Stability of Adaptive Neural Networks for Image Reconstruction

Kristian Monsen Haug

Master's Thesis, Spring 2019

Abstract

Over the past decade, compressive sensing and deep learning have emerged as viable techniques for reconstruction of images using far fewer samples than what Shannon's sampling theory dictates. The two methods are fundamentally quite different. Compressive sensing relies heavily on the existence of a sparsifying transform, such as the discrete wavelet transform. Deep learning, on the other hand, tries to generalize large amounts of training data and therefore avoids a priori assumptions about the image.

While we for compressive sensing have good mathematical results which allow us to control the recovery error, the same cannot be said about deep learning. Here we have no bounds on the error, and it is unclear whether or not stable recovery is possible. Such bounds are important to guarantee stable recovery, which in turn is vital for applications like medical imaging. Otherwise we risk worst-case scenarios such as a tumor showing up in an MRI scan of a healthy patient if they move a few millimeters.

In this thesis we look for connections between the two fields. We consider algorithms for solving compressive sensing, and see that they can be written as neural networks. Because of this, we can for the first time test the stability of these algorithms when exposed to worst case noise. We find promising indications for the stability of certain algorithms. In addition, we see a speedup of several orders of magnitude when we run the algorithms as neural networks.

Acknowledgements

First, I would like to thank my supervisors Anders Hansen and Øyvind Ryan. Anders has provided an interesting research topic and useful input along the way. Øyvind has been of great help during the writing process, by proofreading drafts and suggesting improvements along with additional reading material. A special thanks goes to Vegard Antun, for the invaluable help he has provided during the writing of this thesis, along with ideas, feedback, and enthusiasm.

This past year, I have been lucky enough to share study hall with Mathias Lohne and Keith Zhou. I want to thank them for their support and insightful discussions whenever one of us was stuck on a problem. Mathias is owed a special thanks for collaborating with me on an implementation of the discrete wavelet transform, which I have put to great use in this thesis.

I would also like to extend my gratitude to my girlfriend, Celine, along with friends and family, for their support throughout this past year.

Kristian Monsen Haug
May 2019

Preface

While the code providing the main results of this thesis is not included in the text, the author stress that it constitutes a body of work that should not be diminished. The code provides implementations of two optimization algorithms and code related to the setup of this thesis.

The code is written in Python 3. Most notable are the implementations that use the deep learning framework Tensorflow. This, along with theoretical observations made in this thesis, allows us to easily run stability tests on the algorithms. This has previously been done only for learned neural networks.

Another beneficial consequence of the Tensorflow implementations is that we with ease can run the algorithms on GPUs. This improves the running time an order of magnitude for reasonably large inputs, compared to the CPU implementations.

To make the theoretical setup work, a Tensorflow implementation of the discrete wavelet transform was needed. In collaboration with Mathias Lohne, a Python package to compute the discrete wavelet transform has also been developed.

The optimization library is available as a python package at https://github.com/UiO-CS/optimization, and the wavelet library at https://github.com/UiO-CS/tf-wavelets. Code for generating some of the figures in this thesis can be found at https://github.com/krimha/thesis_code

Contents

CHAPTER 1

Introduction

Since the mid-2000s, sparse regularization techniques such as compressive sensing have been preferred for solving inverse problems in medical imaging. In recent years, the field of deep learning has revolutionized the artificial intelligence community, and has therefore emerged as a competitor to these techniques. It is known that neural networks are good function approximators, but recent discoveries have made it clear that even state of the art deep learning methods are prone to extreme stability issues. Even tiny perturbations of the input yield drastic changes in the output [Ant+19; Hua+18; MFF16; Sze+13]. These issues make neural networks less trustworthy, and less suitable for critical applications such as medical imaging. For compressive sensing, we have theoretical results guaranteeing stable recovery, and it is therefore considered the champion in this regard.

A major downside is that for any input, solving the compressive sensing problem takes considerably longer than evaluating a trained neural network. The deep learning literature tends to focus on this advantage of deep learning [Sch+18], without mentioning the positive properties of compressive sensing.

Due to the increasing popularity of deep learning, several frameworks for high-level languages have been developed to ease the implementation of neural networks on Graphical Processing Units (GPU). A notable example is Tensorflow [TF]. This makes GPU programming possible for anyone with experience with scientific computing in Python, and without the technicalities of relatively low-level languages like C or C++.

The usefulness of these frameworks is not limited to deep learning. Indeed, we shall see that some iterative algorithms used for compressive sensing have structures resembling deep neural networks when unrolled. Therefore, it is possible to implement them on the GPU using Tensorflow. An immediate advantage of this, is that we will get a notable speedup when running the algorithms compared to when they are running on a *Central Processing Unit* (CPU). This also allow us to make a more fair comparison between compressive sensing and deep learning, as they now both can run on the same hardware, using the same software.

Because a Tensorflow implementation makes it trivial to compute the gradient of the algorithms, we are able to perform worst case stability tests, which has previously only been done on trained neural networks. The main contribution of this thesis, is the first initial tests of worst case performance for wavelet reconstruction with compressive sensing decoders.

1.1 Overview

The following is a brief description of each chapter

Chapter 2 provides an overview of compressive sensing. We show fundamental results for when we can recover sparse vectors, and what optimization problems we want to solve.

Chapter 3 talks briefly about the type of inverse problems encountered in medical imaging. In particular, we consider magnetic resonance imaging, and explain how we reach the setup considered for the remained of the thesis

Chapter 4 attempts to apply the theory from Chapter 2 to the medical imaging problem described in Chapter 3. First we discuss how we can achieve a sparse representation of images. The rest of the chapter discusses how we can apply the traditional compressive sensing theory to imaging problems, and what problems need to be addressed.

Chapter 5 gives a quick introduction to deep learning with neural networks for image reconstruction. Most importantly the standard definitions, and the theory that motivates their application to imaging.

Chapter 6 describes and proves the correctness of the two optimization algorithms we will consider in this thesis: FISTA, and Chambolle and Pock's primal-dual algorithm.

Chapter 7 attempts to write the algorithms from Chapter 6 as neural networks, introducing a class of neural networks that adapts to the input.

Chapter 8 demonstrates that the results from Chapter 7 provides successful image reconstruction, and investigates the stability of the algorithms from Chapter 6.

Chapter 9 summarizes the thesis, and emphasises natural directions for further research.

CHAPTER 2

Classical compressive sensing theory

2.1 Notation

Throughout this thesis, we will use the following notation: For convenience, we define $[N]$ to be the set $\{1, 2, \dots, N\}$, and set $a \lesssim b$ to mean that there is a constant C independent of any other values such that $a \leq Cb$.

For any vector $\boldsymbol{x} \in \mathbb{C}^N$ and index set $S \subseteq [N]$, we let $\boldsymbol{x}_S$ denote either the vector in $\mathbb{C}^N$ that is equal to $\boldsymbol{x}$ on S and zero otherwise; or the vector in $\mathbb{C}^{|S|}$ that contain only the entries of $\boldsymbol{x}$ that are indexed by S. Which one of these we are working with will be clear from context, or made precise if necessary.

2.2 Sparse solutions to underdetermined problems

Suppose we have a signal $\boldsymbol{x} \in \mathbb{C}^N$ and a matrix $\boldsymbol{A} \in \mathbb{C}^{m \times N}$ with $m < N$.

We want to recover $\boldsymbol{x}$ from $\boldsymbol{y} \in \mathbb{C}^m$, given by

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$$

This underdetermined linear system has an infinite number of solutions. To correctly recover $\boldsymbol{x}$ from the measurements, we need more information about $\boldsymbol{x}$. The fundamental assumption in compressive sensing is *sparsity.* We call a vector s-sparse if it has *at most* s nonzero entries. The following theorem shows two equivalent conditions that are necessary and sufficient to recover any s-sparse vector:

Theorem 2.1 ([FR13, p. 49])**.** *Given $\boldsymbol{A} \in \mathbb{C}^{m \times N}$, for every s-sparse vector, the following points are equivalent:*

(a) *Every s-sparse vector $\boldsymbol{x} \in \mathbb{C}^N$ is the unique solution of $\boldsymbol{A}\boldsymbol{z} = \boldsymbol{A}\boldsymbol{x}$.*

(b) *The null space* $\ker \boldsymbol{A}$ *of $\boldsymbol{A}$ does not contain any $2s$-sparse vectors other than the zero vector.*

(c) *Every set of $2s$ columns of $\boldsymbol{A}$ is linearly independent.*

Proof. (a) $\implies$ (b): Assume that every s-sparse vector $\boldsymbol{x}$ is the unique solution to $\boldsymbol{A}\boldsymbol{z} = \boldsymbol{A}\boldsymbol{x}$. Let $\boldsymbol{v} \in \ker \boldsymbol{A}$ be a $2s$-sparse vector. We can write $\boldsymbol{v} = \boldsymbol{x} - \boldsymbol{z}$ where both $\boldsymbol{x}$ and $\boldsymbol{z}$ are s-sparse, with disjoint supports. Thus, we have $\boldsymbol{Ax} = \boldsymbol{Az}$, which implies that $\boldsymbol{x} = \boldsymbol{z}$. Because they have disjoint support, they can only be equal if $\boldsymbol{v}$ is he zero vector.

(b) $\Longrightarrow$ (a): Let $\boldsymbol{x}$ and $\boldsymbol{z}$ be two s-sparse vectors such that $\boldsymbol{Ax} = \boldsymbol{Az}$. Thus $\boldsymbol{x} - \boldsymbol{z}$ is in $\ker \boldsymbol{A}$. In addition, this vector is also in the null space. Therefore, by assumption, it must be the zero vector.

■

This also tells us that we need $m \geq 2s$ to recover all s-sparse vectors.

Given the measurement vector, we are now able to recover $\boldsymbol{x}$ by solving the optimization problem

$$\min_{\boldsymbol{z} \in \mathbb{C}^N} \|\boldsymbol{z}\|_0 \quad \text{subject to} \quad \boldsymbol{Az} = \boldsymbol{y} \tag{2.1}$$

This way of solving inverse problems unfortunately gives us some computational problems. The ℓ_0-minimization problem is NP-hard [FR13, p. 54], and the time required will be unreasonably large to actually do it in practice. Instead, we want to solve the problem arising from the convex relaxation of (2.1), which is

$$\min_{\boldsymbol{z} \in \mathbb{C}^N} \|\boldsymbol{z}\|_1 \quad \text{subject to} \quad \boldsymbol{Az} = \boldsymbol{y} \tag{2.2}$$

This problem is referred to as *basis pursuit*. Clearly, this is an entirely different problem, and we risk getting non-sparse solutions, even if the sensing matrix satisfies the criterion set in Theorem 2.1. We need further requirements on $\boldsymbol{A}$ to guarantee sparse solutions. One such property is the null space property:

Definition 2.2 (Null space property [FR13, p. 78]). A matrix $\boldsymbol{A} \in \mathbb{C}^{m \times N}$ is said to satisfy the *null space property* relative to a set $S \in [N]$ if

$$\|\boldsymbol{v}_S\|_1 < \|\boldsymbol{v}_{\overline{S}}\|_1 \quad \text{for all } \boldsymbol{v} \in \ker \boldsymbol{A} \setminus \{\boldsymbol{0}\}$$

Using this definition of the null space property, we reach the following result

Theorem 2.3 ([FR13, p. 79]). *Given a matrix $\boldsymbol{A} \in \mathbb{C}^{m \times N}$, every vector $\boldsymbol{x} \in \mathbb{C}^N$ supported on a set S is the unique solution of (2.2) with $\boldsymbol{y} = \boldsymbol{Ax}$ if and only if $\boldsymbol{A}$ satisfies the null space property relative to S.*

Proof. Suppose that we are given an index set S, and assume that every s-sparse vector is the unique minimizer of $\|\boldsymbol{z}\|_1$ subject to $\boldsymbol{Az} = \boldsymbol{Ax}$. Thus, for any nonzero $\boldsymbol{v} \in \ker \boldsymbol{A}$, we have that $\boldsymbol{v}_S$ is the unique minimizer of $\|\boldsymbol{z}\|_1$ subject to $\boldsymbol{Az} = \boldsymbol{Av}_S$. However, we also have that $\boldsymbol{Av}_{\overline{S}} = \boldsymbol{Av}_S$ which means that both $\boldsymbol{v}_S$ and $\boldsymbol{v}_{\overline{S}}$ are feasible. Therefore, we must have $\|\boldsymbol{v}_S\|_1 < \|\boldsymbol{v}_{\overline{S}}\|_1$.

Conversely, assume that the null space property relative to S holds. Given $\boldsymbol{x}$ and $\boldsymbol{z}$ such that $\boldsymbol{x} \neq \boldsymbol{z}$ and $\boldsymbol{Ax} = \boldsymbol{Az}$. Thus, by setting $\boldsymbol{v} = \boldsymbol{x} - \boldsymbol{z} \in \ker \boldsymbol{A} \setminus \{\boldsymbol{0}\}$ and using the null space property relative to S, we have

$$\begin{aligned}
\|\boldsymbol{x}\|_1 &= \|\boldsymbol{x} - \boldsymbol{z}_S + \boldsymbol{z}_S\|_1 \\
&\leq \|\boldsymbol{x} - \boldsymbol{z}_S\|_1 + \|\boldsymbol{z}_S\|_1 \\
&= \|\boldsymbol{v}_S\|_1 + \|\boldsymbol{z}_S\|_1 \\
&< \|\boldsymbol{v}_{\overline{S}}\|_1 + \|\boldsymbol{z}_S\|_1 \\
&= \|-\boldsymbol{z}_{\overline{S}}\|_1 + \|\boldsymbol{z}_S\|_1 \\
&= \|\boldsymbol{z}\|_1
\end{aligned}$$

Which means that $\boldsymbol{x}$ is the optimal solution. ■

If this holds for all $S \in [N]$ with at most s elements, we get that any s-sparse vector can be recovered successfully by solving(2.2) if $\boldsymbol{A}$ satisfies the null space property of order s.

