$\mathbf{u}^{+}_{\Psi\text{do-F}}$	0.23	28.41	0.84	0.60
$\mathbf{u}_{\Psi\text{do-F}}$	**0.22**	**28.45**	**0.87**	**0.64**
$\mathbf{u}^{+}_{\Psi\text{do-O}}$	0.25	27.70	0.78	0.55
$\mathbf{u}_{\Psi\text{do-O}}$	0.24	28.04	0.84	0.57

Table 4.4: Comparison of reconstruction methods applied to the *Ellipses-128* testset.

Method	RE	PSNR	SSIM	HaarPSI
$\mathbf{u}_{\text{ista}}$	0.24	19.34	0.55	0.37
$\mathbf{u}_{\text{FBP}}$	0.34	16.32	0.42	0.31
$\mathbf{u}^{+}_{\Psi\text{do-F}}$	**0.07**	29.85	0.83	0.73
$\mathbf{u}_{\Psi\text{do-F}}$	**0.07**	**29.88**	**0.84**	**0.74**
$\mathbf{u}^{+}_{\Psi\text{do-O}}$	0.12	25.52	0.74	0.62
$\mathbf{u}_{\Psi\text{do-O}}$	**0.07**	29.83	0.83	0.73

Table 4.5: Comparison of reconstruction methods applied to the *Apples-128* testset.

sense, can be considered as result-wise equivalent. As for the positivity constraint on the soft-thresholding parameter, one can note that, in comparison with ΨDONet-F and ΨDONet-O respectively, ΨDONet-F+ and ΨDONet-O+ may produce slightly smoother reconstructions, with the potential presence of streaking artefacts. The latter, however, are greatly lessened when compared with the ones in the ISTA images (see fig. 4.12f or fig. 4.14i for example). In fact, the SSIM measures of the constrained model reconstructions are significantly greater than in the ISTA case, but always below the SSIM measures of their non-constrained alternative. The latter do a noteworthy job in removing the artefacts and sharpening the edges, as it can be seen *e.g.* in fig. 4.11j and fig. 4.11e. Overall, due to the quality of its reconstructions and the practicability of its training on relatively big images, ΨDONet-O appears as the best method over the four proposed.

For illustration purposes, we provide a visualisation of the learnt parameters α_n, β_n and γ_n for each proposed method when trained on the *Ellipses-128* dataset, in fig. 4.16 and fig. 4.17.

Method	RE	PSNR	SSIM	HaarPSI
$\mathbf{u}_{\text{ista}}$	0.47	23.36	0.48	0.38
$\mathbf{u}_{\text{FBP}}$	0.63	20.69	0.42	0.32
$\mathbf{u}^{+}_{\Psi\text{do-O}}$	0.33	26.33	0.70	0.43
$\mathbf{u}_{\Psi\text{do-O}}$	**0.26**	**28.30**	**0.83**	**0.48**

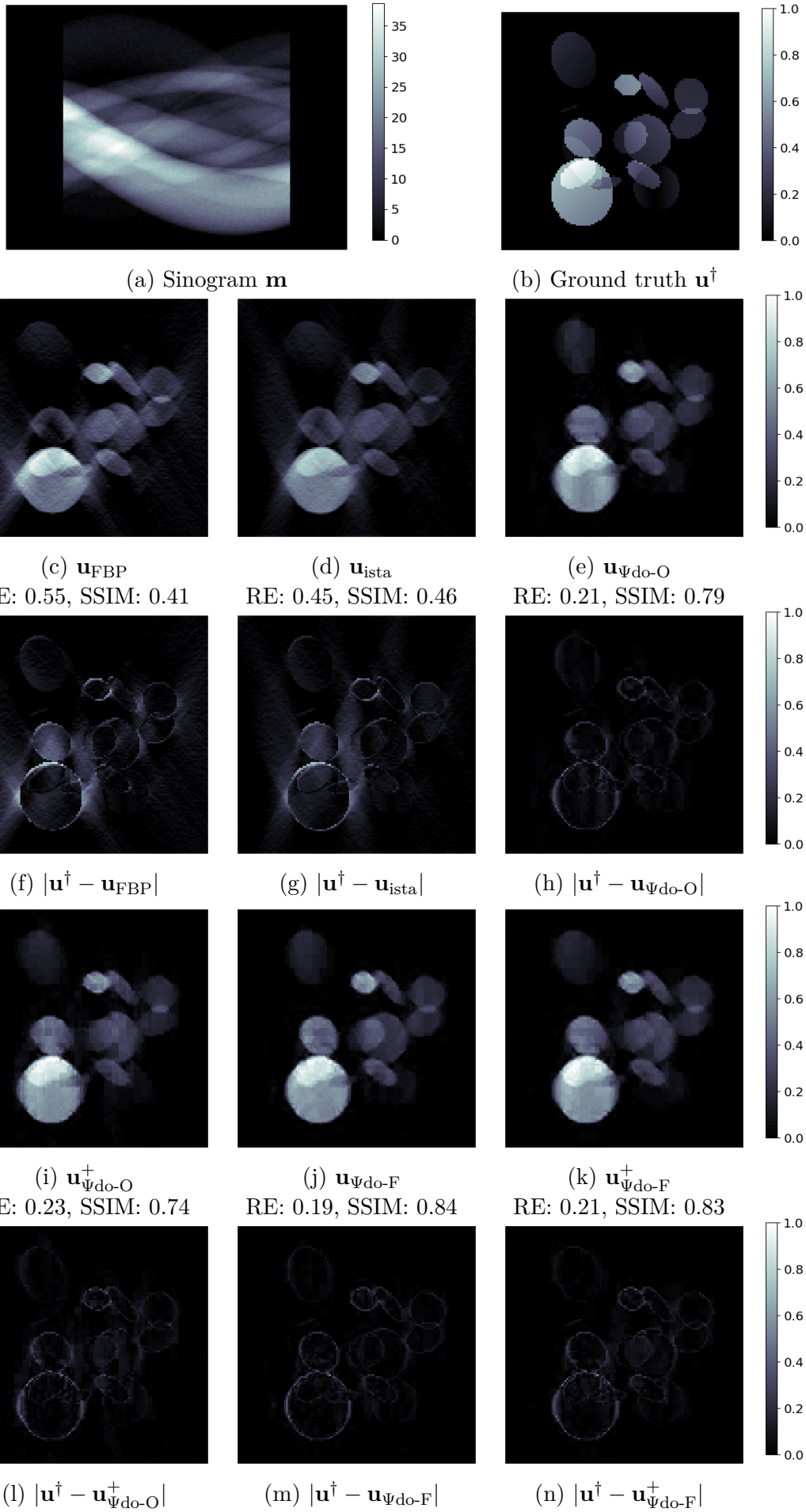
Table 4.6: Comparison of reconstruction methods applied to the *Ellipses-256* testset.

Method	RE	PSNR	SSIM	HaarPSI
$\mathbf{u}_{\text{ista}}$	0.36	18.64	0.45	0.31
$\mathbf{u}_{\text{FBP}}$	0.42	14.63	0.25	0.25
$\mathbf{u}^{+}_{\Psi\text{do-O}}$	0.10	27.32	0.76	0.58
$\mathbf{u}_{\Psi\text{do-O}}$	**0.08**	**29.68**	**0.82**	**0.65**

Table 4.7: Comparison of reconstruction methods applied to the *Apples-256* testset.

Method	RE	PSNR	SSIM	HaarPSI
$\mathbf{u}_{\text{ista}}$	0.50	23.67	0.51	0.38
$\mathbf{u}_{\text{FBP}}$	0.72	20.39	0.28	0.29
$\mathbf{u}^{+}_{\Psi\text{do-O}}$	0.34	26.99	0.75	**0.41**
$\mathbf{u}_{\Psi\text{do-O}}$	**0.29**	**28.33**	**0.84**	0.40

Table 4.8: Comparison of reconstruction methods applied to the dataset *Ellipses-512*.



(a) Sinogram $\mathbf{m}$

(b) Ground truth $\mathbf{u}^\dagger$

(c) $\mathbf{u}_{\text{FBP}}$
RE: 0.55, SSIM: 0.41

(d) $\mathbf{u}_{\text{ista}}$
RE: 0.45, SSIM: 0.46

(e) $\mathbf{u}_{\Psi\text{do-O}}$
RE: 0.21, SSIM: 0.79

(f) $|\mathbf{u}^\dagger - \mathbf{u}_{\text{FBP}}|$

(g) $|\mathbf{u}^\dagger - \mathbf{u}_{\text{ista}}|$

(h) $|\mathbf{u}^\dagger - \mathbf{u}_{\Psi\text{do-O}}|$

(i) $\mathbf{u}^+_{\Psi\text{do-O}}$
RE: 0.23, SSIM: 0.74

(j) $\mathbf{u}_{\Psi\text{do-F}}$
RE: 0.19, SSIM: 0.84

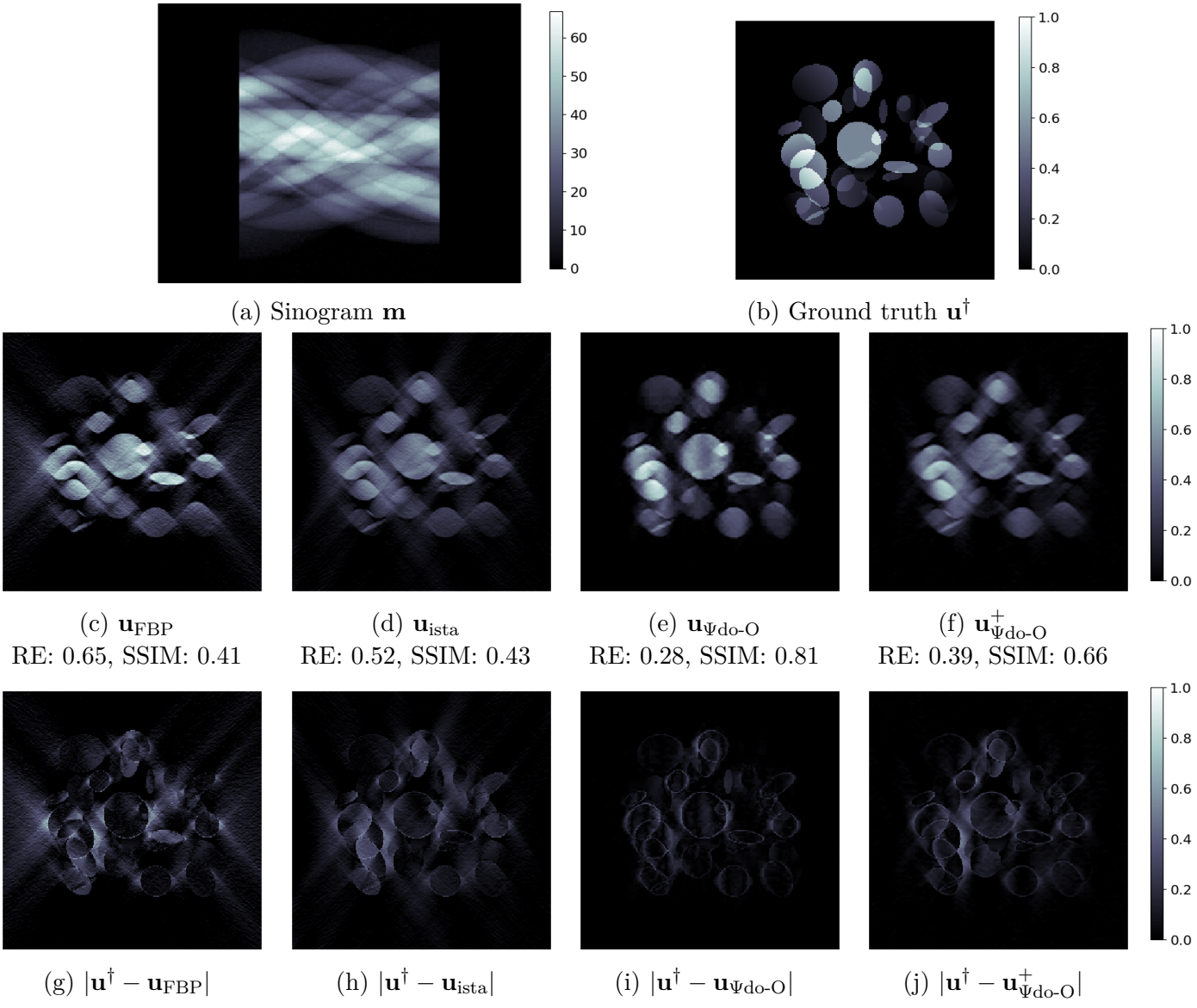
(k) $\mathbf{u}^+_{\Psi\text{do-F}}$
RE: 0.21, SSIM: 0.83

(l) $|\mathbf{u}^\dagger - \mathbf{u}^+_{\Psi\text{do-O}}|$

(m) $|\mathbf{u}^\dagger - \mathbf{u}_{\Psi\text{do-F}}|$

(n) $|\mathbf{u}^\dagger - \mathbf{u}^+_{\Psi\text{do-F}}|$

Figure 4.11: Sinogram and corresponding results for one test image from the *Ellipses-128* dataset.



(a) Sinogram $\mathbf{m}$ (b) Ground truth $\mathbf{u}^\dagger$

(c) $\mathbf{u}_{\text{FBP}}$
RE: 0.65, SSIM: 0.41

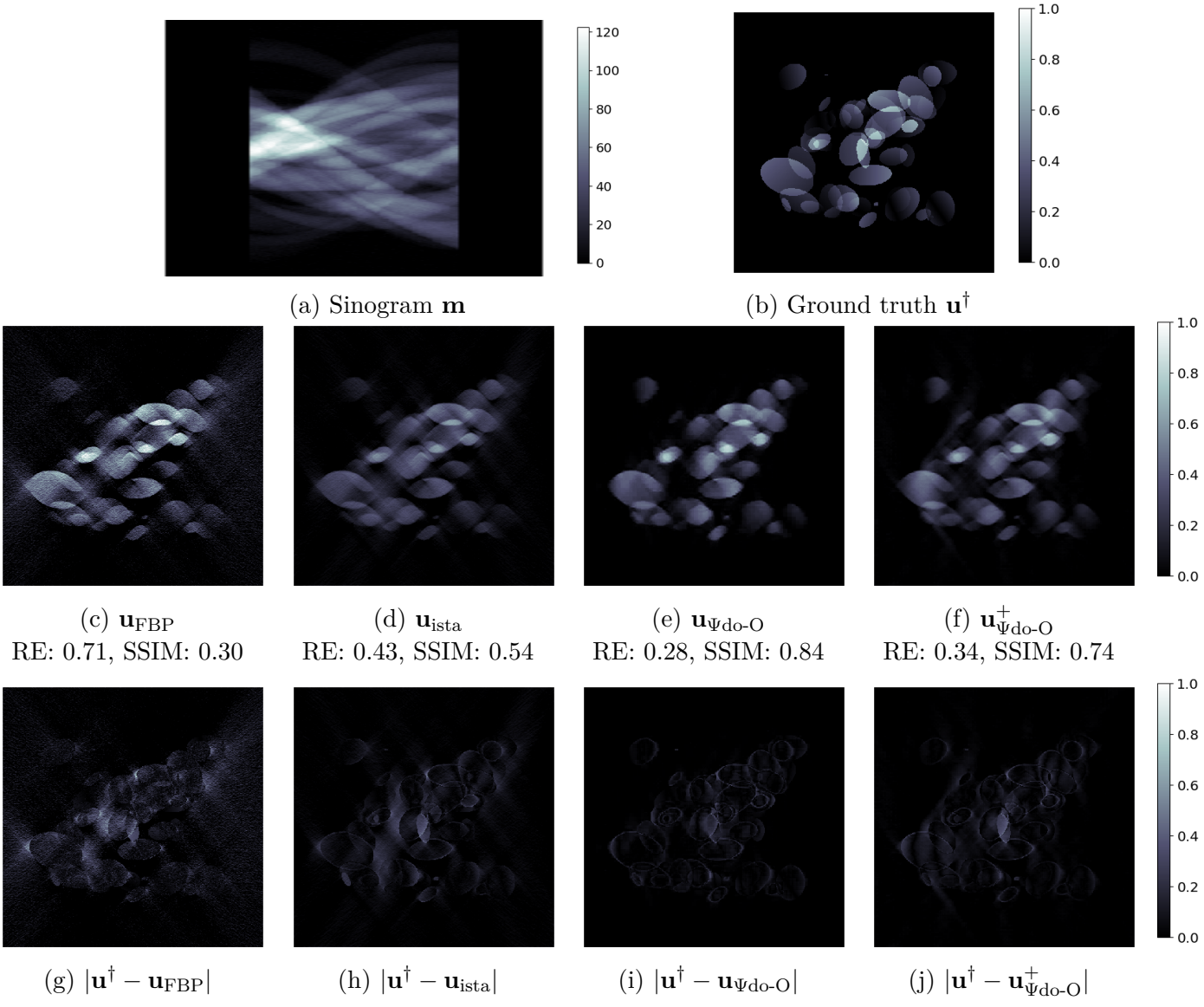
(d) $\mathbf{u}_{\text{ista}}$
RE: 0.52, SSIM: 0.43

(e) $\mathbf{u}_{\Psi\text{do-O}}$
RE: 0.28, SSIM: 0.81

(f) $\mathbf{u}^+_{\Psi\text{do-O}}$
RE: 0.39, SSIM: 0.66

(g) $|\mathbf{u}^\dagger - \mathbf{u}_{\text{FBP}}|$ (h) $|\mathbf{u}^\dagger - \mathbf{u}_{\text{ista}}|$ (i) $|\mathbf{u}^\dagger - \mathbf{u}_{\Psi\text{do-O}}|$ (j) $|\mathbf{u}^\dagger - \mathbf{u}^+_{\Psi\text{do-O}}|$

Figure 4.12: Sinogram and corresponding results for one test image from the *Ellipses-256* dataset.



(a) Sinogram $\mathbf{m}$ (b) Ground truth $\mathbf{u}^\dagger$

(c) $\mathbf{u}_{\text{FBP}}$
RE: 0.71, SSIM: 0.30

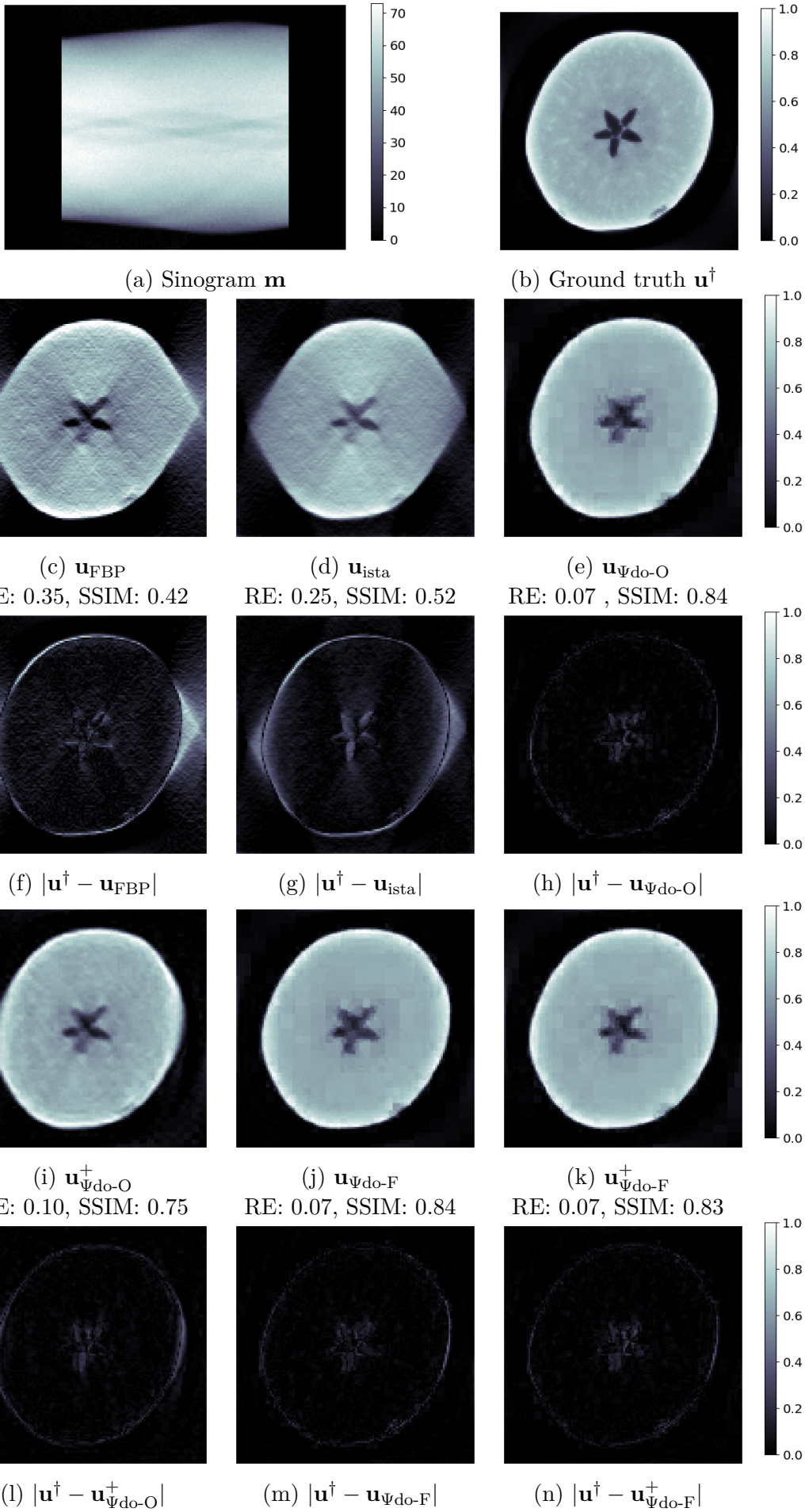
(d) $\mathbf{u}_{\text{ista}}$
RE: 0.43, SSIM: 0.54

(e) $\mathbf{u}_{\Psi\text{do-O}}$
RE: 0.28, SSIM: 0.84

(f) $\mathbf{u}^+_{\Psi\text{do-O}}$
RE: 0.34, SSIM: 0.74

(g) $|\mathbf{u}^\dagger - \mathbf{u}_{\text{FBP}}|$ (h) $|\mathbf{u}^\dagger - \mathbf{u}_{\text{ista}}|$ (i) $|\mathbf{u}^\dagger - \mathbf{u}_{\Psi\text{do-O}}|$ (j) $|\mathbf{u}^\dagger - \mathbf{u}^+_{\Psi\text{do-O}}|$

Figure 4.13: Sinogram and corresponding results for one test image from the *Ellipses-512* dataset.



(a) Sinogram $\mathbf{m}$ (b) Ground truth $\mathbf{u}^\dagger$

(c) $\mathbf{u}_{\text{FBP}}$
RE: 0.35, SSIM: 0.42

(d) $\mathbf{u}_{\text{ista}}$
RE: 0.25, SSIM: 0.52

(e) $\mathbf{u}_{\Psi\text{do-O}}$
RE: 0.07 , SSIM: 0.84

(f) $|\mathbf{u}^\dagger - \mathbf{u}_{\text{FBP}}|$ (g) $|\mathbf{u}^\dagger - \mathbf{u}_{\text{ista}}|$ (h) $|\mathbf{u}^\dagger - \mathbf{u}_{\Psi\text{do-O}}|$

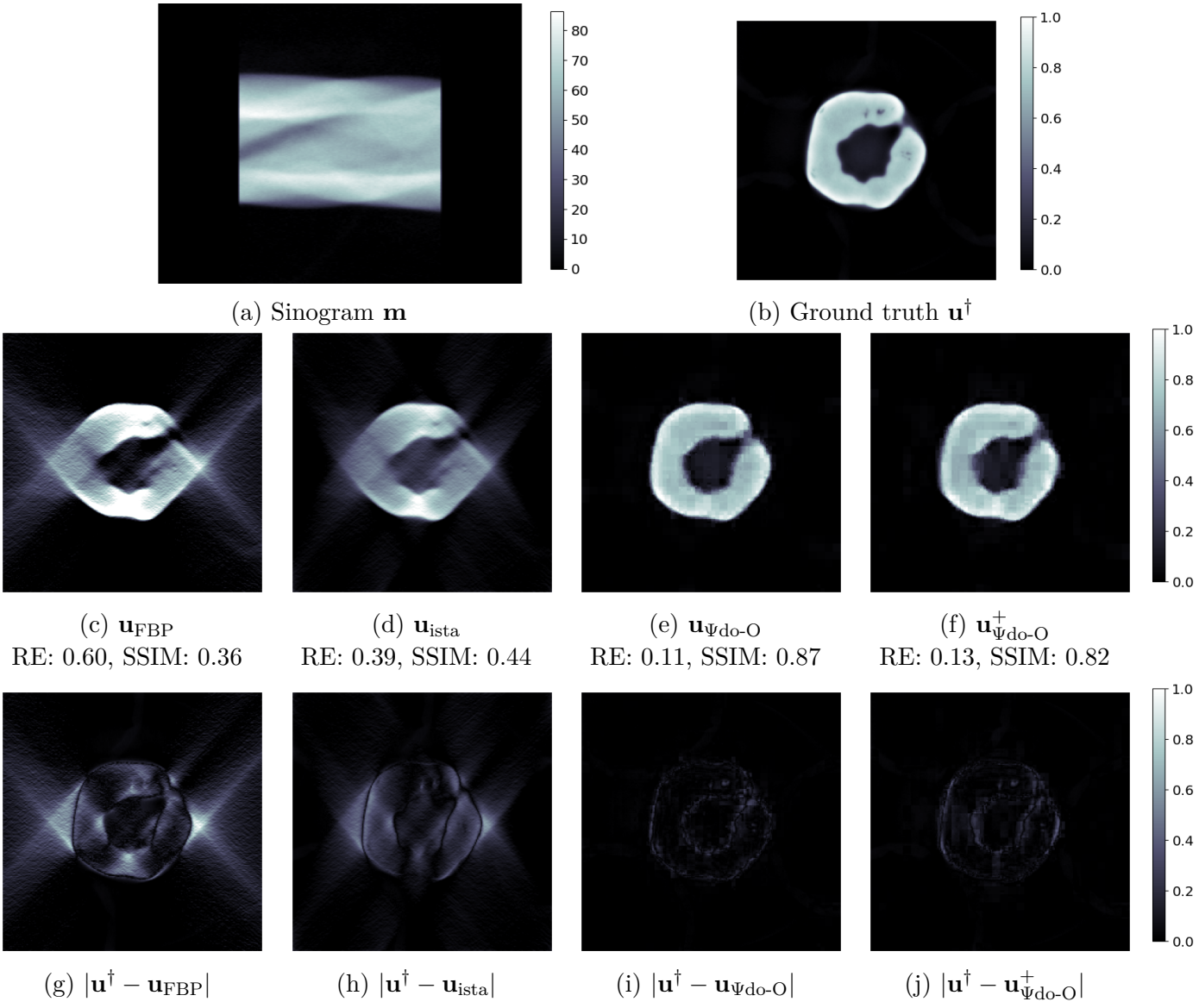
(i) $\mathbf{u}^+_{\Psi\text{do-O}}$
RE: 0.10, SSIM: 0.75

(j) $\mathbf{u}_{\Psi\text{do-F}}$
RE: 0.07, SSIM: 0.84

(k) $\mathbf{u}^+_{\Psi\text{do-F}}$
RE: 0.07, SSIM: 0.83

(l) $|\mathbf{u}^\dagger - \mathbf{u}^+_{\Psi\text{do-O}}|$ (m) $|\mathbf{u}^\dagger - \mathbf{u}_{\Psi\text{do-F}}|$ (n) $|\mathbf{u}^\dagger - \mathbf{u}^+_{\Psi\text{do-F}}|$

Figure 4.14: Sinogram and corresponding results for one test image from the *Apples-128* dataset.