We are also interested in a measure of how suitable a matrix is for successful recovery. We call this measure *coherence*, and desire low values. Several definitions exist, and one is given in Definition 2.4. This is known as the ℓ_1 coherence function

Definition 2.4 (ℓ_1-coherence function [FR13, p. 111]). Let $\boldsymbol{A} = [\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_N]$ be a matrix in $\mathbb{C}^{m\times N}$. The columns $\boldsymbol{a}_i$ satisfy $\|\boldsymbol{a}_i\|_2 = 1$. The ℓ_1-coherence function $\mu_1 \colon [N-1] \to \mathbb{R}$ is given by

$$\mu_1(s) = \max_{i\in[N]} \max \left\{ \sum_{j\in S} \left|\langle \boldsymbol{a}_i, \boldsymbol{a}_j\rangle\right|, S \subseteq [N], |S| = s, i \notin S \right\}$$

The following result tells us what the coherence must be to guarantee successful recovery. This is shown by demonstrating that it implies the null space property.

Theorem 2.5 ([FR13, p. 515]). *Let $\boldsymbol{A} \in \mathbb{C}^{m\times N}$ be a matrix with ℓ_2-normalized columns. If*

$$\mu_1(s) + \mu_1(s-1) < 1 \tag{2.3}$$

then every s-sparse vector $\boldsymbol{x} \in \mathbb{C}^N$ is exactly recovered from the measurement vector $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$, with basis pursuit.

Proof. We will show that Equation (2.3) implies that for every nonzero $\boldsymbol{v} \in \ker \boldsymbol{A}$ and $S \subseteq [N]$ with $|S| = s$, we have

$$\|\boldsymbol{v}_S\|_1 < \|\boldsymbol{v}_{\overline{S}}\|_1$$

In other words, that the null space property of order s is satisfied.

Let $\boldsymbol{v} \in \ker A \setminus \{\boldsymbol{0}\}$, and $S \subseteq [N]$ with $|S| = s$. We denote the columns of $\boldsymbol{A}$ as $\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_N$, and the entires of $\boldsymbol{v}$ as $v_1, \ldots, v_N$.

A key observation, is to see that for any $i \in [N]$ we can write

$$0 = \langle \boldsymbol{A}\boldsymbol{v}, \boldsymbol{a}_i\rangle = \sum_{j=1}^{N} v_j \langle \boldsymbol{a}_j, \boldsymbol{a}_i\rangle$$

And therefore, by isolating the ith term,

$$v_i\langle \boldsymbol{a}_i, \boldsymbol{a}_i\rangle = -\sum_{j=1,j\neq i}^{N} v_j\langle \boldsymbol{a}_j, \boldsymbol{a}_i\rangle$$

Thus, by splitting the sum over $[N]$ into two sums over S and $\overline{S}$, we can write v_i as

$$v_i = v_i\langle \boldsymbol{a}_i, \boldsymbol{a}_i\rangle = -\sum_{l\in\overline{S}} v_l\langle \boldsymbol{a}_l, \boldsymbol{a}_i\rangle - \sum_{j\in S, j\neq i} v_j\langle \boldsymbol{a}_j, \boldsymbol{a}_i\rangle$$

By the triangle inequality, we have

$$|v_i| = \sum_{l\in\overline{S}} |v_l||\langle \boldsymbol{a}_l, \boldsymbol{a}_i\rangle| + \sum_{j\in S, j\neq i} |v_j||\langle \boldsymbol{a}_j, \boldsymbol{a}_i\rangle|$$

By summing over S, we get

$$\begin{aligned}
\|\boldsymbol{v}_S\|_1 = \sum_{i\in S}|v_i| &\leq \sum_{i\in S}\left(\sum_{l\in\overline{S}} |v_l||\langle \boldsymbol{a}_l, \boldsymbol{a}_i\rangle| + \sum_{j\in S, j\neq i} |v_j||\langle \boldsymbol{a}_j, \boldsymbol{a}_i\rangle|\right) \\
&= \sum_{l\in\overline{S}}\left(|v_l|\sum_{i\in S}|\langle \boldsymbol{a}_l, \boldsymbol{a}_i\rangle|\right) + \sum_{j\in S}\left(|v_j|\sum_{i\in S, i\neq j}|\langle \boldsymbol{a}_j, \boldsymbol{a}_i\rangle|\right) \\
&\leq \mu_1(s)\|\boldsymbol{v}_{\overline{S}}\|_1 + \mu_1(s-1)\|\boldsymbol{v}_S\|_1
\end{aligned}$$

This gives

$$(1-\mu_1(s-1))\|\boldsymbol{v}_S\|_1 \leq \mu_1(s)\|\boldsymbol{v}_{\overline{S}}\| < (1-\mu_1(s-1))\|\boldsymbol{v}_{\overline{S}}\|$$

And thus, $\|\boldsymbol{v}_S\|_1 < \|\boldsymbol{v}_{\overline{S}}\|_1$

■

While the coherence seems to work well, better performance can be obtained by a related property known as the *restricted isometry property* (RIP). It is given by

Definition 2.6 ([FR13, p. 133])**.** For a matrix $\boldsymbol{A} \in \mathbb{C}^{m\times N}$, the s-th restricted isometry property constant $\delta_s = \delta_s(\boldsymbol{A})$ is the smallest $\delta \geq 0$ such that

$$(1-\delta)\|\boldsymbol{x}\|_2^2 \leq \|\boldsymbol{A}\boldsymbol{x}\|_2^2 \leq (1+\delta)\|\boldsymbol{x}\|_2^2 \tag{2.4}$$

for all s-sparse vector $\boldsymbol{x} \in \mathbb{C}^N$.

It can be shown that if the restricted isometry constant of a matrix $\boldsymbol{A} \in \mathbb{C}^{m\times N}$ is small, this property will imply the null space property, and thus guarantees successful recovery. The motivation for definition coherence and RIP, instead of simply working with the null space property, is that it makes the construction of suitable matrices easier. Showing that a specific matrix is suitable is difficult. On the other hand, large classes of random matrices are known to satisfy the RIP [FR13, p. 141].

2.3 Alternative optimization problems

So far, we have considered solving the compressive sensing problem with basis pursuit. Because we need them later, we will now introduce two different optimization problems that can be used in compressive sensing.

First, we generalize basis pursuit by assuming that the measurements are noisy, which means that $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{e}$. This is necessary when considering a practical setup. If we assume that $\|\boldsymbol{e}\|_2 \leq \eta$, we want to solve the following optimization problem, known as *Quadratically constrained basis pursuit*:

$$\min_{\boldsymbol{z}\in\mathbb{C}^N} \|\boldsymbol{z}\|_1 \quad \text{subject to } \|\boldsymbol{A}\boldsymbol{z} - \boldsymbol{y}\|_2^2 \leq \eta \tag{2.5}$$

Where $\eta \geq 0$. For $\lambda \geq 0$, we have the following, which we will refer to as *LASSO*.

$$\min_{\boldsymbol{z}\in\mathbb{C}^N} \lambda\|\boldsymbol{z}\|_1 + \|\boldsymbol{A}\boldsymbol{z} - \boldsymbol{y}\|_2 \tag{2.6}$$

To show the equivalence between these problems, we will also consider *Constrained LASSO*.

$$\min_{\boldsymbol{z}\in\mathbb{C}^N} \|\boldsymbol{A}\boldsymbol{z} - \boldsymbol{y}\|_2 \quad \text{subject to } \|\boldsymbol{z}\|_1 \leq \tau \tag{2.7}$$

The above problems are equivalent. Assuming that $\boldsymbol{x}$ is a minimizer of one of the problems above, we can show that $\boldsymbol{x}$ is also a minimizer of the other two problems. This can be done by finding appropriate values of the tuning parameters η, λ and τ. The appropriate values will depend on the minimizer $\boldsymbol{x}$.

Theorem 2.7 ([FR13, p. 64])**.**

(a) *If $\boldsymbol{x}$ is a minimizer of the optimization problem* (2.6) *with $\lambda > 0$, then there exists $\eta \geq 0$ such that $\boldsymbol{x}$ is a minimizer of the problem* (2.5)

(b) *If $\boldsymbol{x}$ is a unique minimizer of the problem in* (2.5) *with $\eta \geq 0$, then there is a $\tau \geq 0$ such that $\boldsymbol{x}$ is a unique minimizer of* (2.7)*.*

(c) *If $\boldsymbol{x}$ is a minimizer of* (2.7) *with $\tau > 0$, then there exists a $\lambda \geq 0$ such that $\boldsymbol{x}$ is a minimizer of the optimization problem* (2.6)*.*

Proof. Let $\boldsymbol{x}$ be a minimizer of Equation (2.6). We set $\eta = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2$. Using that $\boldsymbol{x}$ is a minimizer, all $\boldsymbol{z}$ that are feasible for (2.5), i.e. such that $\|\boldsymbol{A}\boldsymbol{z} - \boldsymbol{y}\|_2 \leq \eta$, gives

$$\lambda\|\boldsymbol{x}\|_1 + \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 \leq \lambda\|\boldsymbol{z}\|_1 + \|\boldsymbol{A}\boldsymbol{z} - \boldsymbol{y}\|_2^2 \leq \lambda\|\boldsymbol{z}\|_1 + \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2$$

This gives us, after simplifying the above expression, that $\|\boldsymbol{x}\|_1 \leq \|\boldsymbol{z}\|_1$, and thus, $\boldsymbol{x}$ is also a minimizer of (2.5)

Next, let $\boldsymbol{x}$ be a unique minimizer of (2.5). We pick $\tau = \|\boldsymbol{x}\|_1$. To show that $\boldsymbol{x}$ is a unique minimizer of (2.7), pick a $\boldsymbol{z} \neq \boldsymbol{x}$ such that $\|\boldsymbol{z}\|_1 \leq \tau$. Because $\|\boldsymbol{z}\|_1 \leq \|\boldsymbol{x}\|_1$ and $\boldsymbol{x}$ is the unique minimizer of (2.5), $\boldsymbol{z}$ cannot be a feasible point. Thus $\|\boldsymbol{A}\boldsymbol{z} - \boldsymbol{y}\|_2 > \eta \geq \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2$, and $\boldsymbol{x}$ is a minimizer of (2.7)

Finally, let $\boldsymbol{x}^\sharp$ be a minimizer of (2.7). This optimization problems is equivalent to

$$\min_{\boldsymbol{z}\in\mathbb{C}^N} \|\boldsymbol{A}\boldsymbol{z}-\boldsymbol{y}\|_2^2 \quad \text{subject to } \|\boldsymbol{z}\|_1 \leq \tau$$

The Lagrange function of this problem is

$$L(\boldsymbol{x},\nu) = \|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{y}\|_2^2 + \nu(\|\boldsymbol{x}\|_1 - \tau)$$

We have strong duality, and therefore there exist a dual optimal $\nu^\sharp$. The saddle point property implies $L(\boldsymbol{x}^\sharp,\nu^\sharp) = L(\boldsymbol{x},\nu^\sharp)$ for all $\boldsymbol{x}$, and so $\boldsymbol{x}^\sharp$ minimizes the mapping $\boldsymbol{x} \mapsto L(\boldsymbol{x},\nu^\sharp)$.

With $\boldsymbol{x}^\sharp$ being a minimizer of the Lagrange function, it will also minimize (2.6), with $\lambda = \nu^\sharp$, because the constant term $-\nu^\sharp\tau$ does not affect the optimal solution, and can therefore be discarded. By omitting the squaring of the ℓ_2-norm, we have reached out conclusion. ■

Remark 2.8. Claiming that the optimization problems discussed in Theorem 2.7 are equivalent is a bit of a misnomer, and a word of caution is in order. Most notably, we observe that the problems have vastly different sets of feasible points. Thus, we in general risk getting a infeasible problem if we try to covert from one to the other [FR13, p. 563].

In addition, it is important to note that the selection of the parameters in the proof is directly related to the minimizer(s). It also depends implicitly on $\boldsymbol{A}$ and $\boldsymbol{y}$, which makes it clear that this theorem is only applicable on a case by case basis, and does not provide any general way to convert between the problems. In fact, we have to find the minimizer before converting the optimization problem, which defeats the purpose. Thus tuning of the parameters must be done in each case separately. For our purposes, this theorem is only useful to see that the problems are related, and is otherwise of little practical interest.

CHAPTER 3

Inverse problems in medical imaging

Inverse problems are common in medical imaging. Instead of directly sampling in the spatial dimension, some form of linear measurements are often used. Thus, we have measurements on the form

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} \tag{3.1}$$

where the matrix $\boldsymbol{A} \in \mathbb{C}^{m \times N}$ models the sampling modality, and $\boldsymbol{x} \in \mathbb{C}^N$ is the unknown signal that we want to recover.

Clearly, if $m = N$ and $\boldsymbol{A}$ is invertible, recovering $\boldsymbol{x}$ is quite straightforward. However, obtaining all of the required measurements might be an expensive or time consuming process. Therefore, we are motivated to recover $\boldsymbol{x}$ from fewer measurements.

This needs to be done in a stable way, so that we can guarantee conditions where the signal will be successfully recovered. These guarantees are important for doctors to give a correct diagnosis. We risk false negatives, as noise and artifacts can obscure e.g. a tumor, and likewise false positives, as artifacts might resemble tumors.

3.1 Magnetic resonance imaging

Already in the first papers introducing compressive sensing [CRT06; Don06], magnetic resonance imaging (MRI) was proposed as an application. To motivate our mathematical model, we will look at a simple description of MRI, based on [Cur+13; FR13; Lus+08].

Unlike digital photography, where the image only takes a moment to be captured, imaging with MRI is a quite time consuming process. Patients are expected to lie still for several minutes, or even hours as the measurements are taken. The reason is that a large number of measurements are needed when sampling with a strategy designed according to the Shannon–Nyquist sampling theorem. Each measurement is relatively time consuming, and the entire process might take a large amount of time. This relationship between time and resolution motivates application of compressive sensing to MRI.

What makes MRI possible, is the fact that the human body in large parts consist of water and fat, which both contains a lot of hydrogen. The positively charged protons found in the nuclei of these hydrogen atoms have a spin with precession (i.e. the axis of rotation also rotates). This gives each proton a magnetic moment.

MRI aims to visualise e.g. the proton density by detecting their magnetic moments. Unfortunately, the orientations of the rotations is random, and the net magnetic moment for the body is therefore zero. A strong static magnetic field is applied to align the spins. This causes the protons to precess either parallel or anti-parallel to the static field. Depending on the orientation, the protons are said to be in the low or high energy state, respectively.

A larger portion of the protons are in the low energy state than the high energy state, yielding a small magnetic moment in the direction of the magnetic field. Because the protons precess out of phase, there is no magnetic moment transverse to the static field. The net magnetic moment is therefore parallel to the static field and not detectable.