(a) Sinogram $\mathbf{m}$ (b) Ground truth $\mathbf{u}^\dagger$

(c) $\mathbf{u}_{\text{FBP}}$
RE: 0.60, SSIM: 0.36

(d) $\mathbf{u}_{\text{ista}}$
RE: 0.39, SSIM: 0.44

(e) $\mathbf{u}_{\Psi\text{do-O}}$
RE: 0.11, SSIM: 0.87

(f) $\mathbf{u}^+_{\Psi\text{do-O}}$
RE: 0.13, SSIM: 0.82

(g) $|\mathbf{u}^\dagger - \mathbf{u}_{\text{FBP}}|$ (h) $|\mathbf{u}^\dagger - \mathbf{u}_{\text{ista}}|$ (i) $|\mathbf{u}^\dagger - \mathbf{u}_{\Psi\text{do-O}}|$ (j) $|\mathbf{u}^\dagger - \mathbf{u}^+_{\Psi\text{do-O}}|$

Figure 4.15: Sinogram and corresponding results for one test image from the *Apples-256* dataset.

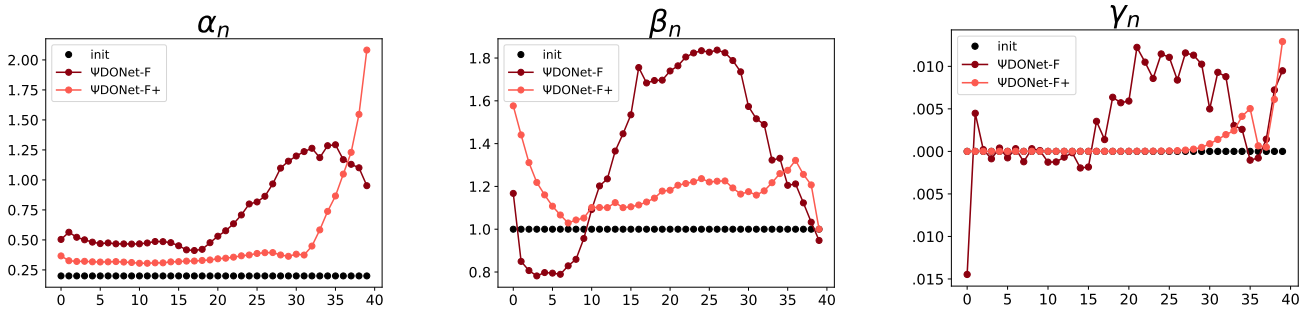
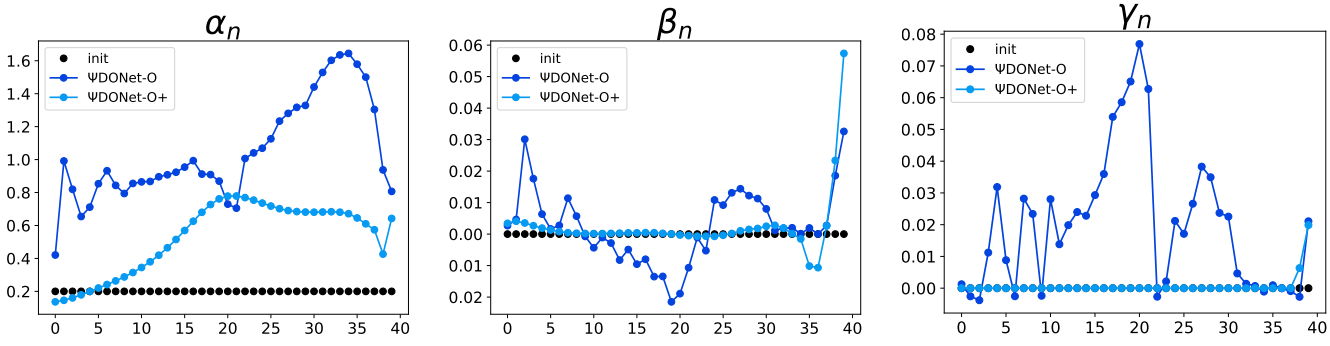


Figure 4.16: Plots of the parameters $(\alpha_n)_{0\leq n\leq N_s-1}$, $(\beta_n)_{0\leq n\leq N_s-1}$ and $(\gamma_n)_{0\leq n\leq N_s-1}$, learnt by the Filter-based implementation of ΨDONet, with and without positivity constraint on γ_n (the models are respectively called ΨDONet-F+ and ΨDONet-F), when trained on the *Ellipses-128* dataset. The y-axis represents the $N_s = 40$ different sets of trainable parameters. Each set is then to be employed over $n_r = 3$ consecutive layers, resulting in $N = N_s \times n_r = 40 \times 3 = 120$ unrolled iterations of ISTA. The black dots on the graph show the values that the parameters should take in order to imitate the exact behaviour of ISTA: $\gamma_n = \frac{\lambda}{L}, \alpha_n = \frac{1}{L}, \beta_n = 1$ (see section 4.1.2.1). Those values are chosen as initialisation for each parameter.



(a) (b) (c)

Figure 4.17: Plots of the parameters $(\alpha_n)_{0\leq n\leq N_s-1}$, $(\beta_n)_{0\leq n\leq N_s-1}$ and $(\gamma_n)_{0\leq n\leq N_s-1}$, learnt by the Operator-based implementation of ΨDONet, with and without positivity constraint on γ_n (the models are respectively called ΨDONet-O+ and ΨDONet-O), when trained on the *Ellipses-128* dataset. The y-axis represents the $N_s = 40$ different sets of trainable parameters. Each set is then to be employed over $n_r = 3$ consecutive layers, resulting in $N = N_s \times n_r = 40 \times 3 = 120$ unrolled iterations of ISTA. The black dots on the graph show the values that the parameters should take in order to imitate the exact behaviour of ISTA: $\gamma_n = \frac{\lambda}{L}, \alpha_n = \frac{1}{L}, \beta_n = 0$ (see section 4.1.2.2). Those values are chosen as initialisation for each parameter.

5 Conclusions

The research activity presented in this doctoral thesis was dedicated to the development of a novel CNN, named ΨDONet, designed to learn convolutional ΨDOs and FIOs in the broad context of linear inverse problems. In particular, we investigated such a new method through the practical case of limited-angle computed tomography. Due to the incompleteness of the acquired data, this X-based data imaging acquisition modality is a severely ill-posed inverse problem for which classical methods usually show poor performance.

The strength of the proposed approach rests upon the fact that it takes advantage of the powerful deep learning technology, whose fundamental principles are detailed in chapter 2, while benefiting from the trustworthiness of a more traditional, variational method: the well-known ISTA. ΨDONet, can thus incorporate both the physical understanding of the problem under consideration and a prior leveraged for the regularisation of the solution, while making use of a learning process to extract meaningful information from a set of training examples.

The theoretical principles underlying ΨDONet stem from the demonstration, provided in chapter 3 and numerically verified in chapter 4, that the unrolled iterations of ISTA can be interpreted as layers of a CNN. More interestingly, the downsampling, upsampling and convolution operations, typically defining a CNN, can be exactly specified by combining the convolutional nature of the normal operator under consideration and basic properties defining an orthogonal wavelet system. This analysis allows to gain understanding and interpretability of the results. We furthermore proved that, for a specific choice of the parameters involved, ΨDONet recovers ISTA, or a perturbation thereof. From the viewpoint of learning, such a characteristic represents a real asset inasmuch as those parameters offer a good initialisation point for the training process.

As a proof of concept, two different implementations of ΨDONet were proposed and tested on several sets of simulated data generated from a limited angle geometry. Both implementations produce equally good and noteworthy results, and show a significant improvement when compared to ISTA and FBP.

The work presented in this thesis opens up new prospects for CT, and more generally for a whole class of inverse problems arising from FIO or ΨDO. The main directions

that can be contemplated for future research include:

- The generalisation of the theoretical results presented in chapter 3, in light of recent contributions such as [54].
- The application of ΨDONet to other inverse problems with ΨDOs and FIOs, such as the geodesic X-ray transform [200], and its applications in seismic imaging, or synthetic-aperture radar (SAR) [159].
- Investigating the possibility of inserting more advanced features in the ΨDONet architecture, such as skip connections or extra residual blocks, with the aim of improving the reconstruction process while preserving full interpretability of the network.
- A further optimisation of the proposed implementations, in particular that of ΨDONet-O, in order to make its training on 'big' images (512×512 and bigger) faster.
- The application of hyperparameter optimisation methods to the ΨDONet architecture, in order to find the hyperparameters (including the number of layers, for instance) that lead to the best reconstructions.
- Exploring smoother types of wavelets and different cost functions $\mathcal{J}_S(\theta)$. For instance, one could consider to add a regularisation term to $\mathcal{J}_S(\theta)$, in order to provide the network with some additional prior knowledge.
- The application of ΨDONet to real data, where the visible wedges are usually smaller and sparser. In breast CT for example, the visible wedge covers only 20° and the number of sampled angles is equal to 11.

Appendix A: Microlocal analysis

In this appendix, we provide some background material concerning microlocal analysis, for the reader's convenience, with a special focus on ΨDOs and FIOs and their role in CT. We only give a glimpse of the general theory, recalling the basic definitions. Most of the material presented here comes from [181, 109, 127, 168] and the reader is referred to them for a deeper discussion on these topics.

Microlocal analysis originated in the 1950s and has since become a substantial mathematical theory whose fields of application include *inter alia* scattering theory, the study of chaotic system behaviour, inverse problems and general relativity. One might regard microlocal analysis as a kind of 'variable coefficient Fourier analysis' for solving variable coefficient PDEs. Another way to apprehend it is to consider microlocal analysis as a time-frequency approach for the study of functions or operators, and their singularities (*wave front sets*). In particular, it has allowed the definition and the analysis of the so-called ΨDOs and FIOs. The former were introduced by Kohn and Nirenberg [124] in 1965, whereas the latter were defined, together with wave front sets in their standard form, by Hörmander [108] in 1971.

In the case of tomography problems, microlocal analysis is used to understand the singularities that can be stably recovered, and helps to explain the presence of artefacts in certain image reconstruction methods. It can thus provide tools particularly useful in setups such as that of limited-angle CT.

Notation and Fourier formulas

In this appendix, we will use the following notation. Let $\mathbb{N}_0 = \{0, 1, 2, \dots\}$ be the set of natural numbers. Then $\mathbb{N}_0^n$ consists of n-tuples of the form $\alpha = (\alpha_1, \dots, \alpha_n)$ where the components α_j are nonnegative integers. We define $|\alpha| = \alpha_1 + \cdots + \alpha_n$, and $\xi^\alpha = \xi_1^{\alpha_1} \cdots \xi_n^{\alpha_n}$, for $\xi \in \mathbb{R}^n$. Regarding the partial derivatives, we will write:

$$\partial_j = \frac{\partial}{\partial x_j}, \quad D_j = \frac{1}{i}\partial_j, \quad D = \frac{1}{i}\nabla,$$

and we define:

$$D^\alpha = D_1^{\alpha_1} \cdots D_n^{\alpha_n}.$$

If $\Omega \subset \mathbb{R}^n$ is a bounded domain with C^∞ boundary, we denote $C^\infty(\bar{\Omega})$ the set of infinitely differentiable functions in Ω whose all derivatives extend continuously to

$\bar{\Omega}$. The space $C_c^\infty(\Omega)$ consists of C^∞ functions having compact support in Ω. In this appendix, all coefficients and boundaries are assumed to be C^∞ for ease of presentation.

We call $\mathcal{S}(\mathbb{R}^n)$ the *Schwartz space* of rapidly decreasing functions, that is the set of smooth functions that decrease (along with all their derivatives) faster than any power of $1/|x|$ at infinity.

We recall the following facts about the Fourier transform:

Definition A.1. *Given a function $u \in \mathcal{S}(\mathbb{R}^n)$. Its Fourier transform is the function:*

$$\hat{u}(\xi) := \mathcal{F}\{u\}(\xi) := \int_{\mathbb{R}^n} e^{-ix\cdot\xi} u(x)dx, \quad \forall \xi \in \mathbb{R}^n. \tag{A.1}$$

Definition A.2. *Given a function $\hat{u}$ integrable in $\mathbb{R}^n$. Its inverse Fourier transform is the function:*

$$\mathcal{F}^{-1}\{\hat{u}\}(x) := (2\pi)^{-n} \int_{\mathbb{R}^n} e^{ix\cdot\xi} \hat{u}(\xi)d\xi, \quad \forall x \in \mathbb{R}^n. \tag{A.2}$$

It is possible to recover a function $u \in \mathcal{S}(\mathbb{R}^n)$ from its Fourier transform $\hat{u}$ thanks to the *Fourier inversion formula*:

$$u = \mathcal{F}^{-1}\{\hat{u}\}. \tag{A.3}$$

Finally, we recall that, if $u \in \mathcal{S}(\mathbb{R}^n)$:

$$(\widehat{D_j u})(\xi) = \xi_j \hat{u}(\xi). \tag{A.4}$$

Roughly speaking, this means that the Fourier transform converts derivatives into polynomials.

Singularities and wavefront set

We first discuss the singular support of u, which consists of the points x_0 such that u is not a smooth function in any neighbourhood of x_0.

Definition A.3. *A function or a distribution u is said to be C^∞ near x_0 if there is a $\varphi \in C_c^\infty(\mathbb{R}^n)$ with $\varphi \equiv 1$ near x_0 such that φu is in $C^\infty(\mathbb{R}^n)$. We define the singular support as:*

$$\text{sing supp}(u) = \mathbb{R}^n \setminus \{x_0 \in \mathbb{R}^n;\ u \text{ is } C^\infty \text{ near } x_0\}. \tag{A.5}$$

Example A.1. Consider the square $S = [0,1]^2$ in $\mathbb{R}^2$. Let u be the characteristic function:

$$u(x,y) = \begin{cases} 1 & \text{if } (x,y) \in S; \\ 0 & \text{otherwise.} \end{cases}$$

Then sing supp(u) is the boundary of the square because that is where u is not smooth.

In some applications, knowing the singularities of u can already provide a useful information. In general, if u represents an image, then its singularities determine the 'sharp features' of the image. In the case of medical imaging, u may represent some internal properties of the body to image: its singularities can then help to locate the interfaces between different tissues.

A more refined notion of singularity is given by the *wavefront set*, which tells not only *where* the function u is singular (information already described by its singular support), but also *how* it is singular, by being more exact about the direction in which the singularity occurs.

Definition A.4. *A distribution u of $\mathbb{R}^n$ is said to be microlocally C^∞ near (x_0, ξ_0) if there exist $\varphi \in C_c^\infty(\mathbb{R}^n)$ with $\varphi \equiv 1$ near ξ_0 and an open cone Γ_{ξ_0}, such that:*

$$\forall N, \exists C_N > 0 \text{ s.t. } |\widehat{\varphi u}(\xi)| \leq C_N(1 + |\xi|)^{-N}, \quad \xi \in \Gamma_{\xi_0}. \tag{A.6}$$

The wavefront set WF(u) consists of the points (x_0, ξ_0) where u is not microlocally C^∞.

The wavefront set describes singularities more precisely than the singular support since it is always true that

$$\pi(WF(u)) = \text{sing supp}(u), \tag{A.7}$$

where $\pi : (x, \xi) \mapsto x$ is the projection to x-space.

Pseudodifferential operators

Wavefront sets and singularities have been particularly investigated for operators falling into the category of ΨDOs. To motivate the definition of ΨDOs, let us first consider a differential operator

$$P = \sum_{|\alpha| \leq m} p_\alpha(x) D^\alpha.$$

Then, the result Pu of P applied to a function $u \in \mathcal{S}(\mathbb{R}^n)$ can be expressed as:

$$\begin{aligned} Pu(x) &= P\left[\mathcal{F}^{-1}\{\hat{u}(\xi)\}\right] \\ &= \sum_{|\alpha| \leq m} p_\alpha(x) D^\alpha \left[(2\pi)^{-n} \int_{\mathbb{R}^n} e^{ix\cdot\xi} \hat{u}(\xi) d\xi\right] \\ &= (2\pi)^{-n} \int_{\mathbb{R}^n} e^{ix\cdot\xi} \left[\sum_{|\alpha| \leq m} p_\alpha(x) \xi^\alpha\right] \hat{u}(\xi) d\xi \\ &= (2\pi)^{-n} \int_{\mathbb{R}^n} e^{ix\cdot\xi} p(x, \xi) \hat{u}(\xi) d\xi \end{aligned}$$

where $p(x, \xi) = \sum_{|\alpha| \leq m} p_\alpha(x) \xi^\alpha$ is the *symbol* of P. In this case, $p(x, \xi)$ is polynomial in ξ.

Studying the operator P proves helpful in imaging problems for the following reasons:

1. Assuming that the symbol of P satisfies some estimates, one can describe precisely the action of P on the singularities of u.

2. If one has a procedure to invert or approximately invert the operator P by another operator Q that exhibits a similar integral representation as that of P, then the singularities of QPu are identical to those of u. In other words, this approximate inversion process offers a way to recover the singularities of u.

Under some assumptions, the theory of differential operators (such as P) can be generalised to a larger class of operators that are not necessarily defined by symbols polynomial in ξ: this broader category is that of ΨDOs.

Definition A.5. *A pseudodifferential operator (ΨDO) is an operator A of the form:*

$$Au(x) = (2\pi)^{-n} \int_{\mathbb{R}^n} e^{ix\cdot\xi} a(x,\xi)\hat{u}(\xi)d\xi \tag{A.8}$$

where $a(x,\xi)$ is a symbol with certain properties. The most standard symbol class $S^m = S^m_{1,0}(\mathbb{R}^n)$ is defined as follows:

Definition A.6. *The symbol class S^m consists of a functions $a \in C^\infty(\mathbb{R}^n \times \mathbb{R}^n)$ such that:*

$$\forall \alpha, \beta \in \mathbb{N}_0^n, \exists C_{\alpha,\beta} > 0, \ \ s.t. \qquad |\partial_x^\alpha \partial_\xi^\beta a(x,\xi)| \leq C_{\alpha,\beta}(1+|\xi|)^{m-|\beta|}, \quad \xi \in \mathbb{R}^n. \tag{A.9}$$

If $a \in S^m$, the corresponding ΨDO $A = \mathrm{Op}(a)$ is defined by eq. (A.8). We denote the set of ΨDOs corresponding to S^m by Ψ^m.

We also define the concept of *principal symbol*, which represents the part of the symbol containing the highest order derivatives:

Definition A.7. *The principal symbol of the operator $A = \mathrm{Op}(a)$ is:*

$$\sigma_{pr}(A) := \sum_{|\alpha|=m} a_\alpha(x)\xi^\alpha. \tag{A.10}$$

We say that A is *elliptic* if its principal symbol is nonvanishing for $\xi \neq 0$.

The class of ΨDOs includes *inter alia* differential operators as well as approximate inverses of elliptic operators. It also encompasses normal operators of common transforms, such as the X-ray or the Radon transforms (see below).

Amongst the important properties of a ΨDO, it is to be noted that, when applied to a function or a distribution, it never creates new singularities:

Theorem A.1. *Any $A \in \Psi^m$ has the pseudolocal property:*

$$\text{sing supp}(Au) \subset \text{sing supp}(u), \tag{A.11}$$

and the microlocal property:

$$WF(Au) \subset WF(u). \tag{A.12}$$

In particular, elliptic operators are those that completely preserve singularities:

Theorem A.2. *Let $A \in \Psi^m$ be elliptic. Then, for any u:*

$$\text{sing supp}(Au) = \text{sing supp}(u), \tag{A.13}$$

$$WF(Au) = WF(u). \tag{A.14}$$

Thus, any solution u of $Au = f$ is singular precisely at those points where f is singular.

Fourier integral operators

As previously mentioned, the class of ΨDOs includes approximate inverses of elliptic operators. In order to handle approximate inverses of hyperbolic and transports equations, it is required to consider a larger class of operators: the class of Fourier Integral Operators (FIOs).

An example of FIO is given by operators A of the form:

$$Au(x) = (2\pi)^{-n} \int_{\mathbb{R}^n} e^{i\varphi(x,\xi)} a(x,\xi)\hat{u}(\xi) d\xi \tag{A.15}$$

where $a(x,\xi)$ is a symbol (for instance in S^m), and $\varphi(x,\xi)$ is a real value function meeting some conditions. The class of FIOs includes *inter alia* ΨDOs (for which $\varphi(x,\xi) = x \cdot \xi$) as well as approximate inverses of hyperbolic and transport operators.

An important property of FIOs is that, unlike ΨDOs, they can move singularities. In general, any FIO has an associated *canonical relation* that describes how the FIO affects the singularities. The canonical relation of the FIO A in eq. (A.15) is:

$$C = \{(x, \xi, \nabla_x \varphi(x,\xi), \nabla_\xi \varphi(x,\xi)) \mid (x,\xi) \in T^*\mathbb{R}^n \setminus 0\}, \tag{A.16}$$

where $T^*\mathbb{R}^n \setminus 0 := \{(x,\xi) \mid x, \xi \in \mathbb{R}^n, \xi \neq 0\}$. Then, A moves the singularities according to the rule:

$$WF(Au) \subset C(WF(u)), \tag{A.17}$$

where

$$C(WF(u)) := \{(x,\xi) \mid (x,\xi,y,\eta) \in C \text{ for some } (y,\eta) \in WF(u)\}. \tag{A.18}$$

The case of the Radon transform

In this section, we provide some useful microlocal analysis results about the Radon transform in the plane, whose definition is:

Definition A.8. *If $u \in C_c^\infty(\mathbb{R}^n)$, then the Radon transform of u is the function:*

$$Ru(s,\omega) := \int_{-\infty}^{+\infty} u(s\omega^\perp + t\omega)\, dt, \qquad s \in \mathbb{R}, \omega, \omega^\perp \in S^1, \tag{A.19}$$

where $\omega^\perp$ denotes the vector in the unit sphere S^1 obtained by rotating ω counterclockwise by $90°$.

If we call $(\widetilde{Ru})(\cdot,\omega)$ the Fourier transform of Ru w.r.t. s, then the Fourier slice theorem establishes a relationship between the Radon transform Ru and the Fourier transform $\hat{u}$, namely:

$$(\widetilde{Ru})(\sigma,\omega) = \hat{u}(\sigma\omega^{\perp}). \tag{A.20}$$

This result gives the proof of injectivity of the Radon transform: if $u \in C_c^{\infty}(\mathbb{R}^2)$ is such that $Rf \equiv 0$, then $\hat{u} \equiv 0$ and consequently $f \equiv 0$.

We now define the *adjoint* of the Radon transform:

Definition A.9. *The adjoint $R^* : C^{\infty}(\mathbb{R}\times S^1) \to C^{\infty}(\mathbb{R})$ of the Radon transform is such that:*

$$R^*h(y) := \int_{S^1} h(y\cdot\omega^{\perp},\omega)d\omega. \tag{A.21}$$

The following result shows that the normal operator R^*R is a classical ΨDO of order -1 in $\mathbb{R}^2$, and gives an inversion formula.

Theorem A.3. *(Normal operator) One has that:*

$$R^*R = \mathcal{F}^{-1}\left\{\frac{4\pi}{|\xi|}\mathcal{F}(\cdot)\right\}, \tag{A.22}$$

and u can be recovered from Ru by the formula:

$$u = \frac{1}{4\pi}|D|R^*Ru, \tag{A.23}$$

where $|D|u := \mathcal{F}^{-1}\{|\xi|\hat{u}(\xi)\}$.

The proof of this theorem is based on computing $\langle Ru, Rg\rangle_{L^2(\mathbb{R}\times S)}$ using the Parseval identity, Fourier slice theorem, symmetry and polar coordinates. The symbol of R^*R is equal to $\frac{2}{|\xi|}$ and is homogeneous and nonwhere zero. Thus, R^*R is a classical elliptic ΨDO. Furthermore, since the symbol is homogeneous of degree -1, R^*R is a ΨDO of degree -1.

The Filtered Backprojection (FBP), based on a similar inversion formula, provides an efficient way to recover u where the measurements have been acquired on a full-range angle and with relatively small noise. However, if one is mainly interested in the singularities (sharp features) of the image, it is possible to use an even simpler method that only consists i applying the operator R^* to the data Ru. This method is called *backprojection*. Since R^*R is an elliptic ΨDO, Theorem A.2 guarantees that the singularities are recovered:

$$\text{sing supp}(R^*Ru) = \text{sing supp}(u) \tag{A.24}$$

Moreover, as R^*R is a ΨDO of order -1, hence smoothing operator of order 1, R^*Ru provides a slightly blurred version of u where the main singularities are still visible.

In the case where the measurements are acquired over a limited angular range $[-\Gamma,\Gamma]$, the Radon transform $R_{\Gamma}^*R_{\Gamma}$ is not a ΨDO anymore, but still belongs to the class of FIOs. Thus, there exists a precise relationship between the singularities of u

and that of $R_\Gamma^* R_\Gamma u$. We do not spell out this relationship in this appendix, but mention some of its consequences.