To obtain a detectable magnetic moment, we excite the protons with a Radio Frequency (RF) signal. If the frequency of the signal matches the precession frequency of the protons, the system will *resonate*. Some protons in the low-energy state will "flip" and go to the high-energy state. This cancels the magnetic moment parallel to the static field. In addition, the protons will start spinning in phase, yielding a magnetic moment rotating transverse to the static field. This spinning magnetic field can be detected by the current it induces in a receiver coil. When the RF signal is turned off, the above process occur in reverse. The protons precess out of phase, and the protons moved to the high-energy state will go back to the low-energy state.

An immediate problem is that we don't know where on the imaging subject the RF signal is coming from. Luckily, this can be found by utilizing the fact that the precession frequency is proportional to the strength of the magnetic field. We can therefore superimpose additional magnetic fields, referred to as gradient fields, that vary linearly with position. Thus, different parts of the subject will resonate with different frequencies. We are then able to use the gradient fields to determine where the RF signals are coming from. We denote the gradient by $\boldsymbol{G}\colon [0, T] \to \mathbb{R}^3$. The precession frequency then becomes, according to Larmors equation

$$\omega(\boldsymbol{z}) = \kappa(B + \langle \boldsymbol{G}, \boldsymbol{z} \rangle), \quad \boldsymbol{z} \in \mathbb{R}^3$$

Where κ is a physical constant, and we assume one gradient field in each Cartesian direction in the 3D case. We are able to code the location in space because the gradients induce a location-dependent linear phase dispersion. The total signal equation becomes

$$s(t) = \int_{\mathbb{R}^3} m(\boldsymbol{r}) e^{-i2\pi \langle \boldsymbol{k}(t), \boldsymbol{r} \rangle} \, d\boldsymbol{r} \tag{3.2}$$

Where $\boldsymbol{k}(t)$ is given as

$$\boldsymbol{k}(t) = \kappa \int_0^t \boldsymbol{G}(\tau) \, d\tau$$

We recognize Equation (3.2) as the 3D-Fourier transform of $m(\boldsymbol{r})$ at the spatial frequency $\boldsymbol{k}(t)$.

Depending on what we set $\boldsymbol{G}(t)$ to be, we decide what frequencies we sample. Setting $\boldsymbol{G}(t)$, and by extension $\boldsymbol{k}(t)$. The acquisition process consists of repeating the process of exciting the protons, and recording the signal L times along the L curves $\boldsymbol{k}_1, \ldots, \boldsymbol{k}_L$.

A common discretization is sampling along lines in the Cartesian spatial frequency space. The discretization of these lines means sampling in a Cartesian grid. Writing out Equation (3.2) in this case, gives us the Discrete Fourier Transform (DFT), and the image can be recovered by calculating the Inverse Discrete Fourier Transform (IDFT).

3.2 The mathematical model

In the previous section, we saw that if we sample cleverly, the process of obtaining the MRI signal is essentially taking the Fourier transform of the desired signal. Therefore, our mathematical MRI machine is going to be the DFT.

Definition 3.1 (The unitary DFT matrix)**.** Let $N \in \mathbb{N}$. The discrete Fourier transform (DFT) matrix is the matrix $\boldsymbol{F} \in \mathbb{C}^{N \times N}$ with entries given as

$$(\boldsymbol{F})_{nk} = \frac{1}{\sqrt{N}} e^{-2\pi nk/N}$$

Further, we will formalize the sense of subsampling. We define a general sampling pattern.

Definition 3.2 (Sampling pattern)**.** For an $N \in \mathbb{N}$, we define a sampling pattern as a subset Ω of $[N]$.

For a given sampling pattern Ω, we define the projection matrix $\boldsymbol{P}_\Omega$ as the rows from the identity matrix $\boldsymbol{I} \in \mathbb{C}^{N \times N}$ that are indexed by Ω.

The setup for subsampled MRI we are going to consider is as follows:

Definition 3.3 (Setup for MRI)**.** For $N \in \mathbb{N}$, pick a sampling pattern $\Omega \subseteq [N]$ with $|\Omega| = m$. We set the measurement matrix for MRI to be

$$\boldsymbol{A} \coloneqq \boldsymbol{P}_\Omega \boldsymbol{F} \in \mathbb{C}^{m \times N}$$

CHAPTER 4

Compressive sensing for medical imaging

This chapter considers the suitability of compressive sensing when it comes to medical imaging. In particular, we consider the MRI setup described in the previous chapter. We first investigate how we are going to satisfy the sparsity assumption made in Chapter 2. We also discuss problems arising from our setup, and how to adapt the compressive sensing theory to obtain successful recovery.

4.1 Sparsity in images using wavelets

A major challenge when applying compressive sensing to imaging problems, is that images rarely are sparse in the standard basis. Sparse images are not desirable, as zeros in the standard basis are manifested as black pixels. If an image contains a lot of black pixels, they will rarely contain useful information.

The Discrete Wavelet Transform (DWT) comes to our rescue. The DWT is applied in image compression to achieve a sparse representation, allowing us to keep only the important components of the image. A notable example of this is the JPEG-2000 standard [SCE01].

Multiresolution analysis

Instead of representing a function as a series of "global" sinusoidal waves, like the Fourier transform does, wavelet decomposition also lets us see *where* in the signal we have the different components. This allows us to approximate the signal better with fewer components, and leads to better sparsity. This is done by considering a *scaling function* $\phi\colon \mathbb{R} \to \mathbb{R}$, and we introduce the notation

$$\phi_{j,k} = 2^{j/2}\phi(2^j x - k) \tag{4.1}$$

By varying j and k, we translate and dilate ϕ. We want to use this to obtain an orthonormal basis for $\mathcal{L}^2(\mathbb{R})$. For the remainder of the thesis, we will assume that we are working with orthonormal bases.

Before proceeding, we need the definition of a Riesz basis

Definition 4.1 (Riesz basis [Chu92]). The set $\{\phi_{j,k}\}$ is a Riesz basis basis for $\mathcal{L}^2(\mathbb{R})$ if the span of the functions are dense in $\mathcal{L}^2(\mathbb{R})$, and that there are constants A, B with $0 < A \leq B < \infty$ such that

$$A\|\{c_{j,k}\}\|_{\ell^2}^2 \leq \left\| \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} c_{j,k}\phi_{j,k} \right\|_2^2 \leq B\|\{c_{j,k}\}\|_{\ell^2}^2$$

For all $\{c_{j,k}\}$ such that $\|c_{j,k}\|_{\ell^2}^2 := \sum_{j\in\mathbb{Z}} \sum_{k\in\mathbb{Z}} |c_{j,k}|^2 < \infty$

A motivation for representing functions in bases formed by (4.1) is that it allows us to consider the signal at multiple levels of detail. This concept is formalized in the definition of *multiresolution analysis* given below.

Definition 4.2 (Multiresolution Analysis [Chu92; Mal09])**.** We call a sequence $\{V_j\}_{j\in\mathbb{Z}}$ of closed subspaces of $\mathcal{L}^2(\mathbb{R})$ a multiresolution anaylysis (MRA) if all of the following points hold.

- $V_j \subset V_{j+1}$ for all $j \in \mathbb{Z}$
- $f(x) \in V_j \iff f(2x) \in V_{j+1}$ for all $j \in \mathbb{Z}$
- $\bigcap_{j\in\mathbb{Z}} V_j = \{\mathbf{0}\}$
- $\text{closure}\left(\bigcup_{j\in\mathbb{Z}} V_j\right) = \mathcal{L}^2(\mathbb{R})$

And there exist a function $\phi \in V_0$ such that $\{\phi(x-n)\}_{n\in\mathbb{Z}}$ is a Riesz basis for V_0.

The term *scaling function* is defined as follows

Definition 4.3 (Scaling function [Chu92])**.** A function $\phi \in \mathcal{L}^2(\mathbb{R})$ is called a scaling function, if the subspaces $V_j \subset \mathcal{L}^2(\mathbb{R})$, defined by

$$V_j = \text{closure}\{\phi_{j,k} \mid k \in \mathbb{Z}\}, \quad j \in \mathbb{Z}$$

Gives rise to an MRA with $\{\phi(x-k) \mid k \in \mathbb{Z}\}$ as a Riesz basis of V_0. We say that ϕ generates a multi resolution analysis $\{V_j\}$ of $\mathcal{L}^2(\mathbb{R})$.

Because $V_j \subset V_{j+1}$, multiresolution analysis allows us to represent an $f \in V_j$ in a finer, more detailed space V_{j+1}. Going the other way, projecting functions from V_{j+1} onto V_j, we will lose information about f. To keep the information, we define the set W_j as the orthogonal complement of V_j in V_{j+1}. We can then decompose any $f \in V_{j+1}$ into one component in V_j and W_j. This gives the relation

$$V_{j+1} = V_j \oplus W_j$$

where $\oplus$ denotes the direct sum. As a consequence of this, we also have

$$\mathcal{L}^2(\mathbb{R}) = \text{closure}\left(\bigoplus_{j=-\infty}^{\infty} W_j \right)$$

Thus, we can represent any function both in terms of basis functions of $\{V_j\}_{j\in\mathbb{Z}}$, $\{W_j\}_{j\in\mathbb{Z}}$ or a combination of these function spaces.

The relation depends on the conjugate mirror filters, whose coefficients arise from expressing $\phi \in V_0$ in V_1 i.e.

$$\phi(x) = \sum_{k=-\infty}^{\infty} h[k]\sqrt{2}\phi(2x-k) \tag{4.2}$$

Where

$$h[k] = \left\langle \phi(x), \sqrt{2}\phi(2x-k) \right\rangle \tag{4.3}$$

Due to the relation between the spaces V_j and W_j, we are able to identify an orthonormal basis for W_j using the conjugate mirror filter h.

Theorem 4.4 ([Mal09, pp. 278–279])**.** *Let ϕ be a scaling function, h the corresponding conjugate mirror filter, and ψ be the function with the Fourier transform*

$$\hat{\psi}(2\omega) = \frac{1}{\sqrt{2}}\hat{\phi}(\omega)\hat{g}(\omega)$$

with

$$\hat{g}(\omega) = e^{-2\pi i\omega}\hat{h}^*(\omega + \frac{1}{2})$$

being the Fourier transform of g and h^* denoting the complex conjugate. Then, for any j, $\{\psi_{j,k}\}_{k\in\mathbb{Z}^2}$ is an orthonormal basis for $\mathcal{L}^2(\mathbb{R})$, where we define

$$\psi_{j,k} = 2^{j/2}\psi(2^j x - k)$$

We refer to ψ as the mother wavelet.

Vanishing moments

The main motivation behind applying wavelets to our problem is to achieve sparsity, we need to investigate under which conditions we have sparsity in wavelets. Multiresolution analysis allows us to repeatedly decompose our function into detail spaces. We will therefore find conditions where the coefficients of the functions in the detail spaces are zero. In other words that the inner products are zero.

$$\langle f, \psi_{k,n} \rangle \approx 0 \quad \forall \psi_{k,n}.$$

We consider inner products with $\psi_{k,n}$ because after a multilevel wavelet transform, a majority of the coefficients will belong to the detail spaces.

A key property in achieving this is the number of vanishing moments

Definition 4.5 (Vanishing moments [Mal09, p. 208])**.** We say that ψ as k vanishing moment if

$$\int_{-\infty}^{\infty} t^l \psi(t) dt = 0 \tag{4.4}$$

for $l = 0, \dots, k-1$

Equation (4.4) tells us that ψ having k vanishing moments is equivalent to ψ being orthogonal to polynomials up to degree $k-1$. Thus, the corresponding scaling function ϕ can be used to express polynomials up to degree $k-1$.

Because polynomials are good at approximating smooth functions, wavelets with several vanishing moments will also work well for approximation.

The discrete wavelet transform

We can now define the discrete wavelet transform

Definition 4.6 (Discrete wavelet transform)**.** Let $\Phi_j = \{\phi_{j,n}\}_{n\in\mathbb{Z}}$, and likewise $\Psi_j = \{\psi_{j,n}\}_{n\in\mathbb{Z}}$. The m-level *discrete wavelet transform* (DWT) of $f \in V_m$ is the change of basis from Φ_m to $(\Phi_0, \Psi_1, \dots, \Psi_{m-1})$. Similarly, we define the *inverse discrete wavelet transform* as the change of basis from $(\Phi_0, \Psi_1, \dots, \Psi_{m-1})$ to Φ_m

The next result shows how to compute the DWT. We define

$$a_j[k] = \langle f, \phi_{j,k}\rangle, \quad d_j[k] = \langle f, \psi_{j,k}\rangle$$

to denote the coefficients of f in the resolution and detail spaces V_j and W_j, respectively. In addition, we introduce the notation $\bar{h}[k] = h[-k]$ for any filter. The relationship between the coefficients of the coefficients of the spaces are given below.

Theorem 4.7 ([Mal09, p. 298])**.** *Let $f \in V_j$. The decomposition of f into V_{j-1} and W_{j-1} is given by*

$$a_{j-1}[k] = \sum_{n=-\infty}^{\infty} h[n-2k]a_j[n] = a_j * \bar{h}[2k]$$
$$d_{j-1}[k] = \sum_{n=-\infty}^{\infty} g[n-2k]a_j[n] = a_j * \bar{g}[2k]$$

where $x * y$ denotes convolution.

The reconstruction back into V_j becomes

$$a_j[k] = \sum_{n=-\infty}^{\infty} h[k-2n]a_{j-1}[n] + \sum_{n=-\infty}^{\infty} g[k-2n]d_{j-1}[n]$$

In applications, we are rarely working directly with the continuous signal f. Rather, we have discrete samples. Instead of working on $\mathcal{L}^2(\mathbb{R})$, we scale the space down to $\mathcal{L}^2([0,1])$, and observe N equispaced samples $f(k/N)$ for $k = 0, 1, \dots, N-1$.

Because it makes the discrete wavelet transform easier to work with, we are going to assume that $N = 2^R$ for some $R \in \mathbb{N}$. Further, we will assume that $f(k/N) \approx \langle f, \phi_{k,R}\rangle$, which is a fair approximation if the scaling functions generate an orthonormal basis, as the inner product will be a weighted average [Mal09, p. 301]. However, it is worth noting that we in fact are committing what is known as *the wavelet crime*. That is, we are assuming that the function samples are coefficients in some space V_R. Usually, we won't run into serious problems because of this, but we might risk serious errors [AH16].

Performing the DWT on the finite $a_R[k]$ will give us problems on the edges. The easiest way to deal with this, is to extend the signal periodically, or equivalently, replace the convolutions in Theorem 4.7 with circular convolutions.