Although $u \in C_c^\infty(\mathbb{R}^2)$ is uniquely determined by limited angle data (Fourier slice and Paley-Wiener theorems), the inverse problem is very unstable. The concept of *visibility* enables to distinguish the singularities that can be stably recovered:

Definition A.10. *Let $\mathcal{A}$ represent the integral lines to which the Radon transform R_Γ is limited (that is $\mathcal{A}$ is such that $R_\Gamma u = Ru|_{\mathcal{A}}$). Then, a singularity at (x_0, ξ_0) is called visible from $\mathcal{A}$ if the line through x_0 in direction $\xi_0^\perp$ is in $\mathcal{A}$.*

One has the following dichotomy:

- If (x_0, ξ_0) is visible from $\mathcal{A}$, then from the singularities of $Ru|_{\mathcal{A}}$, one can determine whether or not $(x_0, \xi_0) \in WF(u)$. If $Ru|_{\mathcal{A}}$ uniquely determines u, one expects the reconstruction of visible singularities to be stable.

- If (x_0, ξ_0) is not visible from $\mathcal{A}$, then this singularity is smoothed out in the measurement $Ru|_{\mathcal{A}}$. Even if $Ru|_{\mathcal{A}}$ uniquely determines u, the inversion is not Lipschitz stable in any Sobolev norms.

Appendix B: Wavelet theory

In this appendix, we recall the general concepts underlying the wavelet theory, both in the continuous and discrete setups.For the sake of simplicity, we limit this presentation to the 1-D case. The material presented here mainly comes from [143, 56]. The reader is invited to consult them, as well as the references therein, for a fuller presentation of this topics.

Wavelets are defined as a family of functions constructed by using translation and dilation of a single function. Since the 1980s, they have been more and more studied and their range of application has not stopped to extend: signal and image processing, sampling theory, differential equations, turbulence, statistics, computer graphics, quality control, finance and economics, neural networks, astrophysics, geophysics, quantum mechanics, medicine, neuroscience, and chemistry.

The Continuous Wavelet Transform

We first give the formal definition of a wavelet.

Definition B.1. *A wavelet is a function $\psi \in L^2(\mathbb{R})$ which satisfies the condition:*

$$C_\psi \equiv \int_{-\infty}^{\infty} \frac{|\hat{\psi}(\xi)|^2}{|\xi|} d\xi < \infty, \tag{B.25}$$

where $\hat{\psi}(\xi)$ is the Fourier transform of $\psi(\xi)$ (see definition of Fourier transform in eq. (A.1)*).*

Example B.1. (The Haar wavelet). The Haar wavelet, introduced in 1910 by A. Haar [85], is the first known wavelet. It is defined as an odd rectangular pulse pair, an as such is the simplest orthonormal wavelet with compact support:

$$\psi(t) = \begin{cases} 1 & \text{if } 0 \leq t < \frac{1}{2}, \\ -1 & \text{if } \frac{1}{2} \leq t < 1, \\ 0 & \text{otherwise.} \end{cases} \tag{B.26}$$

It can be shown that the Haar wavelet does meet condition (B.25).

Condition (B.25), called *admissibility condition*, guarantees the existence of the inversion formula for the continuous wavelet transform. Further properties may also prove useful in certain applications. For example, ψ may be required to have a certain number of vanishing moments, which represent the regularity of the wavelet functions and ability of a wavelet transform to capture localised information. A wavelet $\psi(t)$ has n-vanishing moments if it satisfies:

$$\int_{-\infty}^{\infty} t^k \psi(t) \mathrm{d}t = 0, \quad k = 0, 1, \ldots, n. \tag{B.27}$$

Or equivalently:

$$\left[\frac{d^k \hat{\psi}(\xi)}{d\xi^k} \right]_{\xi=0} = 0, \quad k = 0, 1, \ldots, n. \tag{B.28}$$

Wavelets can be seen as a family of functions built from translation and dilation of a single function ψ, the *mother function*. Formally, they are defined by:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{|a|}} \psi \left(\frac{t-a}{b} \right), \quad a, b \in \mathbb{R}, \quad a \neq 0, \tag{B.29}$$

where a is the *scaling parameter* measuring the degree of compression, or scale, and b is the translation parameter, determining the time location of the wavelet. In other words, as scale parameter a varies, wavelet $\psi_{a,b}(t)$ covers different frequency ranges. As for the the translation parameter b, it defines the point around which the wavelet is centred.

Definition B.2. *If $\psi \in L^2(\mathbb{R})$ and $\psi_{a,b}(t)$ is given by eq.* (B.29)*, then the integral transformation W_ψ defined on $L^2(\mathbb{R})$ by:*

$$(W_\psi f)(a, b) = \langle f, \psi_{a,b} \rangle = \int_{-\infty}^{\infty} f(t) \overline{\psi_{a,b}(t)} dt \tag{B.30}$$

is called a continuous wavelet transform of f(t).

Continuous wavelet transforms benefit from the following properties:

Theorem B.1. *If ψ and ϕ are wavelets, and f and g are functions of $L^2(\mathbb{R})$, then:*

1. *Linearity:* $W_\psi(\alpha f + \beta g)(a, b) = \alpha(W_\psi f)(a, b) + \beta(W_\psi g)(a, b), \quad \alpha, \beta \in \mathbb{R},$
2. *Translation:* $(W_\psi(T_c f))(a, b) = (W_\psi f)(a, b - c),$
3. *Dilation:* $(W_\psi(D_c f))(a, b) = \frac{1}{\sqrt{c}} (W_\psi f) \left(\frac{a}{c}, \frac{b}{c} \right), \quad c > 0,$
4. *Symmetry:* $(W_\psi f)(a, b) = \overline{(W_f \psi) \left(\frac{1}{a}, -\frac{b}{a} \right)}, \quad a \neq 0\ ,$
5. *Antilinearity:* $(W_{\alpha\psi+\beta\phi} f)(a, b) = \bar{\alpha}(W_\psi f)(a, b) + \bar{\beta}(W_\phi f)(a, b),$
6. $(W_{T_c \psi} f)(a, b) = (W_\psi f)(a, b + ca),$

7. $(W_{D_c\psi})(a,b) = \frac{1}{\sqrt{c}}(W_\psi f)(ac,b), \quad c>0.$

where T_c is the translation operator: $T_c f(t) = f(ta-c)$, and D_c is the dilation operator: $D_c(t) = \frac{1}{\sqrt{|c|}} f(\frac{t}{c})$.

Theorem B.2. *If $f \in L^2(\mathbb{R})$, then f can be reconstructed by the inversion formula:*

$$f(t) = \frac{1}{C_\psi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (W_\psi f)(a,b) \psi_{a,b}(t) \frac{dbda}{a^2}, \tag{B.31}$$

where the inequality holds almost everywhere.

The Discrete Wavelet Transform

In many applications, and in particular in signal processing, data are represented by a finite number of values. It is necessary, therefore, to consider discrete versions of the continuous wavelet transform (B.30). For convenience in the discretisation, and without loss of generality, we restrict a to positive values, and we choose it such that $a = a_0^m$, where $m \in \mathbb{Z}$ and the dilation step $a_0 \neq 1$ is fixed. Furthermore, we discretise b such that $b = nb_0a_0^m$, where $n \in \mathbb{Z}$ and the positive constant b_0 is fixed. With such choices of a and b, the continuous family of wavelets $\psi_{a,b}$ as defined in (B.29) becomes:

$$\psi_{m,n}(t) = a_0^{-m/2} \psi\left(a_0^{-m}t - nb_0\right), \tag{B.32}$$

Then, the Discrete Wavelet Transform (DWT) of a function f is defined by:

$$(W_\psi f) = \langle f, \psi_{m,n} \rangle = a_0^{-m/2} \int_{-\infty}^{\infty} f(t) \overline{\psi}(a_0^{-m}t - nb_0) \mathrm{d}t, \tag{B.33}$$

where both f and ψ are continuous and $\psi_{0,0}(t) = \psi(t)$. Thus, the DWT represents a function by a countable set of wavelet coefficients. If the set $\{\psi_{m,n} \mid m,n \in \mathbb{Z}\}$ defined by (B.32) is complete in $L^2(\mathbb{R})$ for some choice of ψ, a_0 and b_0, then the set is an *affine wavelet* and any function $f \in L^2(\mathbb{R})$ can be expressed as the superposition:

$$f(t) = \sum_{m\in\mathbb{Z}} \sum_{n\in\mathbb{Z}} \langle f, \psi_{m,n} \rangle \psi_{m,n}(t). \tag{B.34}$$

For computational efficiency, $a_0 = \frac{1}{2}$ and $b_0 = 1$ are typically employed so that the results lead to a binary dilation of 2^m and a dyadic translation of $n2^{-m}$. Therefore, a common sampling method is:

$$\psi_{m,n}(t) = 2^{m/2} \psi\left(2^m t - n\right), \tag{B.35}$$

Often, it may be useful to build an orthogonal family $\{\psi_{m,n} \mid m,n \in \mathbb{Z}\}$, that is:

$$\langle \psi_{m,n}, \psi_{k,l} \rangle = \int_{-\infty}^{\infty} \psi_{m,n}(t) \psi_{k,l}(t) \mathrm{d}t = \delta_{m,k}\delta_{n,l}, \quad m,n,k,l \in \mathbb{Z}. \tag{B.36}$$

Definition B.3. *A wavelet $\psi \in L^2(\mathbb{R})$ is called orthonormal if the family of functions $\psi_{m,n}$ generated from ψ given by* (B.35) *is orthonormal.*

Appendix C: Further proofs

Proof of Proposition 3.2

Proof. According to [66, Section 3], the following variational source condition is satisfied by every $w \in \ell^1(\mathbb{N})$:

$$\beta\|w - w^\dagger\|_{\ell^1} \leq \|w\|_{\ell^1} - \|w^\dagger\|_{\ell^1} + C\|AW^*w - AW^*w^\dagger\|_Y. \tag{C.37}$$

We aim at applying it to $w = w_{\delta,p,q} \in W_p \subset \ell^1(\mathbb{N})$. First consider the term $\|w\|_{\ell^1} - \|w^\dagger\|_{\ell^1}$ in the right-hand side. Since $w_{\delta,p,q}$ is a solution of (3.6),

$$\begin{aligned}\lambda\|w_{\delta,p,q}\|_{\ell^1} &= \left(\|A_{p,q}W^*w_{\delta,p,q} - \mathbb{P}_q m\|_Y^2 + \lambda\|w_{\delta,p,q}\|_{\ell^1}\right) - \|A_{p,q}W^*w_{\delta,p,q} - \mathbb{P}_q m\|_Y^2 \\ &\leq \|A_{p,q}W^*\mathbb{P}_p w^\dagger - \mathbb{P}_q m\|_Y^2 + \lambda\|\mathbb{P}_p w^\dagger\|_{\ell^1} - \|A_{p,q}W^*w_{\delta,p,q} - \mathbb{P}_q m\|_Y^2,\end{aligned}$$

whence

$$\|w_{\delta,p,q}\|_{\ell^1} - \|w^\dagger\|_{\ell^1} \leq \frac{1}{\lambda}\|A_{p,q}W^*\mathbb{P}_p w^\dagger - \mathbb{P}_q m\|_Y^2 - \frac{1}{\lambda}\|A_{p,q}W^*w_{\delta,p,q} - \mathbb{P}_q m\|_Y^2.$$

We can easily check that $A_{p,q}W^*\mathbb{P}_p = \mathbb{P}_q AW^*\mathbb{P}_p$; then, since $\|\mathbb{P}_q\|_{Y\to Y} \leq 1$, denoting by $Q = \|A_{p,q}W^*w_{\delta,p,q} - \mathbb{P}_q m\|_Y$, we have

$$\begin{aligned}\|w_{\delta,p,q}\|_{\ell^1} - \|w^\dagger\|_{\ell^1} &\leq \frac{1}{\lambda}\|AW^*\mathbb{P}_p w^\dagger - m\|_Y^2 - \frac{1}{\lambda}Q^2 \\ &\leq \frac{1}{\lambda}\|AW^*(\mathbb{P}_p w^\dagger - w^\dagger)\|_Y^2 + \frac{1}{\lambda}\|AW^*w^\dagger - m\|_Y^2 - \frac{1}{\lambda}Q^2.\end{aligned}$$

In conclusion,

$$\|w_{\delta,p,q}\|_{\ell^1} - \|w^\dagger\|_{\ell^1} \leq \frac{1}{\lambda}\|A\|^2\|w^\dagger - \mathbb{P}_p w^\dagger\|_{\ell^2}^2 + \frac{1}{\lambda}\delta^2 - \frac{1}{\lambda}Q^2. \tag{C.38}$$

The second term in the right-hand side of (C.37), instead, can be bounded as follows:

$$\begin{aligned}\|AW^*(w_{\delta,p,q} - w^\dagger)\|_Y &= \|\mathbb{P}_q AW^*(w_{\delta,p,q} - w^\dagger)\|_Y + \|(I - \mathbb{P}_q)AW^*(w_{\delta,p,q} - w^\dagger)\|_Y \\ &\leq \|A_{p,q}W^*w_{\delta,p,q} - \mathbb{P}_q m\|_Y + \delta + \|(I - \mathbb{P}_q)A\|_{X\to Y}\|w_{\delta,p,q} - w^\dagger\|_{\ell^1} + \delta \\ &\leq Q + M\|(I - \mathbb{P}_q)A\|_{X\to Y} + \delta,\end{aligned} \tag{C.39}$$

where the positive constant M depends on $\|w^\dagger\|_{\ell^2}$. In order to get an estimate for $Q = \|A_{p,q}W^*w_{\delta,p,q} - \mathbb{P}_q m\|_Y$, we use (C.37): since $0 \leq \beta\|w_{\delta,p,q} - w^\dagger\|_{\ell^1}$, using (C.38) and (C.39) we have

$$0 \leq \frac{1}{\lambda}\|A\|\|w^\dagger - \mathbb{P}_p w^\dagger\|_{\ell^2}^2 + \frac{1}{\lambda}\delta^2 - \frac{1}{\lambda}Q^2 + Q + M\|(I - \mathbb{P}_q)A\|_{X\to Y} + \delta.$$

By solving this second-order inequality we get

$$\begin{aligned} Q &\leq \frac{\lambda}{2} + \frac{\lambda}{2}\left(1 + \frac{4}{\lambda^2}\|A\|^2\|w^\dagger - \mathbb{P}_p w^\dagger\|_{\ell^2}^2 \frac{4}{\lambda}\delta^2 + \frac{4M}{\lambda}\|(I-\mathbb{P}_q)A\|_{X\to Y}\frac{4}{\lambda}\delta\right)^{\frac{1}{2}} \\ &\leq \lambda + \delta + \|A\|\|w^\dagger - \mathbb{P}_p w^\dagger\|_{\ell^2} + M\|(I-\mathbb{P}_q)A\|_{X\to Y} \end{aligned} \tag{C.40}$$

Combining (C.37), (C.38), (C.39), and (C.40) we easily conclude the proof. □

Proof of Proposition 3.4

Proof. Consider the sequence $e_n = \|w_\rho^{(n+1)} - w^{(n+1)}\|_{\ell^2}$. Thanks to the nonexpansivity of the operator $S_{\lambda/L}$, it holds that

$$e_0 = \|\mathcal{T}_{\mathcal{Z}}(w^{(0)}) - \mathcal{T}(w^{(0)})\|_{\ell^2} \leq \frac{1}{L}\|WA_{p,q}^* A_{p,q} W^* - Z\|\|w^{(0)}\|_{\ell^2} \leq \frac{1}{L}\rho\|w^{(0)}\|_{\ell^2}. \tag{C.41}$$

Analogously, for $n \geq 1$,

$$\begin{aligned} e_n &\leq \|I - \frac{1}{L}Z\|\|w^{(n)} - w_\rho^{(n)}\|_{\ell^2} + \frac{1}{L}\|WA_{p,q}^* A_{p,q} W^* - Z\|\|w^{(n)}\|_{\ell^2} \\ &\leq \|I - \frac{1}{L}Z\|e_{n-1} + \frac{1}{L}\rho\|w^{(n)}\|_{\ell^2}. \end{aligned} \tag{C.42}$$

Since $L \geq \|WA_{p,q}^* A_{p,q} W^*\|$, then

$$\|I - \frac{1}{L}Z\| \leq \|I - \frac{1}{L}WA_{p,q}^* A_{p,q} W^*\| + \frac{1}{L}\|WA_{p,q}^* A_{p,q} W^* - Z\| \leq 1 + \frac{1}{L}\rho.$$

Moreover, since the sequence $\{w^{(n)}\}$ is convergent, then it is also bounded: let, *e.g.*, $\|w^{(n)}\|_{\ell^2} \leq M$. As a consequence of (C.41) and (C.42),

$$e_N \leq \sum_{n=0}^{N}\left(1 + \frac{1}{L}\rho\right)^{N-n}\frac{1}{L}\rho\|w^{(n)}\|_{\ell^2} \leq M\left(\left(1 + \frac{1}{L}\rho\right)^{N+1} - 1\right)$$

Let now $N \geq N_0$ and $\rho N \leq \eta_0$: then, with a constant $c = c(N_0, \eta_0)$, it holds:

$$\|w_\rho^{(N)} - w^{(N)}\|_{\ell^2} = e_N \leq M(e^{\frac{1}{L}\rho N} - 1) \leq c\frac{M}{L}\rho N.$$

Combining this result with (3.9), we can guarantee that

$$\|w_\rho^{(N)} - w_{\delta,p,q}\|_{\ell^2} \leq \|w_\rho^{(N)} - w^{(N)}\|_{\ell^2} + \|w^{(N)} - w_{\delta,p,q}\|_{\ell^2} \leq c_3 a^N + \tilde{c}_4\rho N,$$

which proves (3.12). To obtain (3.13), simply substitute $N = \log_a \delta$ and $\rho = \frac{\delta}{N}$ and consider $c_4 = c_3 + \tilde{c}_4$. □

Bibliography

[1] J. Adler, H. Kohr, and O. Öktem. Operator discretization library (ODL), 2017.

[2] J. Adler and O. Oktem. Learned primal-dual reconstruction. *IEEE transactions on medical imaging*, 37(6):1322—1332, 2018.

[3] C. C. Aggarwal. *Neural Networks and Deep Learning*. Springer, 2018.

[4] N. S. Altman. An introduction to kernel and nearest neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[5] A. H. Andersen and A. Kak. Simultaneous algebraic reconstruction technique (SART): A superior implementation of the ART algorithm. *Ultrasonic Imaging*, 6:81 – 94, 1984.

[6] R. Anirudh, H. Kim, J. J. Thiagarajan, K. A. Mohan, K. Champley, and T. Bremer. Lose the views: Limited angle CT reconstruction via implicit sinogram completion. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6343–6352, 2018.

[7] C. Badea and R. Gordon. Experiments with the nonlinear and chaotic behaviour of the multiplicative algebraic reconstruction technique (MART) algorithm for computed tomography. *Physics in Medicine and Biology*, 49(8):1455–1474, 2004.

[8] C. Baese. Method of and apparatus for the localisation of foreign objects in and the radiotherapeutic treatment of the human body by the xrays. *UK Patent N° 100491*, 1916-7. (several references quote 1915 but number unknown and search failed cit by WEBB St: From the watching of shadows. The origins of radiological tomography. Adam Hilger, Bristol and New-York, 1990, 347 pp).

[9] A. R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945, 1993.

[10] J. Barzilai and J. M. Borwein. Two-Point Step Size Gradient Methods. *IMA Journal of Numerical Analysis*, 8(1):141–148, 1988.

[11] F. J. Beekman and C. Kamphuis. Ordered subset reconstruction for x-ray CT. *Physics in Medicine and Biology*, 46(7):1835–1844, 2001.

[12] M. Beister, D. Kolditz, and W. A. Kalender. Iterative reconstruction methods in X-ray CT. *Physica Medica*, 28(2):94—108, 2012.

[13] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[14] M. Bertero and P. Boccacci. *Introduction to inverse problems in imaging*. Bristol: IoP, Institute of Physics Publishing, 1998.

[15] 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), 2020.

[16] M. Bertram, J. Wiegert, D. Schafer, T. Aach, and G. Rose. Directional view interpolation for compensation of sparse angular sampling in cone-beam CT. *IEEE transactions on medical imaging*, 28(7):1011—1022, 2009.

[17] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic programming*, volume 5. Athena Scientific Belmont, MA, 1996.

[18] G. Beylkin, R. Coifman, and V. Rokhlin. Fast wavelet transforms and numerical algorithms i. *Communications on Pure and Applied Mathematics*, 44:141–183, 1991.

[19] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, 2006.

[20] A. E. M. Bocage. Procédé et dispositif de radiographie sur plaque en mouvement. *French Patent No. 536,464*, 1921.

[21] J. Bolte, T. Nguyen, J. Peypouquet, and B. Suter. From error bounds to the complexity of first-order descent methods for convex functions. *Mathematical Programming volume*, 165(2):471–507, 2017.

[22] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual Workshop on Computational Learning Theory*, page 144–152, 1992.

[23] L. Bottou. On-line learning and stochastic approximations. In *On-line Learning in Neural Networks*, pages 9–42, 1998.

[24] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *Society for Industrial and Applied Mathematics Review (SIREV)*, 60:223–311, 2018.

[25] C. A. Bouman and K. Sauer. A unified approach to statistical tomography using coordinate descent optimization. *IEEE Transactions on Image Processing*, 5:480–492, 1996.

[26] 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 International Conference on Machine Learning*, page 111–118, 2010.

[27] K. Bredies and D. Lorenz. Linear convergence of iterative soft-thresholding. *Journal of Inverse and Ill-Posed Problems*, 14(5-6):813–837, 2008.

[28] L. Breiman. Bagging Predictors. *Machine Learning*, 24:123–140, 1996.

[29] T. A. Bubba, M. Galinier, M. Lassas, M. Prato, L. Ratti, and S. Siltanen. Deep neural networks for inverse problems with pseudodifferential operators: an application to limited-angle tomography, 2020.

[30] 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.

[31] T. M. Buzug. *Computed Tomography, From Photon Statistics to Modern Cone-Beam CT, 1st edition.* Springer, 2008.

[32] E. J. Candes, J. Romberg, and T. Tao. Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory*, 52(2):489–509, 2006.

[33] A. Cauchy. Méthode générale pour la résolution des systèmes d'équations simultanées. *Compte rendu des séances de l'académie des sciences*, 25:536–538, 1847.

[34] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In *Proceedings of the International Conference on Neural Information Processing Systems*, page 6571–6583, 2018.

[35] X. Chen, J. Liu, Z. Wang, and W. Yin. Theoretical linear convergence of unfolded ista and its practical weights and thresholds. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NeurIPS)*, page 9079–9089, 2018.

[36] 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.

[37] Z. Chen, X. Jin, L. Li, and G. Wang. A limited-angle CT reconstruction method based on anisotropic tv minimization. *Physics in Medicine and Biology*, 58(7):2119—2141, 2013.

[38] J. Chien and C. Lee. Deep unfolding for topic models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(2):318–331, 2018.

[39] A. Choromanska, M. Henaff, M. Mathieu, G. Arous, and Y. LeCun. The loss surfaces of multilayer networks. *Journal of Machine Learning Research*, 38:192–204, 2015.

[40] R. Cierniak. *X-Ray Computed Tomography in Biomedical Engineering*. Springer, 2011.

[41] R. Clackdoyle and M. Defrise. Tomographic reconstruction in the 21st century, region-of-interest reconstruction from incomplete data. *IEEE Signal Processing Magazine*, 27(4):60–80, 2010.

[42] M. Claesen and B. Moor. Hyperparameter search in machine learning. *ArXiv*, abs/1502.02127, 2015.

[43] S. B. Coban, V. Andriiashen, P. S. Ganguly, M. van Eijnatten, and K. J. Batenburg. Parallel-beam X-ray CT datasets of apples with internal defects and label balancing for machine learning, 2020.

[44] A. M. Cormack. Representation of a function by its line integrals, with some radiological applications. *Journal of Applied Physics*, 34(9):2722–2727, 1963.

[45] A. M. Cormack. Representation of a function by its line integrals, with some radiological applications, II. *Journal of Applied Physics*, 35(9):2908–2913, 1964.

[46] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.

[47] D. Cox and N. Pinto. Beyond simple features: A large-scale feature search approach to unconstrained face recognition. *Face and Gesture*, pages 8–15, 2011.

[48] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines: And Other Kernel-Based Learning Methods*. Cambridge University Press, 1999.

[49] W. Dahmen. Wavelet and multiscale methods for operator equations. *Acta Numerica*, 6:55–228, 1997.

[50] W. Dahmen, S. Prössdorf, and R. Schneider. Wavelet approximation methods for pseudodifferential equations: I stability and convergence. *Mathematische Zeitschrift*, 215(1):583–620, 1994.

[51] X. Dai, J. Bai, T. Liu, and L. Xie. Limited-view cone-beam CT reconstruction based on an adversarial autoencoder network with joint loss. *IEEE Access*, 7:7104–7116, 2019.

[52] I. Daubechies, M. Defrise, and C. De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics*, 57(11):1413–1457, 2004.

[53] Y. Dauphin, R. Pascanu, Çaglar Gülçehre, 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 27*, 2014.

[54] M. de Hoop, M. Lassas, and C. Wong. Deep learning architectures for nonlinear operator functions and nonlinear inverse problems. *arXiv*, 1912.11090, 2019.

[55] B. de Man and J. A. Fessler. Statistical iterative reconstruction for x-ray computed tomography. In *Biomedical Mathematics: Promising Directions in Imaging, Therapy Planning and Inverse Problems*, pages 113–140. Medical Physics Publishing, 2010.

[56] L. Debnath and F. A. Shah. *Lecture Notes on Wavelet Transforms*. Birkhaüser, 2017.

[57] H. B. Demuth, M. H. Beale, O. De Jess, and M. T. Hagan. *Neural Network Design, 2nd edition*. Martin Hagan, 2014.

[58] B. G. Z. des Plantes. Eine neue methode zur differenzierung in der röntgenographie (planigraphie). *Acta Radiologica*, 13:182–192, 1932.

[59] C. Dong, C. C. Loy, K. He, and X. Tang. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):295–307, 2016.