Theorem 4.7 motivates a fast and straightforward way of computing the DWT and IDWT. Indeed, if we have the coefficients a_j, the computation of the

DWT is applying the filters $\bar{h}$ and $\bar{g}$, and downsample, keeping only every other entry. Likewise, we see that the IDWT can be computed by first upsampling the coefficient vectors, and then apply the filters h and g. The complexity of this algorithm becomes $\mathcal{O}(N)$.

Setup for compressive sensing

With the knowledge of wavelets' role in achieving sparsity in our signal, we will aim to instead recover the wavelet coefficients. When these are successfully recovered, we can easily find the actual signal by computing the IDWT.

The optimization problem we want to solve, as previously discussed compressive sensing becomes, according to e.g. [RHA14]

$$\min_{\boldsymbol{z}\in\mathbb{C}^N} \|\boldsymbol{z}\|_1 \quad \text{subject to} \quad \|\boldsymbol{A}\boldsymbol{W}^{-1}\boldsymbol{z} - \boldsymbol{y}\|_2 \leq \eta \tag{4.5}$$

Which is Equation (2.5) where the constraint does not only include the sensing matrix $\boldsymbol{A}$, but also the IDWT matrix $\boldsymbol{W}^{-1}$. It is worth noting that the measurement vector $\boldsymbol{y}$ is obtained only from multiplication with $\boldsymbol{A}$, and not with $\boldsymbol{A}\boldsymbol{W}^{-1}$.

Next, we consider this setup, and see how well it fits the previously discussed compressive sensing theory, and potentially how it can be improved.

4.2 Applying classical compressive sensing to MRI

In the classical compressive sensing theory, the fundamental requirement for successful recovery is *sparsity*. No assumptions about the non-zero entries of the signal $\boldsymbol{x}$ are made, and thus theory has been developed for *uniform random subsampling*.

As a measure of the suitability of the measurement matrix $\boldsymbol{A}$ for successful recovery, we have the *coherence*. In the literature, many different definitions of the coherence are used, such as Definition 2.4. We will consider the following definition:

Definition 4.8 (coherence)**.** Let $\boldsymbol{U} = (u_{ij}) \in \mathbb{C}^{N\times N}$ be a unitary matrix. The *coherence* of $\boldsymbol{U}$ is

$$\mu(\boldsymbol{U}) = \max_{i,j} |u_{ij}|^2 \in [1/N, 1]$$

By assuming that $\boldsymbol{U}$ is unitary, the coherence $\mu(\boldsymbol{U})$ tells us something about how much the values of $\boldsymbol{U}$ are "spread out". Further, this plays an important role when it comes to how many samples we need to recover the signal with high probability.

Theorem 4.9 ([AH16])**.** *Let $\boldsymbol{U} \in \mathbb{C}^{N\times N}$ be a unitary matrix, and set $\varepsilon \in (0,1)$. Assume that $\boldsymbol{x} \in \mathbb{C}^N$ has support $S \subseteq [N]$ with $|S| = s$.*

If the number of samples $m \in \mathbb{N}$ satisfies

$$m \gtrsim \mu(\boldsymbol{U}) \cdot N \cdot s \cdot \left(1 + \log \varepsilon^{-1}\right) \cdot \log N$$

And the elements of $\Omega \subseteq [N]$ with $|\Omega| = m$ are chosen uniformly at random.

Then, $\boldsymbol{x}$ can be recovered with probability at least $1-\varepsilon$ from measurements $\boldsymbol{y} = \boldsymbol{P}_\Omega \boldsymbol{U}\boldsymbol{x}$ by solving

$$\min_{\boldsymbol{z}\in\mathbb{C}^N} \|\boldsymbol{z}\|_1 \quad \textit{subject to} \quad \|\boldsymbol{y} - \boldsymbol{P}_\Omega \boldsymbol{U}\boldsymbol{z}\|_2 \leq \eta$$

According to Theorem 4.9, low coherence is crucial to recover $\boldsymbol{x}$ successfully from few samples. Unfortunately, the sensing operators in medical imaging, such as the Fourier-Wavelet setup discussed above, are often coherent, meaning that $\mu(\boldsymbol{U}) = 1$. The theory then says that full sampling is required. The classical compressive sensing theory is therefore not suitable for this type of problems.

4.3 Changing the sampling operator

In Chapter 2 we developed theory that guaranteed recovery of sparse vectors. However, as we have seen, the theory does not work well when applied to MRI due to the high coherence of the sensing matrix.

We know that large classes of matrices with low coherence exist. For instance matrices with entries draw from a Gaussian distribution [FR13, p. 279]. We might be tempted to use such matrices instead of the previously described setup. Unfortunately, this is not possible due to the following reasons:

Physical limitations

We saw in Chapter 3, that the sensing matrix arises from the sampling modality. Magnetic resonance imaging works by sampling the continuous Fourier transform, and it is reasonable to choose the sampling strategy to be such that the resulting matrix is the DFT. Although we have some freedom to select which Fourier samples we want, this is not flexible enough to represent any general matrix.

Thus, while the theory allows us to set the sampling operator to whatever we want, we can't always do this in practice due to physical limitations of the sampling modality.

Computational issues

A more general problem with using arbitrary matrices in compressive sensing is directly related to computing. Most notably when it comes to storage. For instance, assume that we are working with images with a size of one megapixel (10^6 pixels). Then the signal is a vector $\boldsymbol{x} \in \mathbb{R}^{10^6}$. The sensing matrix must therefore have a million columns. Assuming that we sample only 12.5% of the rows, the resulting matrix takes 1 TB of memory, assuming that the entires are 64-bit floating point number. Working with matrices of this size quickly becomes difficult. They cannot be stored in memory. As this is only a moderately sized setup, the problem becomes even more evident when increasing the size or number of samples. This is both due to the storage problem, and the additional fact that a matrix-vector multiplication is computed in $\mathcal{O}(mN)$ time.

Using the DFT matrix, on the other hand, has a quite substantial advantage over general matrices. It can be computed efficiently, using the Fast Fourier Transform (FFT). We solve the problems mentioned above quite easily. Not only does the FFT *not* realize the Fourier matrix, which allows us to compute the DFT of even large signals without using much memory, but also does so quickly, in $\mathcal{O}(N \log_2 N)$ time. We also get similar advantages from using the discrete wavelet transform, as it can be computed in $\mathcal{O}(N)$.

4.4 Structured compressive sensing

We have seen that even though the signals we are trying to recover in MRI are sparse, the setup does not perform well according to the classical compressive sensing theory due to high coherence. However, our setup has the advantage that it makes computation possible. Therefore, we would rather improve upon the theory rather than modify the setup. Enter *structured compressive sensing*.

As a motivation, consider the wavelet transform of a signal. The signal is sparse, but the sparsity differs in different parts. In the wavelet case, we also have a clear structure. The wavelet transform is sparse for the detail spaces, while the nonzero coefficients exist among the low resolution coefficients, causing the image to not be sparse at all in this region. Below, we will generalize several of the definitions from Chapter 2, to allow structure. Therefore, we can for instance move away from the notion of global sparsity to *sparsity in levels*.

Definition 4.10 (Sparsity in levels)**.** Let $\boldsymbol{x} \in \mathbb{C}^N$. For $r \in \mathbb{N}$, set the sparsity levels $\boldsymbol{M} = (M_1, \dots, M_r) \in \mathbb{N}^r$ and sparsities $\boldsymbol{s} = (s_1, \dots, s_r) \in \mathbb{N}^r$, with $s_k \leq M_k - M_{k-1}, k = 1, \dots, r$, where $M_0 = 0$. We say that $\boldsymbol{x}$ is $(\boldsymbol{s}, \boldsymbol{M})$-sparse if, for each $k = 1, \dots, r$, the *sparsity band*

$$\Delta_k = \operatorname{supp}(x) \cap \{M_{k-1} + 1, \dots, M_k\}$$

satisfies $|\Delta_k| \leq s_k$. We denote the set of $(\boldsymbol{s}, \boldsymbol{M})$-sparse vectors by $\Sigma_{\boldsymbol{s},\boldsymbol{M}}$.

Likewise, this motivates a similar definition for structured sampling.

Definition 4.11 (Multilevel random subsampling [AL])**.** For $r \in \mathbb{N}$ let $\boldsymbol{N} = (N_1, \dots, N_r)$ where $1 \leq N_1 < \dots < N_r = N$ and $\boldsymbol{m} = (m_1, \dots, m_r)$, where $m_k \leq N_k - N_{k-1}$ for $k = 1, \dots, r$ and $N_0 = 0$. For each $k = 0, \dots, r$ let $t_{k,1}, \dots, t_{k,m_k}$ be drawn uniformly and independently from $\{N_{k-1} + 1, \dots, N_k\}$, and set $\Omega_k = \{t_{k,1}, \dots, t_{k,m_k}\}$. If $\Omega = \Omega_{\boldsymbol{N},\boldsymbol{m}} = \Omega_1 \cap \dots \cap \Omega_r$ we refer to Ω as an $\boldsymbol{N}, \boldsymbol{m}$-multilevel subsampling scheme.

Remark 4.12. We emphasise that Definition 4.11 allows us to sample the same point multiple times. Sampling uniformly and independently in each level, makes the analysis easier [AH19, pp. 190–191] , as we are working with several independent, identically distributed random variables. For practical purposes, on the other hand, this is not desirable. Because we allow ourselves to only take a limited number of samples, we clearly would want to obtain as much information as possible. Repeating the same point twice is therefore wasteful.

Not only is there a structure in the sparsity of the signal, but also in the coherence of the sensing matrix. At this point, we have only considered the *global coherence*. Because this considers the matrix as a whole, we do not know if there is a certain part of the matrix that gives the high coherence, or if high valued entries are scattered around different places in the matrices.

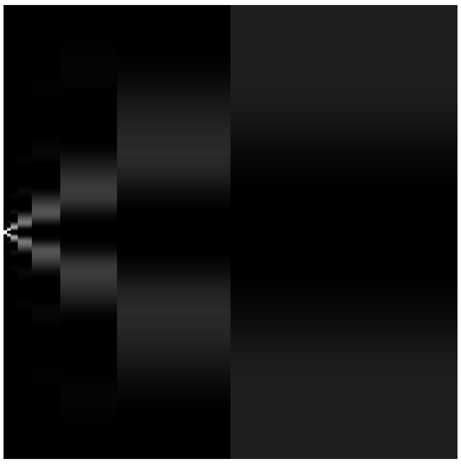


Figure 4.1: The coherence between the Fourier matrix and the DB4 wavelet bases

As seen in Figure 4.1 only a few rows in the sensing matrix contributes to the high coherence. To investigate this further, one might want to look at the coherence of submatrices. This motivates *coherence in levels*

Definition 4.13 (Coherence in levels)**.** Let $\boldsymbol{N} = (N_1, \dots, N_r)$ be the sampling levels, and $\boldsymbol{M} = (M_1, \dots, M_r)$ be the sparsity levels. If $\boldsymbol{U} \in \mathbb{C}^{N \times N}$ is an isometry, The $(k,l)^{\text{th}}$ local coherence is given as

$$\mu_{k,l} = \max\Big\{|U_{ij}|^2 : i = N_{k-1}+1, \dots, N_k, j = M_{l-1}+1, \dots, M_l\Big\}$$

As previously mentioned, several definitions can be found for the global coherence. This is also the case for local coherence in levels. The definitions are often similar, but not necessarily equivalent.

The above definition is chosen because we are concerned with *uniform recovery guarantees*. We will investigate under which conditions we can recover any $(\boldsymbol{s}, \boldsymbol{M})$-sparse vector.

A sufficient requirement in the classical compressive sensing theory is the *restricted isometry property* (RIP). Below is a more general definition, which allows levels.

Definition 4.14 (RIP in levels [AL])**.** Let $\boldsymbol{s}, \boldsymbol{M}$ be a sparsity pattern, and $\boldsymbol{A} \in \mathbb{C}^{m \times N}$ with $m < N$. The $\boldsymbol{s}^{\text{th}}$ restricted isometry constant in levels (RICL) $\delta_{\boldsymbol{s},\boldsymbol{M}}$ is the smallest $\delta \geq 0$ such that

$$(1-\delta)\|\boldsymbol{x}\|_2^2 \leq \|\boldsymbol{A}\boldsymbol{x}\|_2^2 \leq (1+\delta)\|\boldsymbol{x}\|_2^2, \quad \forall \boldsymbol{x} \in \Sigma_{\boldsymbol{s},\boldsymbol{M}}$$

$\boldsymbol{A}$ is said to satisfy the restricted isometry property in levels (RIPL) of order $(\boldsymbol{s}, \boldsymbol{M})$ if $0 < \delta_{\boldsymbol{s},\boldsymbol{M}} < 1$.

Indeed, this gives us stable and robust recovery of $(\boldsymbol{s}, \boldsymbol{M})$-sparse vectors.

Theorem 4.15 ([AL]). *Suppose that $\boldsymbol{A} \in \mathbb{C}^{m\times N}$ satisfies the RIPL of order $(2\boldsymbol{s}, \boldsymbol{M})$ with constant*

$$\delta_{(2\boldsymbol{s},\boldsymbol{M})} < \frac{1}{\sqrt{r(\sqrt{\rho}+1/4)^2+1}}$$

where

$$\rho = \rho_{(\boldsymbol{s},\boldsymbol{M})} = \max_{k,l=1,\dots,r} \{s_k/s_l\}$$

Let $\boldsymbol{x} \in \mathbb{C}^N$ and $\boldsymbol{y} = \boldsymbol{Ax}+\boldsymbol{e}$ where $\|e\|_2 \le \eta$. Then, for any minimizer $\boldsymbol{x}^* \in \mathbb{C}^N$ of

$$\min_{\boldsymbol{z}\in\mathbb{C}^N} \|\boldsymbol{z}\|_1 \quad \textit{subject to} \quad \|\boldsymbol{Az}-\boldsymbol{y}\|_2 \le \eta$$

we have

$$\|\boldsymbol{x}^* - \boldsymbol{x}\|_1 \lesssim \sigma_{\boldsymbol{s},\boldsymbol{M}}(\boldsymbol{x}) + \sqrt{s}\eta \tag{4.6}$$

and

$$\|\boldsymbol{x}^* - \boldsymbol{x}\|_2 \lesssim \left(1+(r\rho)^{1/4}\right)\frac{\sigma_{\boldsymbol{s},\boldsymbol{M}}(\boldsymbol{x})}{\sqrt{s}} + (1+(rp)^{1/4})\eta \tag{4.7}$$

where $s = s_1 + \cdots + s_r$

Theorem 4.15 gives us an interesting result. Looking at the performance guarantees (4.6) and (4.7), we see the error is bounded by the best $(\boldsymbol{s},\boldsymbol{M})$-sparse approximation, and the level of noise η. This means that in the case where $\boldsymbol{A}$ satisfies the RIPL, $\boldsymbol{x}$ is $(\boldsymbol{s},\boldsymbol{M})$-sparse, and there is no measurement error, we are able to recover $\boldsymbol{x}$ exactly.