[60] J. Dong, J. Fu, and Z. He. A deep learning reconstruction framework for x-ray computed tomography with incomplete data. *PLOS ONE*, 14(11):1–17, 11 2019.

[61] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

[62] H. W. Engl, M. Hanke, and A. Neubauer. *Regularization of Inverse Problems*. Kluwer Academic Publishers, 1996.

[63] H. Erdogan and J. Fessler. Ordered subsets algorithms for transmission tomography. *Physics in medicine and biology*, 44:2835–2851, 1999.

[64] D. Fanelli and O. Öktem. Electron tomography: a short overview with an emphasis on the absorption potential model for the forward problem. *Inverse Problems*, 24:013001, 2008.

[65] J. A. Fessler and S. D. Booth. Conjugate-gradient preconditioning methods for shift-variant pet image reconstruction. *IEEE Transactions on Image Processing*, 8:688–699, 2002.

[66] J. Flemming and D. Gerth. Injectivity and weak*-to-weak continuity suffice for convergence rates in ℓ_1-regularization. *J. Inverse Ill-Pose. P.*, 26(1):85–94, 2018.

[67] R. Fletcher. *Practical Methods of Optimization, 2nd edition.* Wiley-Interscience, 1987.

[68] G. Franchini, M. Galinier, and M. Verucchi. Mise en abyme with artificial intelligence: How to predict the accuracy of NN, applied to hyper-parameter tuning. In *Recent Advances in Big Data and Deep Learning. Proceedings of the International Neural Networks Society 2019*, volume 1, 2020.

[69] J. Frikel. Sparse regularization in limited angle tomography. *Applied and Computational Harmonic Analysis*, 34(1):117 – 141, 2013.

[70] L. Fu and B. D. Man. A hierarchical approach to deep learning and its application to tomographic reconstruction. In *Proceedings of SPIE, 15th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine*, volume 11072, 2019.

[71] K. Fukunaga. *Introduction to Statistical Pattern Recognition, 2nd edition).* Academic Press Professional, Inc., 1990.

[72] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 2004.

[73] L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style, 2015. cite arxiv:1508.06576.

[74] M. U. Ghani and W. C. Karl. Deep learning-based sinogram completion for low-dose CT. In *2018 IEEE 13th Image, Video, and Multidimensional Signal Processing Workshop (IVMSP)*, pages 1–5, 2018.

[75] P. Gilbert. Iterative methods for the three-dimensional reconstruction of an object from projections. *Journal of Theoretical Biology*, 36(1):105 – 117, 1972.

[76] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, 2010.

[77] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[78] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, page 2672–2680, 2014.

[79] I. J. Goodfellow and O. Vinyals. Qualitatively characterizing neural network optimization problems. *Computing Research Repository (CoRR)*, abs/1412.6544, 2015.

[80] R. Gordon, R. Bender, and G. T. Herman. Algebraic reconstruction techniques (ART) for three-dimensional electron microscopy and x-ray photography. *Journal of Theoretical Biology*, 29(3):471 – 481, 1970.

[81] K. Gregor and Y. LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, page 399–406, 2010.

[82] L. Grippo and M. Sciandrone. Metodi di ottimizzazione per le reti neurali. *Rapporto Tecnico*, 2003.

[83] C. W. Groetsch. *Inverse problems in the mathematical sciences*. Springer Vieweg, 1993.

[84] G. Grossmann. Procédé et dispositif pour la représentation radiographique des sections des corps. *French Patent N° 771887*, 1934.

[85] A. Haar. Zur Theorie der orthogonalen Funktionensysteme. *Mathematische Annalen*, 69(3):331–371, 1910.

[86] J. Hadamard. Sur les problèmes aux dérivées partielles et leur signification physique. *Princeton university bulletin*, pages 49–52, 1902.

[87] Y. Han and J. C. Ye. Framing u-net via deep convolutional framelets: Application to sparse-view CT. *IEEE Transactions on Medical Imaging*, 37(6):1418–1429, 2018.

[88] P. C. Hansen. *Rank-Deficient and Discrete Ill-Posed Problems: Numerical Aspects of Linear Inversion*. Monographs of Mathematical Modeling and Computation. Society for Industrial and Applied Mathematics (SIAM), 1998.

[89] K. Haug and M. Lohne. TF-Wavelets, 2019. Link: https://github.com/UiO-CS/tf-wavelets.

[90] S. S. Haykin. *Neural networks and learning machines, 3rd edition*. Pearson Education, 2009.

[91] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

[92] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[93] M. Heath, K. Bowyer, D. Kopans, W. P. Kegelmeyer, R. Moore, K. Chang, and S. MunishKumaran. Current status of the digital database for screening mammography. In *Proceedings of the Fourth International Workshop on Digital Mammography*, pages 457–460, 1998.

[94] M. Heath, K. Bowyer, D. Kopans, R. Moore, and W. P. Kegelmeyer. The digital database for screening mammography. In *Proceedings of the Fifth International Workshop on Digital Mammography*, pages 212–218. Medical Physics Publishing, 2001.

[95] K. Heiskanen, H. Rhim, and P. Monteiro. Computer simulations of limited angle tomography of reinforced concrete. *Cement Concrete Research*, 21(4):625–634, 1991.

[96] W. R. Hendee. Cross sectional medical imaging: a history. *Radiographics*, 9(6):1155–1180, 1989.

[97] G. T. Herman. *Tomography*, page 801–845. In [184], 2015.

[98] J. Hershey, J. L. Roux, and F. Weninger. Deep unfolding: Model-based inspiration of novel deep architectures. *ArXiv*, abs/1409.2574, 2014.

[99] T. Heskes and H. Kappen. On-line learning processes in artificial neural networks. *North-holland Mathematical Library*, 51:199–233, 1993.

[100] G. E. Hinton. Neural networks for machine learning. Coursera, video lectures, 2012.

[101] G. E. Hinton and S. Osindero. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:2006, 2006.

[102] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[103] G. N. Hounsfield. A method of and apparatus for measuring x or gamma radiation. *UK Patent N° 1283915*, 1968-1972.

[104] J. Hsieh, B. Nett, Z. Yu, K. Sauer, J.-B. Thibault, and C. A. Bouman. Recent advances in CT image reconstruction. *Current Radiology Reports*, 1(1):39–51, 2013.

[105] Y. Huang, A. Preuhs, M. Manhart, G. Lauritsch, and A. Maier. Data consistent ct reconstruction from insufficient data with learned prior images. *ArXiv*, abs/2005.10034, 2020.

[106] Y. Huang, O. Taubmann, X. Huang, V. Haase, G. Lauritsch, and A. Maier. Scale-space anisotropic total variation for limited angle tomography. *IEEE Transactions on Radiation and Plasma Medical Sciences*, 2(4):307–314, 2018.

[107] D. Hubel and T. Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, 124(3):574–591, 1959.

[108] L. Hörmander. *Fourier integral operators. I.*, volume 1-2, page 79–183. Acta Mathematica, 1971.

[109] L. Hörmander. *The analysis of linear partial differential operators. I-IV*. Springer-Verlag, 1983-1985.

[110] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, volume 37, page 448–456, 2015.

[111] D. Ito, S. Takabe, and T. Wadayama. Trainable ISTA for sparse signal recovery. *IEEE Transactions on Signal Processing*, 67(12):3113–3125, 2019.

[112] R. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1:295–307, 1988.

[113] N. Jaitly and E. Hinton. Vocal tract length perturbation (VTLP) improves speech recognition. In *Proceedings of International Conference on Machine Learning*, 2013.

[114] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, pages 2146–2153, 2009.

[115] 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 : a publication of the IEEE Signal Processing Society*, 26(9):4509—4522, 2017.

[116] U. S. Kamilov and H. Mansour. Learning optimal nonlinearities for iterative thresholding algorithms. *IEEE Signal Processing Letters*, 23(5):747–751, 2016.

[117] C. Kamphuis and F. J. Beekman. Accelerated iterative transmission CT reconstruction using an ordered subsets convex algorithm. *IEEE Transactions on Medical Imaging*, 17(6):1101–1105, 1998.

[118] T. Karras, S. Laine, and T. Aila. A style-based generator architecture for generative adversarial networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4396–4405, 2019.

[119] J. B. Keller. Inverse problems. *The American Mathematical Monthly*, 83(2):107–118, 1976.

[120] J. Kieffer. X-ray device and method of technique. *EU Patent N° 1954321*, 1929.

[121] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *Computing Research Repository (CoRR)*, abs/1412.6980, 2015.

[122] E. Kobler, T. Klatzer, K. Hammernik, and T. Pock. Variational networks: Connecting variational methods and deep learning. In *German Conference on Pattern Recognition (GCPR)*, 2017.

[123] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, volume 2, page 1137–1143, 1995.

[124] J. J. Kohn and L. Nirenberg. An algebra of pseudo-differential operators. *Communications on Pure and Applied Mathematics*, 18:269–305, 1965.

[125] V. Kolehmainen, S. Siltanen, S. Järvenpää, J. Kaipio, P. Koistinen, M. Lassas, J. Pirttilä, and E. Somersalo. Statistical inversion for medical X-ray tomography with few radiographs: II. application to dental radiology. *Physics in Medicine and Biology*, 48:1465–1490, 2003.

[126] H. Kostler, M. Prummer, U. Rude, and J. Hornegger. Adaptive variational sinogram interpolation of sparsely sampled CT data. In *18th International Conference on Pattern Recognition*, volume 3, pages 778–781, 2006.

[127] V. P. Krishnan and E. T. Quinto. *Microlocal analysis in tomography*, pages 847–902. In [184], 2015.

[128] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25, pages 1097–1105. MIT Press, 2012.

[129] J. Kukacka, V. Golkov, and D. Cremers. Regularization for deep learning: A taxonomy. *ArXiv*, abs/1710.10686, 2017.

[130] K. Lange and R. Carson. EM reconstruction algorithms for emission and transmission tomography. *Journal of computer assisted tomography*, 8:306–316, 1984.

[131] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[132] G. Lee, R. Gommers, F. Waselewski, K. Wohlfahrt, and A. O'Leary. PyWavelets: A Python package for wavelet analysis. *Journal of open source software*, 4(36):1237, 2019.

[133] H. Lee, J. Lee, H. Kim, B. Cho, and S. Cho. Deep-neural-network-based sinogram synthesis for sparse-view CT image reconstruction. *IEEE Transactions on Radiation and Plasma Medical Sciences*, 3(2):109–119, 2019.

[134] S.-J. Lee. Accelerated coordinate descent methods for Bayesian reconstruction using ordered subsets of projection data. In *Mathematical Modeling, Estimation, and Imaging*, volume 4121, pages 170 – 181, 2000.

[135] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6:861–867, 1993.

[136] Y. Li, Y. Chen, Y. Hu, A. Oukili, L. Luo, W. Chen, and C. Toumoulin. Strategy of computed tomography sinogram inpainting based on sinusoid-like curve decomposition and eigenvector-guided interpolation. *Journal of the Optical Society of America*, 29(1):153–163, 2012.

[137] Y. Li, K. Li, C. Zhang, J. Montoya, and G. Chen. Learning to reconstruct computed tomography images directly from sinogram data under a variety of data acquisition conditions. *IEEE Transactions on Medical Imaging*, 38(10):2469–2481, 2019.

[138] Z. Li, A. Cai, L. Wang, W. Zhang, C. Tang, L. Li, N. Liang, and B. Yan. Promising generative adversarial network based sinogram inpainting method for ultra-limited-angle computed tomography imaging. *Sensors*, 19(18):3941, 2019.

[139] G. Lin, C. Shen, A. van dan Hengel, and I. Reid. Efficient piecewise training of deep structured models for semantic segmentation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3194–3203, 2016.

[140] J. Liu, X. Chen, Z. Wang, and W. Yin. ALISTA: Analytic weights are as good as learned weights in LISTA. In *International Conference on Learning Representations (ICLR)*, 2019.

[141] X. Luo, W. Yu, and C. Wang. An image reconstruction method based on Total Variation and wavelet tight frame for limited-angle CT. *IEEE Access*, 6:1461–1470, 2018.

[142] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning (ICML), Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[143] S. Mallat. *A wavelet tour of signal processing. The sparse way. 3rd edition.* Elsevier/Academic Press, 2009.

[144] S. Manglos, G. Gagne, A. Krol, F. Thomas, and R. Narayanaswamy. Transmission maximum-likelihood reconstruction with ordered subsets for cone beam CT. *Physics in medicine and biology*, 40(7):1225—1241, 1995.

[145] 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*, abs/1711.10046, 2017.

[146] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[147] C. E. Metz. Basic principles of roc analysis. *Seminars in Nuclear Medicine*, 8(4):283 – 298, 1978.

[148] M. Minsky and S. Papert. *Perceptrons. An introduction to computational geometry*. MIT press, 1969.

[149] R. V. Mises and H. Pollaczek-Geiringer. Praktische Verfahren der Gleichungsauflösung . *Zamm-zeitschrift Fur Angewandte Mathematik Und Mechanik*, 9(2):152–164, 1929.

[150] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[151] V. Monga, Y. Li, and Y. C. Eldar. Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *ArXiv*, abs/1912.10557, 2019.

[152] J. L. Mueller and S. Siltanen. *Linear and Nonlinear Inverse Problems with Practical Applications*. Society for Industrial and Applied Mathematics (SIAM), 2012.