Another important thing to take away from this theorem, is how the error changes in terms of the noise level η. For practical applications, we must allow some error, but we need to ensure that this does not have drastic effects on the output. Indeed, we see that the error scales linearly with the error, which makes the output easier to trust.

The constant ρ depends on the sparsity pattern. A large value for ρ means that at least two sparsity levels have a very different number of nonzero entries. This means that the nonzero entries are not very uniformly distributed. This leads to a stricter requirement, namely that the RIPL constant $\delta_{\boldsymbol{s},\boldsymbol{M}}$ becomes smaller. At the same time, the error estimate becomes bigger, and the recovery can get worse.

We will now need to make sure that our setup satisfies the RIPL. Let $\boldsymbol{U} \in \mathbb{C}^{N\times N}$ be a unitary matrix. We consider a matrix on the form

$$A = \begin{bmatrix} 1/\sqrt{p_1}\boldsymbol{P}_{\Omega_1}\boldsymbol{U} \\ 1/\sqrt{p_2}\boldsymbol{P}_{\Omega_2}\boldsymbol{U} \\ \vdots \\ 1/\sqrt{p_r}\boldsymbol{P}_{\Omega_r}\boldsymbol{U} \end{bmatrix} \in \mathbb{C}^{m\times N} \tag{4.8}$$

Where $p_k = \frac{m_k}{N_k - N_{k-1}}$ for $k = 1,\dots,r$, and $m = m_1 + m_2 + \cdots + m_r$

Theorem 4.16 (Subsampled matrices and the RIPL I [AL]). *Let $\boldsymbol{U} \in \mathbb{C}^{N\times N}$ be a unitary matrix, $r \in \mathbb{N}$ be the number of sampling levels, and $0 < \varepsilon, \delta < 1$. Let*

$\Omega = \Omega_{\boldsymbol{N},\boldsymbol{m}}$ be an $(\boldsymbol{N}, \boldsymbol{m})$-multilevel sampling scheme, and $\boldsymbol{M}$ and $\boldsymbol{s}$ be sparsity levels and local sparsities respectively. Suppose that

$$m_k \gtrsim \delta^{-2} \cdot (N_k - N_{k-1}) \cdot \left(\sum_{l=1}^{r} \mu_{k,l} \cdot s_l \right) \cdot \Big(r \log(2m) \log(2N) \log^2(2s) + \log\Big(\varepsilon^{-1}\Big) \Big)$$

For $k = 1, \dots, r$ where $m = m_1 + \cdots + m_r$. Then with probability at least $1 - \varepsilon$, the matrix Equation (4.8) satisfies the RIPL of order $(\boldsymbol{s}, \boldsymbol{M})$ with constant $\delta_{\boldsymbol{s},\boldsymbol{M}} \leq \delta$.

A more general version of this theorem, that allows fully sampling the first r_0 levels, is given below.

Theorem 4.17 (Subsampled matrices and the RIPL II [AL])**.** *Let $\boldsymbol{U} \in \mathbb{C}^{N\times N}$ be a unitary matrix, $r \in \mathbb{N}$ be the number of sampling levels, $0 < \varepsilon, \delta < 1$, and $0 \leq r_0 \leq r$. Let $\Omega = \Omega_{\boldsymbol{N},\boldsymbol{m}}$ be an $(\boldsymbol{N}, \boldsymbol{m})$-multilevel sampling scheme, and $\boldsymbol{M}$ and $\boldsymbol{s}$ be sparsity levels and local sparsities respectively. Suppose that*

$$m_k = N_k - N_{k-1}, \quad k = 1, \dots, r_0$$

and

$$m_k \gtrsim \delta^{-2} \cdot (N_k - N_{k-1}) \cdot \left(\sum_{l=1}^{r} \mu_{k,l} \cdot s_l \right) \cdot \Big(r \log(2\hat{m}) \log(2N) \log^2(2s) + \log\Big(\varepsilon^{-1}\Big) \Big)$$

For $k = r_0 + 1, \dots, r$ where $\hat{m} = m_{r_0+1} + \cdots + m_r$. Then with probability at least $1 - \varepsilon$, the matrix Equation (4.8) satisfies the RIPL of order $(\boldsymbol{s}, \boldsymbol{M})$ with constant $\delta_{\boldsymbol{s},\boldsymbol{M}} \leq \delta$.

This theorem is motivated by the previously discussed structure The first few levels of the wavelet transform is rarely sparse. In fact, the opposite is often the case. Therefore, it might be a good idea to fully sample the first levels.

This can be confirmed by looking at the total number of required samples as in [AL]. This assumes the setup with the Fourier matrix and the Haar wavelet. Theorem 4.16 gives us, with $m = m_1 + \cdots + m_r$

$$m \geq C_1 \delta^{-2} \cdot \left(\sum_{i=1}^{r} s_i \right) \cdot \mathcal{L} \tag{4.9}$$

While Theorem 4.17 gives

$$m \geq N_{r_0} + C_2 \delta^{-2} \cdot \left(\sum_{i=r_0+1}^{r} s_i \right) \cdot \mathcal{L} \tag{4.10}$$

In these two equations, the constants C_1, C_2 are independent of all parameters, and $\mathcal{L}$ contains all the log factors. In addition N_{r_0} is the total number of entries in the first r_0 levels.

Assuming $s_i = N_i$ for $i = 1, \dots, r_0$, the estimate in Equation (4.10) will be smaller than in Equation (4.9) because we don't have to multiply the sparisty (i.e. N_{r_0}) with the log factors and global constants. In fact, this also outperforms incoherent sampling (e.g. sampling with gaussian matrix).

This demonstrates that there exist matrices that satisfies the RIPL, and that they are indeed able to recover the signal successfully. Although initially, compressive sensing seemed difficult to use on MRI, Theorem 4.15 and Theorem 4.17 have shown that we at least theoretically are able to apply compressive sensing to MRI with success.

CHAPTER 5

Deep learning for medical imaging

We will now provide a brief overview over the basics of deep learning in the context of neural networks applied to inverse problems in medical imaging. We will consider the basic definitions and theory that motivates the feasibility of this approach. Finally, we consider approaches for training neural methods, and related problems.

5.1 Definitions of dense and convolutional neural networks

An alternative approach to solving the inverse problem, is *deep learning*. In this thesis, we will consider deep learning in the context of deep neural networks, which are artificial neural networks, defined in Definition 5.1, with several layers. As with compressed sensing, the goal is for us to be able to recover an image from undersampled Fourier measurements.

Neural networks form a very flexible class of functions. Theoretically, we are able to approximate large classes of functions, which is useful when we know little or nothing about what the function we are approximating looks like, apart from a limited number of samples. According to [Pin99], there are no universally accepted definition of neural networks. We will work with the one given below, as it is suitable for the theoretical results we are going to show, and well suited for the extensions we are going to perform in Chapter 7.

Definition 5.1. Let $L, d, N_1, N_2, \dots, N_L \in \mathbb{N}$. A neural network is a mapping $\Phi\colon \mathbb{R}^d \to \mathbb{R}^{N_L}$ given by

$$\Phi(\boldsymbol{x}) = W_L(\rho(W_{L-1}(\rho(\dots\rho(W_1(x))))))$$

Where $\rho\colon \mathbb{R} \to \mathbb{R}$ is a non-linear function that extended to be applied componentwise on vectors, and $W_\ell\colon \mathbb{R}^{N_{\ell-1}} \to \mathbb{R}^{N_\ell}$ is an affine mapping given by

$$W_\ell(\boldsymbol{x}) = \boldsymbol{A}_\ell \boldsymbol{x} + \boldsymbol{b}_\ell$$

The mapping W_ℓ is referred to as the ℓ-th layer. And the entries of $\boldsymbol{A}_\ell$ and $\boldsymbol{b}_\ell$ are the *weights* of that layer.

The general neural network defined above, is often referred to as a *dense neural network* (DNN). The reason being that the weight matrices can be dense. Interpreting the networks as nodes (neurons) organized in layers where each

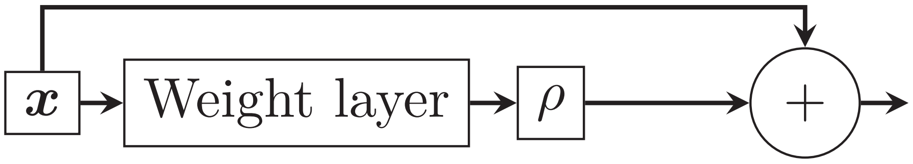


Figure 5.1: Skip connection

node in one layer is connected to every node in the next layer, also motivates this term as the network will form a dense graph. For imaging purposes, it is more common to use a subclass of DNNs, where the matrices $\boldsymbol{A}_\ell$ are restricted to being convolutional operators. A neural network of this type is called a *convolutional neural network* (CNN). This class of networks is often used in imaging because it scales better with the input size in terms of storage size. We don't need to store large matrices, only the filter kernel. Thus, we can work quite efficiently with even rather large images.

Remark 5.2. Modern Deep Learning largely deviates from the definition of neural networks given in Definition 5.1. The definition above works well if we want to consider e.g. theoretical analysis. However, there are operations and structures used today that can not be expressed by the classical definition. One such operation is *max pooling*, which behaves like a convolutional operator, but pick the greatest element instead of computing the inner product with a filter kernel. This operator is not linear, and does therefore not fit the definition.

A structural deviation is *skip connection*. Skip connections mean that in addition to feeding input through normal weight layers like the classical definition, and the output is combined with the input in a way. This is illustrated in Figure 5.1.

Because of these deviations, it would be easier to model modern neural networks in a more general way. One alternative is with a graph definition, where the vertices are operations, and the edges direct the input and output. We would then define neural networks to be Directed Acyclic Graphs (DAG), but this easily becomes too general to be useful for our purposes.

5.2 Neural networks as function approximators

Few theoretical results exist for neural networks. Among those that exist, are *universal approximation theorems*. It turns out that any continuous function can be approximated arbitrarily well. Several different variants of these results exist, for a variety of activation functions, and motivates the feasibility of neural using neural networks in medical imaging

We are going to consider the result found in [Cyb89]. This paper consider neural networks containing a single layer, and uses a fairly flexible class of functions. These function are known as *sigmoidal functions* and are defined in Definition 5.3.

Definition 5.3 ([Cyb89])**.** Sigmoidal activation functions are continuous functions $\sigma\colon \mathbb{R} \to \mathbb{R}$ such that

$$\sigma(t) \to \begin{cases} 1 & \text{as } t \to \infty \\ 0 & \text{as } t \to -\infty \end{cases} \tag{5.1}$$

In the literature, this class of functions is defined in a variety of ways [Pin99]. In our case, we will only demand continuity in addition to the limits given in Definition 5.3.

Further, we define $I_n = [0,1]^n$ and let the set $C(I_n)$ be the set of real-valued continuous functions on I_n. In addition, we need the space $M(I_n)$ of finite signed Borel measures on I_n.

Instead of working directly with the sigmoidal function when proving the universal approximation theorem, we are going to use a more general class of functions, called discriminatory functions.

Definition 5.4 ([Cyb89])**.** We say that σ is discriminatory if for a measure $\mu \in M(I_n)$

$$\int_{I_n} \sigma(\boldsymbol{y}^T \boldsymbol{x} + \theta) d\mu(x) = 0$$

For all $\boldsymbol{y} \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$ implies that $\mu = 0$.

This generalization is supported by the following lemma

Lemma 5.5. *Sigmoidal function are discriminatory.*

Proof. let σ be a sigmoidal function. To show that it is discriminatory, we employ contrapositivity. Assume that the measure μ is nonzero. It is then enough to show that we can pick $\boldsymbol{y} \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$ so that the integral

$$\int_{I_n} \sigma(\boldsymbol{y}^T \boldsymbol{x} + \theta) d\mu(\boldsymbol{x})$$

is different from zero.

Because σ is sigmoidal, which in means that $\sigma(t) \to 1$ as $t \to \infty$. We will attempt to find a set on which σ is strictly positive μ-almost everywhere, which means that the integral cannot be zero.

We can determine the existence of a set by using that the function converges to 1. We set $1 > \varepsilon > 0$. There must be an x such that when $t > x$, we have $\sigma(t) \in (1-\varepsilon, 1+\varepsilon)$.

By setting $\theta > x$ and $\boldsymbol{y} = \boldsymbol{0}$, we have that

$$\left| \int_{I_n} \sigma(\theta) d\mu(\boldsymbol{x}) \right| \geq \left| \int_{I_n} \inf_{t > x} \sigma(t) d\mu(x) \right| \geq (1-\varepsilon)|\mu(I_n)| \neq 0$$

Which completes the proof. ■

We can now state and prove the universal approximation theorem for the functions given above.

Theorem 5.6 ([Cyb89])**.** *Let σ be any continuous sigmoidal function. Then finite sums of the form*

$$G(x) = \sum_{j=1}^{r} w_j \sigma(\boldsymbol{y}_j^T \boldsymbol{x} + b_j) \tag{5.2}$$

Are dense in $C(I_n)$.

Proof. We denote the set of one-layered neural networks on the form Equation (5.2) as $\mathcal{N}$. To show that this is dense in $C(I_n)$, we are going to show that the closure of $\mathcal{N}$ in $C(I_n)$ is equal to $C(I_n)$.

To show this, we are going to assume for contradiction that the closure R of $\mathcal{N}$ is a closed, proper subset of $C(I_n)$.

We can therefore apply the Hahn-Banach theorem. There exists a nontrivial bounded linear operator L on $C(I_n)$ such that $L(h) = 0$ for $h \in R$, and $L(h) \neq 0$ when $h \in C(I_n) \setminus R$.

By Riesz' Representation Theorem, there is a $\mu \in \boldsymbol{M}(I_n)$ that lets us write L on the form

$$L(h) = \int_{I_n} h(x) d\mu(x)$$

The mapping $\boldsymbol{x} \mapsto \sigma(\boldsymbol{y}^T \boldsymbol{x} + \theta)$ is in R, and we therefore have that

$$\int_{I_n} \sigma(\boldsymbol{y}^T \boldsymbol{x} + \theta) d\mu(x) = 0$$

But because the function also is discriminatory, this must mean that $\mu = 0$. This implies $L = 0$, which contradicts the nontriviality of L. Thus, R cannot be a proper subset of $C(I_n)$, and $\mathcal{N}$ must be dense in $C(I_n)$. ■

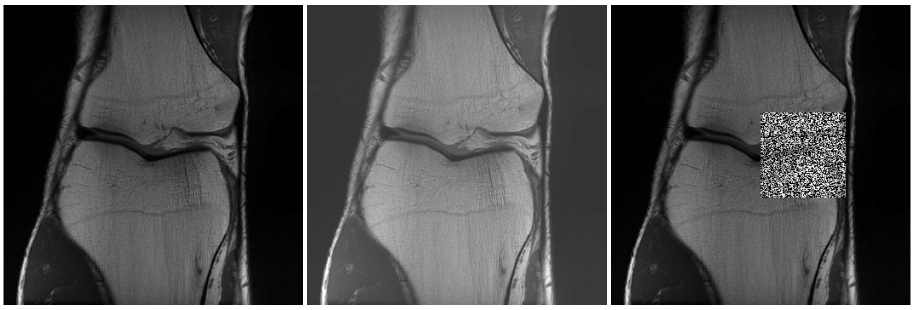
Significance and lack of practical usefulness

Density as proved in Theorem 5.6 is indeed useful, as it certainly is a necessary condition for arbitrary precision in our neural networks. On the other hand, the result only tells us that a suitable neural network exist, it does not tell us how to find a suitable network. Thus, finding the correct $\boldsymbol{y}$, θ and w_j for $j \in [r]$. In addition, we don't know a priori what the value of r must be. That is, even though the universal approximation theorem holds, there is no guarantee that we are able to find a good approximation for any fixed r [Pin99].

In particular, the theorem does not consider the feasibility of *learning*. Even though we theoretically are able to approximate any function, the proposed architecture might be unsuitable for any sort of training scheme. Indeed, architectures in deep learning, that have several layers, have turned out to perform better when training with methods explained in Section 5.3, compared to the single layer model that the theorem consider. Intuitively, this improvement can be explained by more freedom to create an abstract representation of the input, which allows for easier feature extraction.

5.3 Training neural networks

We would like to find a function $f \colon \mathbb{C}^m \to \mathbb{C}^N$ that maps the measurement vector $\boldsymbol{y}$ to the recovered image. We might be tempted to learn this mapping directly with a neural network. Unfortunately, this turns out to be quite



(a) Original image (b) Brightness adjusted (c) Noise in part of image

Figure 5.2: Different modifications of images can lead to misleading ℓ_2-loss.

difficult . A major source of problems is that the input vector contains Fourier measurements in the MRI setting, but we want the output in the standard basis. Therefore, the network essentially need to learn the inverse discrete Fourier transform (IDFT). This has been done with some success, but due to the aforementioned problems, we are severely limited in terms of image size. Doing this with CNNs, is even more difficult, because the IDFT must be expressed as a digital filter.

Learning the DFT has been done with some success in [Zhu+18]. The network has a few dense layers in the beginning, and the input is the measurement vector. A motivation behind this type of architecture, is desires of a network that can recover images independently of imaging techniques and sampling pattern. In addition, we want to find a way to more effectively represent the data in a low-dimensional space. However, as the number of weights increases with the input size, so does the storage space and computational time required. This limits the network to small resolutions.

By taking the adjoint of our measurements, in our case the IDFT of the measurement vector, we will get an image that resembles the desired result. However, due to the subsampling, we may have noise or artifacts we want to get rid of. The noise depends in large part by the sampling pattern. An alternative approach is therefore to consider our problem as a denoising problem:

Given measurements $\boldsymbol{y}$, we want our neural network Φ to take the "noisy" image $\boldsymbol{A}^*\boldsymbol{y}$ and return the (negative) noise from the image. This output can then be added to the noisy image to get the recovered image.

That is, the function we want to find is

$$f(y) = \boldsymbol{A}^*\boldsymbol{y} + \Phi(\boldsymbol{A}^*\boldsymbol{y}) \tag{5.3}$$

Where $\Phi\colon \mathbb{C}^N \to \mathbb{C}^N$ is a neural network that finds the noise in of its input. If this function is trained correctly, it would yield an image that is noiseless.

The desired output of the neural network when we input a noisy image, is an image that is as close to the fully sampled image as possible. When talking about training a neural network, we mean finding weight that give such output. To train, we need a cost function $\mathcal{C}\colon \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}$ that is a measure of whether two images are similar. A popular choice is $\mathcal{C}(\boldsymbol{x}, \boldsymbol{z}) = \|\boldsymbol{x} - \boldsymbol{z}\|_2^2$.

Remark 5.7. Several other cost functions exists. For instance, we could consider other norms. In principle, any differentiable function can be applied, as long as $\mathcal{C}(\boldsymbol{x}, \boldsymbol{x}) = 0$, to ensure that the we can identify the correct minimizer.

The job of the cost function is to tell us something about how "close" the two inputs are to each other. For imaging, this poses an additional challenge. Finding good cost functions is difficult.

By definition of the norm, the aforementioned cost function work well in the way that it is zero if and only if the output from the neural network is identical to the original image. However, achieving this is unfeasible that the network will become that good without severely overtraining the network.

It is imaginable that by using the aforementioned cost functions, we might get a large value for the cost, even though the images are quite similar visually. The reason for this is limitation in human vision. For instance, even if several pixels in the output deviate slightly from the original, we might not be able to detect it visually. Thus, assuming that details in the image is preserved, this could be a satisfying result, and we would expect a low value of the cost function. However, the small deviations will add up, and yield a net high cost. This would motivate more training even though we would get quite satisfying results. This loss might not be as large as when the differences between the images are more severe, but does not reflect the differences (or lack thereof) well.

For instance, consider a scaling of an image $\boldsymbol{x}$ by $\alpha \in \mathbb{R}$

$$\mathcal{C}(\boldsymbol{x}, \alpha\boldsymbol{x}) = \|\boldsymbol{x} - \alpha\boldsymbol{x}\|_2^2 = (1-\alpha)^2\|\boldsymbol{x}\|_2^2$$

This value can be quite large, depending on the dimensions of $\boldsymbol{x}$ and α. This is undesirable, as the scaling of α only changes the brightness of the image.

This is also demonstrated in Figure 5.2, where the cost between the original image Figure 5.2a and Figure 5.2b is larger than the cost between the original and Figure 5.2c. However, the Figure 5.2b is clearly the image that is visually closest to the original. The two loss values are approximately 70 and 47, respectively.

Assume that we are given a training set $\mathcal{T} = ((\boldsymbol{x}_1, \boldsymbol{y}_1), \dots, (\boldsymbol{x}_T, \boldsymbol{y}_T))$ where $\boldsymbol{x}_i$ is the true image, and $\boldsymbol{y}_i$ is the corresponding subsampled measurements for $i = 1, 2 \dots, T$. Training the neural network means solving the optimization problem

$$\boldsymbol{w} := (\boldsymbol{A}_\ell, \boldsymbol{b}_\ell, \dots, \boldsymbol{A}_1, \boldsymbol{b}_1) = \operatorname{argmin}_{(\boldsymbol{A}_\ell, \boldsymbol{b}_\ell, \dots, \boldsymbol{A}_1, \boldsymbol{b}_1)} \sum_{(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{T}} \mathcal{C}(\boldsymbol{x}, \Phi(\boldsymbol{y})) \tag{5.4}$$

Where $\boldsymbol{w}$, for ease of notation is the vector containing all the weight in the neural network. Minimizing the total $\mathcal{C}$ in terms of the weights of the network is therefore the goal. Doing this analytically is impossible. Instead, iterative methods are usually preferred. One of the most important algorithms are *gradient descent* and its variations [Goo+16].

Gradient descent

The idea behind gradient descent (GD), is that if we are given a starting point, we can iteratively improve it by iteratively finding points that improves the value of the objective function. The algorithm is greedy, in the sense that it will always go in the direction that improves the objection function the most. From calculus, we know that the gradient of a differentiable function points in the direction where the function is steepest upwards, and that the negative gradient points in the steepest direction downward. Thus, the chosen direction will always be that of the negative gradient. Because of this, gradient descent is also known as *steepest descent.*

Each step in gradient descent for solving the deep learning optimization problem is given by

$$\boldsymbol{w}^{n+1} = \boldsymbol{w}^n - \frac{\varepsilon}{T} \sum_{i=1}^{T} \nabla_{\boldsymbol{w}} \mathcal{C}(\boldsymbol{x}_i, \boldsymbol{y}_i; \boldsymbol{w}^n) \tag{5.5}$$

Where the step size $\varepsilon > 0$ is referred to as the *learning rate* in the context of deep learning. Setting an appropriate value can be difficult, if it is too small, the algorithm will converge slowly. On the other hand, if the step size is too large, we might have difficulties getting convergence, because the algorithm might overshoot, and miss the minimum.

To justify taking the average in the equation above, We stress that we are solving an optimization problem where the weights are the variables. Thus, we take the average of all the samples in the training set to get a value of the gradient that represents the class of images we are working with.

Stochastic gradient descent

When training a neural network, we want as many training samples as possible. Because one step of gradient descent involves computing the gradient for each sample in the training set, the time required to compute a single step with gradient descent will become quite large.

Stochastic gradient descent (SGD) utilizes the training data differently from GD. Instead of using the entire dataset in each iteration, the training set is randomly subsampled. Because of this, the time spent for each step can be reduced drastically.

At first glance, SGD may not seem like a significant improvement to GD. Using fewer samples improves the time spent for each iteration, but gives us poorer estimates, which gives us unsteady convergence. However, in the time frame GD uses to perform one step, we are able to perform enough steps with SGD to outperform GD.

In training step i, we create a set $\mathcal{T}_i \subset \mathcal{T}$ of randomly subsamples examples from the training set. The update step then becomes

$$\boldsymbol{w}^{n+1} = \boldsymbol{w}^n - \frac{\varepsilon}{|\mathcal{T}_i|} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{T}_i} \nabla_{\boldsymbol{w}} \mathcal{C}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}^n) \tag{5.6}$$

Other modifications can be made to improve gradient descent. For instance, it is normal to change the learning rate ε in each iteration. Another approach is to preserve the “momentum”. Usually, we only consider the gradient in the current point to determine which direction we will go next. When using momentum, however, we combine this with the step direction given in the previous step. This makes it more difficult for the algorithm to change the direction in each iteration, and provides a more steady path.

Challenges in training

Several issues exist when it comes to training neural networks with this approach.

Local minima The above algorithms work well on convex functions. However, the optimization problem (5.4) we are trying to solve when training neural networks is not a convex optimization problem. Therefore, we don't have any guarantees that applying the above algorithms will find a global minimum.

Ideally, gradient descent will stop when a stationary point is reached. For convex functions, we only have one such type of points, as the local and global minima are the same. If the function is. For neural networks, on the other hand, we might risk finding local minima, or saddle points. The functions are difficult to visualize, and it is therefore difficult to determine what type of point we have found.

Initial values The initial values of the weights are also important for the result. Ideally, we would like to pick initial values that are close to the global minimum, or at least places somewhere that would guarantee that we find the global minimum instead of some other stationary point. For other classes of functions, we might have some idea about what a good starting point is. However, due to the extreme complexity of neural networks, caused by the often vast number of weight, we do not have this luxury when working with neural networks. Therefore, it is normal to initialize the weights by random numbers, typically drawn from a Gaussian or uniform distribution [Goo+16].

Overfitting Deep neural networks are statistical models with many hyperparameters. Therefore, they are very flexible, and able to express a wide range of functions. The drawback to this is that they are prone to overfitting, meaning that they generalize poorly to unseen data. Generally, it helps to have large amounts of training samples. This is often easier said than done, as data acquisition may be expensive. Therefore, it is normal to limit the model in other ways. Typically, some regularization techniques are applied. One such technique is *dropout*, where we randomly "remove" nodes from the network, with a set probability.

Other problems with neural networks

Stability As briefly mentioned previously, neural networks often have severe stability issues [Ant+19; Hua+18; MFF16; Sze+13]. Most notably in the image classification setting, where imperceptible perturbations can drastically change the suggested label. This is true even for the images in the training set.

For inverse problems, this issue is manifested in that the network amplifies noise added to the image, resulting in potentially illegible output [Ant+19].

Lack of generality Neural networks are often trained on a specific class of images. This means that a net trained on MRI images may not work well on data from other medical imaging techniques, or Fourier measurements of e.g. everyday items. More surprisingly, neural networks also perform worse if they are given more samples [Ant+19].

CHAPTER 6

Solving the compressive sensing optimization problem

In this section, we are going to present two optimization algorithms for recovering a signal from the setup described in Chapter 4. Because this is a convex optimization problem, we are going to consider two algorithms for solving convex problems, FISTA and the Chambolle–Pock primal-dual algorithm. We will see that they work quite differently, and solves two different optimization problems. However, they also have some similarities. Most notably, they both utilize what is referred to as the *proximal operator*, defined by

Definition 6.1 (Proximal operator [FR13, p. 553])**.** The proximal operator P_F for a function F is the mapping

$$P_F(\boldsymbol{z}) = \operatorname{argmin}_{\boldsymbol{x}\in\mathbb{C}^N}\left\{F(\boldsymbol{x}) + \frac{1}{2}\|\boldsymbol{x}-\boldsymbol{z}\|_2^2\right\}$$

This is particularly useful in the cases where we want to minimize a convex function that is not differentiable. The proximal operator gives rise to an iterative optimization method. We see that the value of $P_F(\boldsymbol{z})$ is an $\boldsymbol{x}$ such that F has a low value, with $\boldsymbol{x}$ being close to $\boldsymbol{z}$. By iteratively applying the proximal operator, we are therefore able to improve the estimate of the minimizer of F without having to calculate the gradient.

Computing the value of the proximal operator does unfortunately involve solving an optimization problem. Luckily, for our cases, this can be found analytically.