[153] 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 9*, pages 599–605. MIT Press, 1997.

[154] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning (ICML)*, 2010.

[155] F. Natterer. *The Mathematics of Computerized Tomography. Classics in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), 2001. Reprint of the 1986 original.

[156] K. Niinimäki, S. Siltanen, and V. Kolehmainen. Bayesian multiresolution method for local tomography in dental X-ray imaging. *Physics in Medicine and Biology*, 52(22):6663–6678, oct 2007.

[157] S. Niu, Y. Gao, Z. Bian, J. Huang, W. Chen, G. Yu, Z. Liang, and J. Ma. Sparse-view X-ray CT reconstruction via total generalized variation regularization. *Physics in medicine and biology*, 59(12):2997—3017, 2014.

[158] P. Ochs, R. Ranftl, T. Brox, and T. Pock. Techniques for gradient based bilevel optimization with nonsmooth lower level problems. *Journal of Mathematical Imaging and Vision*, 56(2):175–194, 2016.

[159] C. Oliver. Synthetic-aperture radar imaging. *Journal of Physics D: Applied Physics*, 22(7):871–890, 1989.

[160] J. M. Park, E. A. Franken, M. Garg, L. L. Fajardo, and L. T. Niklason. Breast tomosynthesis: present considerations and future applications. *Radiographics : a review publication of the Radiological Society of North America, Inc*, 27 Suppl 1:S231—S240, 2007.

[161] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning*, volume 28, pages 1310–1318, 2013.

[162] E. L. Piccolomini and F. Zama. Regularization algorithms for image reconstruction from projections. In *Series on Advances for Applied Science, volume 45 of World Scientific*. World Scientific, 1996.

[163] E. L. Piccolomini and F. Zama. The conjugate gradient regularization method in computed tomography problems. *Applied Mathematics and Computation*, 102(1):87 – 99, 1999.

[164] N. Pinto, Z. Stone, T. E. Zickler, and D. Cox. Scaling up biologically-inspired computer vision: A case study in unconstrained face recognition on facebook. *IEEE Computer Vision and Pattern Recognition (CVPR) 2011 workshops*, pages 35–42, 2011.

[165] E. Pohl. Method and apparatus for making roentgen projections. *EU Patent N° 332496*, 1927.

[166] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[167] F. Portes and M. Chausse. Procédé pour la mise au point radiologique sur un plan d'un solide, ainsi que pour la concentration sur une zone déterminée d'une action radiothérapeutique maximum, et dispositifs en permettant la réalisation. *French Patent N° 541914*, 1921-2.

[168] E. T. Quinto. An introduction to X-ray tomography and radon transforms. In *Proceedings of Symposia in Applied Mathematics*, 2006.

[169] J. Radon. Über die bestimmung von funktionen durch ihre integralwerte längs gewisser mannigfaltigkeiten. *Berichte über Verhandlungen der Sächsischen Akademie der Wissenschaften zu Leipzig. Mathematisch-physische Klasse.*, 69:262–277, 1917.

[170] R. Reisenhofer, S. Bosse, G. Kutyniok, and T. Wiegand. A Haar wavelet-based perceptual similarity index for image quality assessment. *Signal Processing: Image Communication*, 61:33–43, 2018.

[171] G. Riegler, D. Ferstl, M. Rüther, and H. Bischof. A deep primal-dual network for guided depth super-resolution. In *Proceedings of the British Machine Vision Conference (BMVC)*, pages 1–14, 2016.

[172] L. Ritschl, F. Bergner, C. Fleischmann, and M. Kachelrieß. Improved total variation-based CT image reconstruction applied to clinical data. *Physics in Medicine and Biology*, 56(6):1545–1561, 2011.

[173] H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

[174] L. Rokach and O. Z. Maimon. *Data mining with decision trees: theory and applications*, volume 69. World scientiffic, 2008.

[175] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, 2015.

[176] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

[177] D. E. Rumelhart, J. L. McClelland, and eds. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1.* MIT Press, 1986.

[178] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach, 3rd edition.* Prentice Hall Press, 2009.

[179] W. C. Röentgen. Eine neue art von strahlen (ii. mittheilung). *Sitzungsberichten der Würzburger Physikalisch-Medizinische. Gesellschaft*, 1895.

[180] W. C. Röentgen. Ueber eine neue art von strahlen (vorläufige mittheilung). *Sitzungsberichten der Würzburger Physikalisch-Medizinische. Gesellschaft*, 1895.

[181] M. Salo. Lecture notes on applications of microlocal analysis to inverse problems, June 2019.

[182] K. Sauer and C. Bouman. A local update strategy for iterative reconstruction from projections. *IEEE Transactions on Signal Processing*, 41:534–548, 1993.

[183] A. M. Saxe, P. W. Koh, Z. Chen, M. Bhand, B. Suresh, and A. Ng. On random weights and unsupervised feature learning. In *International Conference on Machine Learning (ICML)*, 2011.

[184] O. Scherzer. *Handbook of Mathematical Methods in Imaging. 2nd edition.* Springer New York, 2015.

[185] A. G. Schwing and R. Urtasun. Fully connected deep structured networks. *Computing Research Repository (CoRR)*, abs/1503.02351, 2015.

[186] E. Y. Sidky and X. Pan. Image reconstruction in circular cone-beam computed tomography by constrained, total-variation minimization. *Physics in Medicine and Biology*, 53(17):4777–4807, 2008.

[187] J. Sietsma and R. J. Dow. Creating artificial neural networks that generalize. *Neural Networks*, 4(1):67 – 79, 1991.

[188] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[189] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*, volume 2, page 2951–2959, 2012.

[190] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[191] I. Sutskever, J. Martens, G. Dahl, and G. E. Hinton. On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning (ICML)*, volume 28, page 1139–1147, 2013.

[192] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[193] R. S. Sutton and A. G. Barto. *Introduction to reinforcement learning*, volume 2. MIT press Cambridge, 1998.

[194] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[195] O. Taubmann, M. Unberath, G. Lauritsch, S. Achenbach, and A. Maier. Spatio-temporally regularized 4-d cardiovascular C-arm CT reconstruction using a proximal algorithm. In *2017 IEEE 14th International Symposium on Biomedical Imaging (ISBI 2017)*, pages 52–55, 2017.

[196] J.-B. Thibault, K. Sauer, C. Bouman, and J. Hsieh. Three-dimensional statistical approach to improved image quality for multislice helical ct. *Medical Physics*, 34:4526–4544, 2007.

[197] R. V. Tiggelen. In search for the third dimension: from radiostereoscopy to three-dimensional imaging. *Belgian Journal of Radiology*, 85(5):266–270, 2002.

[198] A. N. Tikhonov. Solution of incorrectly formulated problems and the regularization method. *Soviet mathematics - Doklady*, 4:1035–1038, 1963.

[199] A. M. Turing. I.— Computing machinery and intelligence. *Mind*, LIX(236):433–460, 1950.

[200] G. Uhlmann and A. Vasy. The inverse problem for the local geodesic ray transform. *Inventiones mathematicae*, 205(1):83–120, 2016.

[201] A. Vallebona. Una modalità di tecnica per la dissociazione radiografica delle ombre applicata allo studio del cranio. *La Radiologia medica*, 17:1090–1097, 1932.

[202] W. van Aarle, W. J. Palenstijn, J. Cant, E. Janssens, F. Bleichrodt, A. Dabravolski, J. D. Beenhouwer, J. Batenburg, and J. Sijbers. Fast and flexible X-ray tomography using the ASTRA toolbox. *Optics Express*, 24(22):25129–25147, 2016.

[203] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, and T. Yu. scikit-image: Image processing in Python, 2014.

[204] V. N. Vapnik. Principles of risk minimization for learning theory. In *Advances in Neural Information Processing Systems 4*, pages 831–838. MIT Press, 1992.

[205] V. N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., 1995.

[206] V. N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998.

[207] H. Villarraga-Gómez, E. L. Herazo, and S. T. Smith. X-ray computed tomography: from medical imaging to dimensional metrology. *Precision Engineering*, 60:544–569, 2019.

[208] C. Vogel and T. Pock. A primal dual network for low-level vision problems. In *German Conference on Pattern Recognition (GCPR)*, 2017.

[209] C. R. Vogel. *Computational Methods for Inverse Problems*. Frontiers in Applied Mathematics. Society for Industrial and Applied Mathematics (SIAM), 2002.

[210] S. Wang, S. Fidler, and R. Urtasun. Proximal deep structured models. In *Proceedings of the International Conference on Neural Information Processing Systems*, 2016.

[211] T. Wang, K. Nakamoto, H. Zhang, and H. Liu. Reweighted anisotropic total variation minimization for limited-angle CT reconstruction. *IEEE Transactions on Nuclear Science*, 64:2742–2760, 2017.

[212] 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.

[213] Z. Wang, Q. Ling, and T. Huang. Learning deep ℓ_0 encoders. In *Proceedings of the 13th Association for the Advancement of Artificial Intelligence (AAAI) conference*, 2016.

[214] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, 1989.

[215] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

[216] M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, page 681–688, 2011.

[217] D. R. Wilson and T. R. Martinez. The general inefficiency of batch training for gradient descent learning. *Neural Networks*, 16(10):1429–1451, 2003.

[218] J. M. Wolterink, T. Leiner, M. A. Viergever, and I. Išgum. Generative adversarial networks for noise reduction in low-dose CT. *IEEE Transactions on Medical Imaging*, 36(12):2536–2545, 2017.

[219] D. Wynen, C. Schmid, and J. Mairal. Unsupervised learning of artistic styles with archetypal style analysis. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, page 6584–6593, 2018.

[220] T. Würfl, M. Hoffmann, V. Christlein, K. Breininger, Y. Huang, M. Unberath, and A. K. Maier. Deep learning computed tomography: Learning projection-domain weights from image domain in limited angle problems. *IEEE Transactions on Medical Imaging*, 37(6):1454–1463, 2018.

[221] F. Xu, W. Xu, M. Jones, B. Keszthelyi, J. Sedat, D. Agard, and K. Mueller. On the efficiency of iterative ordered subset reconstruction algorithms for acceleration on GPUs. *Computer Methods and Programs in Biomedicine*, 98(3):261 – 270, 2010.

[222] L. Xu, J. S. J. Ren, C. Liu, and J. Jia. Deep convolutional neural network for image deconvolution. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, page 1790–1798, 2014.

[223] Q. Xu, H. Yu, X. Mou, L. Zhang, J. Hsieh, and G. Wang. Low-dose X-ray CT reconstruction via dictionary learning. *IEEE Transactions on Medical Imaging*, 31:1682–1697, 2012.

[224] K. Yan, X. Wang, L. Lu, and R. Summers. DeepLesion: automated mining of large-scale lesion annotations and universal lesion detection with deep learning. *Journal of medical imaging*, 5(3), 2018. Database available at `https://nihcc.app.box.com/v/DeepLesion`.

[225] C. Yang, R. Liu, L. Ma, X. Fan, H. Li, and M. Zhang. Unrolled optimization with deep priors for intrinsic image decomposition. In *2018 IEEE 4th International Conference on Multimedia Big Data (BigMM)*, pages 1–7, 2018.

[226] Q. Yang, P. Yan, Y. Zhang, H. Yu, Y. Shi, X. Mou, M. K. Kalra, Y. Zhang, L. Sun, and G. Wang. Low-dose CT image denoising using a generative adversarial network with wasserstein distance and perceptual loss. *IEEE Transactions on Medical Imaging*, 37(6):1348–1357, 2018.

[227] X. Yi, E. Walia, and P. Babyn. Generative adversarial network in medical imaging: A review. *Medical Image Analysis*, 58:101552, 2019.

[228] H. Yu and G. Wang. A soft-threshold filtering approach for reconstruction from a limited number of projections. *Physics in Medicine and Biology*, 55(13):3905–3916, 2010.

[229] Z. Yu, J. Thibault, C. A. Bouman, K. D. Sauer, and J. Hsieh. Fast model-based X-ray CT reconstruction using spatially nonhomogeneous icd optimization. *IEEE Transactions on Image Processing*, 20(1):161–175, 2011.

[230] M. D. Zeiler, M. A. Ranzato, and R. M. et al. On rectified linear units for speech processing. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3517–3521, 2013.

[231] J. Zhang and B. Ghanem. ISTA-Net: Interpretable optimization-inspired deep network for image compressive sensing. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1828–1837, 2018.

[232] 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.

[233] Y. Zhang, H. P. Chan, B. Sahiner, et al. A comparative study of limited–angle cone-beam reconstruction methods for breast tomosynthesis. *Medical Physics*, 33(10):3781–3795, 2006.

[234] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. S. Torr. Conditional random fields as recurrent neural networks. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1529–1537, 2015.

[235] Y. Zhou and R. Chellappa. Computation of optical flow using a neural network. *IEEE 1988 International Conference on Neural Networks*, 2:71–78, 1988.

[236] B. Zhu, J. Z. Liu, B. Rosen, and M. Rosen. Image reconstruction by domain-transform manifold learning. *Nature*, 555:487–492, 2018.

[237] H. Zhu, H. Shu, J. Zhou, and L. Luo. A weighted least squares pet image reconstruction method using iterative coordinate descent algorithms. In *IEEE Symposium Conference Record Nuclear Science*, volume 6, pages 3380–3384, 2004.