6.1 FISTA

The Fast Iterative Shrinkage-Thresholding Algorithm (FISTA), introduced in [BT09] solves problems on the form

$$\min\left\{F(\boldsymbol{x}) = f(\boldsymbol{x}) + g(\boldsymbol{x})\colon \boldsymbol{x}\in\mathbb{R}^N\right\}$$

where

- $g\colon \mathbb{R}^N \to \mathbb{R}$ is a continuous convex function which is possibly *nonsmooth*.

- $f\colon \mathbb{R}^N \to \mathbb{R}$ is a smooth convex function which is continuously differentiable with Lipschitz continuous gradient $L(f)$, i.e.

$$\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{z})\|_2 \le L(f)\|\boldsymbol{x}-\boldsymbol{z}\|_2 \quad \forall \boldsymbol{x},\boldsymbol{z} \in \mathbb{R}^N$$

Because we need our objective function to be continuous, we will consider the optimization problem

$$\min_{\boldsymbol{z}\in\mathbb{C}^N} \lambda\|\boldsymbol{z}\|_1 + \|\boldsymbol{A}\boldsymbol{z}-\boldsymbol{y}\|_2^2$$

By Theorem 2.7 this is equivalent to the quadratically constrained basis pursuit problem Equation (2.5), with some caveats discussed in Remark 2.8.

Although we earlier defined $\boldsymbol{A} = \boldsymbol{P}_\Omega \boldsymbol{F}$, we will for ease of notation write $\boldsymbol{A} = \boldsymbol{P}_\Omega \boldsymbol{F}\boldsymbol{W}^{-1}$. Because we are working directly with the optimization problem, we are not concerned with differentiating between the sampling operator and the sparsity basis.

We set $f(\boldsymbol{z}) = \|\boldsymbol{A}\boldsymbol{z}-\boldsymbol{y}\|_2^2$ and $g(\boldsymbol{z}) = \lambda\|\boldsymbol{z}\|_1$. Both functions are continuous. The function f has continuous gradient $\nabla f(\boldsymbol{z}) = 2\boldsymbol{A}^*(\boldsymbol{A}\boldsymbol{z}-\boldsymbol{y})$.

For $\boldsymbol{z},\boldsymbol{x} \in \mathbb{C}^N$, we find the smallest Lipschitz constant for $\boldsymbol{A} = \boldsymbol{P}_\Omega \boldsymbol{F}\boldsymbol{W}^{-1}$ by the unitary invariance of the spectral norm:

$$\begin{aligned}\|\nabla f(\boldsymbol{z}) - \nabla f(\boldsymbol{x})\|_2 &= 2\|\boldsymbol{A}^*\boldsymbol{A}(\boldsymbol{z}-\boldsymbol{x})\|_2\\ &\le 2\|\boldsymbol{A}^*\boldsymbol{A}\|_2\|\boldsymbol{z}-\boldsymbol{x}\|_2\\ &= 2\|\boldsymbol{P}_\Omega^*\boldsymbol{P}_\Omega\|_2\|\boldsymbol{z}-\boldsymbol{x}\|_2\\ &= 2\|\boldsymbol{z}-\boldsymbol{x}\|_2\end{aligned}$$

Where we use that projection operators have spectrum $\{0,1\}$. Therefore, we will set $L = 2$.

FISTA relies on the quadratic approximation of F around $\boldsymbol{y}$ given by

$$Q(\boldsymbol{x},\boldsymbol{y}) = f(\boldsymbol{y}) + \langle \boldsymbol{x}-\boldsymbol{y}, \nabla f(\boldsymbol{y})\rangle + \frac{L}{2}\|\boldsymbol{x}-\boldsymbol{y}\|_2^2 + g(\boldsymbol{x})$$

This approximation has properties that we will discuss later. The proximal operator introduced in [BT09], is given by

$$p_L(\boldsymbol{y}) = \operatorname{argmin}_{\boldsymbol{z}}\left\{ g(\boldsymbol{z}) + \frac{L}{2}\left\|\boldsymbol{z} - \left(\boldsymbol{y} - \frac{1}{L}\nabla f(\boldsymbol{y})\right)\right\|_2^2\right\} \tag{6.1}$$

We see that it by definition computes the minimizer of $\boldsymbol{x} \mapsto Q(\boldsymbol{x},\boldsymbol{y})$.

In [BT09], this function is referred to as a proximal operator. However, it does not fit Definition 6.1. Instead, we see that it can be written as

$$p_L(\boldsymbol{y}) = P_g\left(\boldsymbol{y} - \frac{1}{L}\nabla f(\boldsymbol{y})\right)$$

which makes things a bit clearer. We recognize $\boldsymbol{y} - L^{-1}\nabla f(\boldsymbol{y})$ as one step of gradient descent, which will minimize f. By using this as the input to the proximal function, we will minimize both f and g simultaneously, which in turn means that we are minimizing the sum $F = f + g$.

It turns out that P_g is the soft thresholding operator, defined as follows

Definition 6.2 (Soft thresholding operator [FR13])**.** For $\alpha \geq 0$, the *soft thresholding operator* is the function $\mathcal{S}_\alpha \colon \mathbb{C}^N \to \mathbb{C}^N$. The entries of the output are given by

$$\mathcal{S}_\alpha(\boldsymbol{x})_i = \operatorname{sign}(\boldsymbol{x}_i) \max\{|\boldsymbol{x}_i| - \alpha, 0\} \tag{6.2}$$

This is shown in the next proposition

Proposition 6.3 (Proximal operator for FISTA)**.** *The proximal operator p_L for the setup in compressive sensing can be written as*

$$p_L(\boldsymbol{y}) = \mathcal{S}_{\lambda/L}(y - L^{-1}\nabla f(\boldsymbol{y}))$$

Proof. We first define the function

$$\begin{aligned} q_L(\boldsymbol{b}) &= \operatorname{argmin}_{\boldsymbol{x}} \left\{ \lambda \|\boldsymbol{x}\|_1 + \frac{L}{2}\|\boldsymbol{x} - \boldsymbol{b}\|_2^2 \right\} \\ &= \operatorname{argmin}_{\boldsymbol{x}} \left\{ \frac{\lambda}{L} \|\boldsymbol{x}\|_1 + \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{b}\|_2^2 \right\} \end{aligned} \tag{6.3}$$

and see that we can write $p_L(\boldsymbol{y}) = q_L(\boldsymbol{y} - L^{-1}\nabla f(\boldsymbol{y}))$. It remains to show that Equation (6.3) is the soft thresholding operator

By the definition of the norms, we see that we can write the minimization problem as

$$\sum_{i=1}^{N} \left(\frac{\lambda}{L}|x_i| + \frac{1}{2}(x_i - b_i)^2 \right)$$

We will minimize each term by itself. We set $\alpha = \lambda/L$, and define the function

$$h(x_i) = \begin{cases} \alpha x_i + \frac{1}{2}(x_i - b_i)^2 & \text{if } x_i \geq 0 \\ -\alpha x_i + \frac{1}{2}(x_i - b_i)^2 & \text{if } x_i \leq 0 \end{cases}$$

Taking the derivative

$$h'(x_i) = \begin{cases} \alpha + x_i - b_i & \text{if } \boldsymbol{x}_i \geq 0 \\ -\alpha + x_i - b_i & \text{if } \boldsymbol{x}_i \leq 0 \end{cases}$$

Setting this equal to zero, for $x_i \geq 0$, the minimizer is $x_i = b_i - \alpha$ or 0. For $x_i \geq 0$ we attain the minimum in $b_i + \alpha$ or 0.

The minimized points become

$$\begin{aligned} h(b_i - \alpha) &= \alpha\left(b_i - \frac{1}{2}\alpha\right) && b_i > \alpha \\ h(0) &= \frac{1}{2}b_i^2 && |b_i| \leq \alpha \\ h(b_i + \alpha) &= \alpha\left(b_i + \frac{1}{2}\alpha\right) && -b_i > \alpha \end{aligned}$$

To summarize, the minimizer is $\operatorname{sign}(b_i)\max\{|b_i| - \alpha, 0\}$, and $p_L(\boldsymbol{y}) = \mathcal{S}_{\lambda/L}\Big(\boldsymbol{y} - L^{-1}\nabla f(\boldsymbol{y})\Big)$ ■

We now have everything we need to describe the algorithm.

Next, we will show the that FISTA in fact converges to the minimizer.

Algorithm 6.1 FISTA

```
Input:  $L = L(f)$ – A Lipschitz constant of $\nabla f$
Initialization:  Take $\boldsymbol{y}_1 = \boldsymbol{x}_0 = \boldsymbol{A}^*\boldsymbol{y}$, $t_1 = 1$
Step $k \geq 1$:  Compute
    $\boldsymbol{x}_k = p_L(\boldsymbol{y}_k)$    (6.4)
    $t_{k+1} = \frac{1+\sqrt{1+4t_k^2}}{2}$    (6.5)
    $\boldsymbol{y}_{k+1} = \boldsymbol{x}_k + \left(\frac{t_k - 1}{t_{k+1}}\right)(\boldsymbol{x}_k - \boldsymbol{x}_{k-1})$    (6.6)
```

Correctness of the algorithm

The main selling point of FISTA is that while its predecessor ISTA converges with a rate of $\mathcal{O}(1/k)$, FISTA converges with a rate of $\mathcal{O}(1/k^2)$. This is a consequence of the main result in this section.

Before showing the results, we note that while the Lipschitz constant is easily computable for our setup, this is not the case in general. Therefore, an alternative to the variant of FISTA given in Algorithm 6.1 is also introduced in [BT09]. The paper proves the correctness for both variants simultaneously, but for ease of notation, we have omitted details concerning the alternative variant.

Central to the proof, is some tools from convex analysis

Definition 6.4 (Subgradient [FR13, p. 551])**.** The subdifferential of a convex function $F\colon \mathbb{C}^N \to (0, \infty]$ at a point $\boldsymbol{x} \in \mathbb{C}^N$ is defined by

$$\partial F(\boldsymbol{x}) = \left\{ \boldsymbol{v} \in \mathbb{C}^N \mid F(\boldsymbol{z}) \geq F(\boldsymbol{x}) + \langle \boldsymbol{v}, \boldsymbol{z} - \boldsymbol{x} \rangle \; \forall \boldsymbol{z} \in \mathbb{C}^N \right\}$$

The subdifferential in a point can be interpreted as the candidates to the gradient. The next result is therefore not surprising

Lemma 6.5 ([FR13, p. 552])**.** *The vector $\boldsymbol{x}$ is a minimizer of F if and only if $\boldsymbol{0} \in \partial F(\boldsymbol{x})$*

Proof. The proof is shown by applying the definition of the subdifferential.

$$\begin{aligned}
& \boldsymbol{0} \in \partial F(\boldsymbol{x}) \\
\iff & F(\boldsymbol{z}) \geq F(\boldsymbol{x}) + \langle \boldsymbol{0}, \boldsymbol{z} - \boldsymbol{x} \rangle && \forall \boldsymbol{z} \\
\iff & F(\boldsymbol{z}) \geq F(\boldsymbol{x}) && \forall \boldsymbol{z} \\
\iff & \boldsymbol{x} \text{ is a minimizer of } F
\end{aligned}$$

■

The subdifferential plays an important role in the following lemma

Lemma 6.6 ([BT09])**.** *For any $\boldsymbol{y} \in \mathbb{C}^N$, one has $\boldsymbol{z} = p_L(\boldsymbol{y})$ if and only if there exists a $\gamma(\boldsymbol{y}) \in \partial g(\boldsymbol{z})$, such that*

$$\nabla f(\boldsymbol{y}) + L(\boldsymbol{z} - \boldsymbol{y}) + \gamma(\boldsymbol{y}) = \boldsymbol{0}$$

Proof. We see that the result holds by considering the objective function

$$Q(\boldsymbol{z}, \boldsymbol{y}) = f(\boldsymbol{y}) + \langle \boldsymbol{z} - \boldsymbol{y}, \nabla f(\boldsymbol{y}) \rangle + \frac{L}{2} \|\boldsymbol{z} - \boldsymbol{y}\|_2^2 + g(\boldsymbol{z})$$

By the Moreau–Rockafellar theorem, we can compute the subgradient of Q by adding the subdifferentials of its terms. In addition, we have that $\partial f(x) = \{\nabla f(x)\}$ for differentiable functions. We get

$$\partial Q(\boldsymbol{z}, \boldsymbol{y}) = \{\nabla f(\boldsymbol{y})\} + \{L(\boldsymbol{z} - \boldsymbol{y})\} + \partial g(\boldsymbol{z})$$

Because $\boldsymbol{z}$ is the minimizer if and only if $0 \in \partial Q(\boldsymbol{z}, \boldsymbol{y})$, and $\partial g(\boldsymbol{z})$ is the only set in the above expression that can contain more than one element, we have that $\boldsymbol{0}$ is in the subdifferential if and only if there is a $\gamma(\boldsymbol{y}) \in \partial g(\boldsymbol{z})$ such that

$$\nabla f(\boldsymbol{y}) + L(\boldsymbol{z} - \boldsymbol{y}) + \gamma(\boldsymbol{y}) = \boldsymbol{0}$$

■

A consequence of the next lemma, is that the quadratic approximation $Q(x, y)$ is always greater than or equal to F, so the premise of Lemma 6.8 is always satisfied for our setup.

Lemma 6.7 ([BT09])**.** *Let $f \colon \mathbb{C}^N \to \mathbb{R}$ be a continuously differentiable function with Lipschitz continuous gradient and Lipschitz constant $L(f)$. Then, for any $L \geq L(f)$,*

$$f(\boldsymbol{x}) \leq f(\boldsymbol{y}) + \langle \boldsymbol{x} - \boldsymbol{y}, \nabla f(\boldsymbol{y}) \rangle + \frac{L}{2} \|\boldsymbol{x} - \boldsymbol{y}\|_2^2$$

for every $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{C}^N$

Lemma 6.8 plays an important role in the main result

Lemma 6.8 ([BT09])**.** *Let $\boldsymbol{y} \in \mathbb{C}^N$ and $L > 0$ be such that*

$$F(p_L(\boldsymbol{y})) \leq Q(p_L(\boldsymbol{y}), \boldsymbol{y})$$

Then for any $\boldsymbol{x} \in \mathbb{C}^N$,

$$F(\boldsymbol{x}) - F(p_L(\boldsymbol{y})) \geq \frac{L}{2} \|p_L(\boldsymbol{y}) - \boldsymbol{y}\|_2^2 + L\langle \boldsymbol{y} - \boldsymbol{x}, p_L(\boldsymbol{y}) - \boldsymbol{y} \rangle$$

Proof. From the assumptions in the lemma, we have

$$F(\boldsymbol{x}) - F(p_L(\boldsymbol{y})) \geq F(\boldsymbol{x}) - Q(p_L(\boldsymbol{y}), \boldsymbol{y}) \tag{6.7}$$

By the convexity of f and g, we have

$$\begin{aligned} f(\boldsymbol{x}) &\geq f(\boldsymbol{y}) + \langle \boldsymbol{x} - \boldsymbol{y}, \nabla f(\boldsymbol{y}) \rangle \\ g(\boldsymbol{x}) &\geq g(p_L(\boldsymbol{y})) + \langle \boldsymbol{x} - p_L(\boldsymbol{y}), \gamma(\boldsymbol{y}) \rangle \end{aligned}$$

Here, $\gamma(\boldsymbol{y}) \in \partial g(p_L(\boldsymbol{y}))$ is given as in Lemma 6.6. Next, we sum the inequalities above to obtain

$$F(\boldsymbol{x}) \geq f(\boldsymbol{y}) + \langle \boldsymbol{x} - \boldsymbol{y}, \nabla f(\boldsymbol{y}) \rangle + g(p_L(\boldsymbol{y})) + \langle \boldsymbol{x} - p_L(\boldsymbol{y}), \gamma(\boldsymbol{y}) \rangle$$

We also have

$$Q(p_L(\boldsymbol{y}),\boldsymbol{y}) = f(\boldsymbol{y}) + \langle p_L(\boldsymbol{y}) - \boldsymbol{y}, \nabla f(\boldsymbol{y})\rangle + \frac{L}{2}\|p_L(\boldsymbol{y}) - \boldsymbol{y}\|_2^2 + g(p_L(\boldsymbol{y}))$$

Applying the two above functions, along with Lemma 6.6, we get

$$\begin{aligned}
F(\boldsymbol{x}) - F(p_L(\boldsymbol{y})) &\geq F(\boldsymbol{x}) - Q(p(\boldsymbol{y}),\boldsymbol{y}) \\
&\geq -\frac{L}{2}\|p_L(\boldsymbol{y}) - \boldsymbol{y}\|_2^2 + \langle \boldsymbol{x} - p_L(\boldsymbol{y}), \nabla f(\boldsymbol{y}) + \gamma(\boldsymbol{y})\rangle \\
&= -\frac{L}{2}\|p_L(\boldsymbol{y}) - \boldsymbol{y}\|_2^2 + \langle \boldsymbol{x} - p_L(\boldsymbol{y}), \nabla f(\boldsymbol{y}) \\
&\qquad\qquad + (-\nabla f(\boldsymbol{y}) - L(p_L(\boldsymbol{y}) - \boldsymbol{y}))\rangle \\
&= -\frac{L}{2}\|p_L(\boldsymbol{y}) - \boldsymbol{y}\|_2^2 + L\langle \boldsymbol{x} - p_L(\boldsymbol{y}), \boldsymbol{y} - p_L(\boldsymbol{y})\rangle \\
&= -\frac{L}{2}\|p_L(\boldsymbol{y}) - \boldsymbol{y}\|_2^2 + L\langle \boldsymbol{x} + \boldsymbol{y} - \boldsymbol{y} - p_L(\boldsymbol{y}), \boldsymbol{y} - p_L(\boldsymbol{y})\rangle \\
&= \frac{L}{2}\|p_L(\boldsymbol{y}) - \boldsymbol{y}\|_2^2 + L\langle \boldsymbol{y} - \boldsymbol{x}, p_L(\boldsymbol{y}) - \boldsymbol{y}\rangle
\end{aligned}$$

And we have arrived at our conclusion. ■

Lemma 6.9 ([BT09]). *The sequences $\{\boldsymbol{x}_k, \boldsymbol{y}_k\}$ generated by FISTA with constant step size satisfies for every $k \geq 1$*

$$\frac{2}{L}t_k^2 v_k - \frac{2}{L}t_{k+1}^2 v_{k+1} \geq \|\boldsymbol{u}_{k+1}\|_2^2 - \|\boldsymbol{u}_k\|_2^2$$

where $v_k = F(\boldsymbol{x}_k) - F(\boldsymbol{x}^*)$ and $\boldsymbol{u}_k = t_k\boldsymbol{x}_k - (t_k - 1)\boldsymbol{x}_{k-1} - \boldsymbol{x}^*$, with $\boldsymbol{x}^*$ being a minimizer.

Proof. We first apply Lemma 6.8 twice, both for $\boldsymbol{x} = \boldsymbol{x}_k, \boldsymbol{y} = \boldsymbol{y}_{k+1}$ and with $\boldsymbol{x} = \boldsymbol{x}^*, \boldsymbol{y} = \boldsymbol{y}_{k+1}$. By rearranging and the definition of $\{v_k\}$ and $\boldsymbol{x}_{k+1} = p_L(\boldsymbol{y}_{k+1})$, we obtain

$$\begin{aligned}
2L^{-1}(v_k - v_{k+1}) &\geq \|\boldsymbol{x}_{k+1} - \boldsymbol{y}_{k+1}\|_2^2 + 2\langle \boldsymbol{x}_{k+1} - \boldsymbol{y}_{k+1}, \boldsymbol{y}_{k+1} - \boldsymbol{x}_k\rangle \\
-2L^{-1}v_{k+1} &\geq \|\boldsymbol{x}_{k+1} - \boldsymbol{y}_{k+1}\|_2^2 + 2\langle \boldsymbol{x}_{k+1} - \boldsymbol{y}_{k+1}, \boldsymbol{y}_{k+1} - \boldsymbol{x}^*\rangle
\end{aligned}$$

We multiply the first inequality by $(t_{k+1} - 1)$ and add it to the second inequality to get

$$\begin{aligned}
\frac{2}{L}((t_{k+1} - 1)v_k - t_{k+1}v_{k+1}) \geq & t_{k+1}\|\boldsymbol{x}_{k+1} - \boldsymbol{y}_{k+1}\|_2^2 \\
& + 2\langle \boldsymbol{x}_{k+1} - \boldsymbol{y}_{k+1}, t_{k+1}\boldsymbol{y}_{k+1} \\
& - (t_{k+1} - 1)\boldsymbol{x}_k - \boldsymbol{x}^*\rangle
\end{aligned}$$

From the definition of t_k, we have $t_k^2 = t_{k+1} - t_{k+1}^2$, which gives

$$\begin{aligned}
\frac{2}{L}(t_k^2 v_k - t_{k+1}^2 v_{k+1}) \geq & \|t_{k+1}(\boldsymbol{x}_{k+1} - \boldsymbol{y}_{k+1})\|_2^2 \\
& + 2t_{k+1}\langle \boldsymbol{x}_{k+1} - \boldsymbol{y}_{k+1}, t_{k+1}\boldsymbol{y}_{k+1} - (t_{k+1} - 1)\boldsymbol{x}_k - \boldsymbol{x}^*\rangle
\end{aligned}$$

The relation

$$\|\boldsymbol{b}-\boldsymbol{a}\|_2^2 + 2\langle \boldsymbol{b}-\boldsymbol{a}, \boldsymbol{a}-\boldsymbol{c}\rangle = \|\boldsymbol{b}-\boldsymbol{c}\|_2^2 - \|\boldsymbol{a}-\boldsymbol{c}\|_2^2$$

with

$$\boldsymbol{a} = t_{k+1}\boldsymbol{y}_{k+1}, \quad \boldsymbol{b} = t_{k+1}\boldsymbol{x}_{k+1}, \quad \boldsymbol{c} = (t_{k+1}-1)\boldsymbol{x}_k + \boldsymbol{x}^*$$

Applied to the right hand side of the above inequality, we obtain

$$\begin{aligned}\frac{2}{L}(t_k^2 v_k - t_{k+1}^2 v_{k+1}) \geq \|t_{k+1}\boldsymbol{x}_{k+1} - (t_{k+1}-1)\boldsymbol{x}_k - \boldsymbol{x}^*\|_2^2 \\ - \|t_{k+1}\boldsymbol{y}_{k+1} - (t_{k+1}-1)\boldsymbol{x}_k - \boldsymbol{x}^*\|_2^2\end{aligned}$$

Substitute the definition of $\boldsymbol{y}_{k+1}$ and $\boldsymbol{u}_k$ completes the proof. ■

The next simple results are also important for proving the main result

Lemma 6.10. *Let $\{a_k, b_k\}$ be sequences of positive reals satisfying*

$$a_k - a_{k+1} \geq b_{k+1} - b_k \quad \forall k \geq 1, \textit{with } a_1 + b_1 \leq c, c > 0$$

Then $a_k \leq c$ for every $k \geq 1$.

Proof. We rewrite

$$a_k - a_{k+1} \geq b_{k+1} - b_k$$

to obtain

$$a_k + b_k \geq b_{k+1} + a_{k+1}$$

Which means that the sequence $\{a_k + b_k\}$ is a decreasing sequence. Because $b_k \geq 0$ for all k, we therefore have

$$c \geq a_1 + b_1 \geq a_k + b_k \geq a_k$$

which completes the proof. ■

Lemma 6.11. *The sequence $\{t_k\}$ of positive numbers generated by FISTA with $t_1 = 1$ satisfies $t_k \geq (k+1)/2$ for all $k \geq 1$.*

Proof. We prove this by induction. The statement holds in the base case $k = 1$ as $t_1 = 1 = \frac{1+1}{2} = 1$. Assuming that the statement holds for t_k, it remains to show that $t_{k+1} = \frac{k+2}{2}$. Indeed, because the square root is a monotone function, we get

$$t_{k+1} = \frac{1+\sqrt{1+4t_k^2}}{2} \geq \frac{1+\sqrt{4t_k^2}}{2} = \frac{1+(k+1)}{2}$$

Where the induction step used in the equality completes the proof. ■

We can now establish the main result, whose proof builds on the conclusion of Lemma 6.9.

Theorem 6.12 ([BT09])**.** *Let $\{\boldsymbol{x}_k\}, \{\boldsymbol{y}_k\}$ be generated by FISTA. Then for any $k \geq 1$*

$$F(\boldsymbol{x}_k) - F(\boldsymbol{x}^*) \leq \frac{2L\|\boldsymbol{x}_0 - \boldsymbol{x}^*\|_2^2}{(k+1)^2}$$

For all minimizers $\boldsymbol{x}^*$.

Proof. We define

$$a_k := \frac{2}{L} t_k^2 v_k, \quad b_k := \|\boldsymbol{u}_k\|_2^2, \quad c := \|\boldsymbol{y}_1 - \boldsymbol{x}^*\|_2^2$$

Where we recall that we previously defined

$$\begin{aligned} v_k &:= F(\boldsymbol{x}_k) - F(\boldsymbol{x}^*) \\ \boldsymbol{u}_k &:= t_k \boldsymbol{x}_k - (t_k - 1)\boldsymbol{x}_{k-1} - \boldsymbol{x}^* \end{aligned}$$

By Lemma 6.9, we have that for every $k \geq 1$.

$$a_k - a_{k+1} \geq b_{k+1} - b_k$$

Applying Lemma 6.10, would complete the proof. To apply it, we need to show that show that $a_1 + b_1 \leq c$. We see that

$$\begin{aligned} F(\boldsymbol{x}^*) - F(\boldsymbol{x}_1) &= F(\boldsymbol{x}^*) - F(p_L(\boldsymbol{y}_1)) \\ &\geq \frac{L}{2}\|p_L(\boldsymbol{y}_1) - \boldsymbol{y}_1\|_2^2 + L\langle \boldsymbol{y}_1 - \boldsymbol{x}^*, p_L(\boldsymbol{y}_1) - \boldsymbol{y}_1\rangle \\ &= \frac{L}{2}\|\boldsymbol{x}_1 - \boldsymbol{y}_1\|_2^2 + L\langle \boldsymbol{y}_1 - \boldsymbol{x}^*, \boldsymbol{x}_1 - \boldsymbol{y}_1\rangle \\ &= \frac{L}{2}\left(\|\boldsymbol{x}_1 - \boldsymbol{x}^*\|_2^2 - \|\boldsymbol{y}_1 - \boldsymbol{x}^*\|_2^2\right) \end{aligned}$$

Where we have applied Lemma 6.8 with $\boldsymbol{x} = \boldsymbol{x}^*$ and $\boldsymbol{y} = \boldsymbol{y}_1$ to obtain the inequality in the expression above. The requirements in Lemma 6.8 are satisfied by Lemma 6.7. Thus, we have that since $t_1 = 1$

$$a_1 = \frac{2}{L} v_1, \quad b_1 = \|\boldsymbol{x}_2 - \boldsymbol{x}^*\|_2$$

And so

$$\frac{2}{L} v_1 \leq \|\boldsymbol{y}_1 - \boldsymbol{x}^*\|_2^2 - \|\boldsymbol{x}_1 - \boldsymbol{x}^*\|_2^2$$

And we have established that $a_1 + b_1 \leq c$. Thus, by Lemma 6.10

$$\frac{2}{L} t_k^2 v_k \leq \|\boldsymbol{x}_0 - \boldsymbol{x}^*\|_2^2$$

The proof is completed by Lemma 6.9, as

$$v_k \leq \frac{L\|\boldsymbol{x}_0 - \boldsymbol{x}^*\|_2^2}{2t_k^2} \leq \frac{2L\|\boldsymbol{x}_0 - \boldsymbol{x}^*\|_2^2}{(k+1)^2}.$$

■

6.2 Chambolle and Pock's primal-dual algorithm

We begin this section by introducing the notion of lower semicontinuous functions and extended real-valued functions. The latter are functions that are allowed to take $\pm\infty$, while lower semicontinuity is given by Definition 6.13

Definition 6.13 (Lower semicontinuity [FR13, p. 547])**.** A function $F\colon \mathbb{C}^N \to (-\infty, \infty]$ is called lower semicontinuous if, for every $\boldsymbol{x} \in \mathbb{C}^N$, and every sequence $\{\boldsymbol{x}_j\}_{j\geq 1}$ converging to $\boldsymbol{x}$,

$$\liminf_{j\to\infty} F(\boldsymbol{x}_j) \geq F(\boldsymbol{x})$$

Chambolle and Pock's primal-dual algorithm, solves problems on the form

$$\min_{\boldsymbol{x}\in\mathbb{C}^N} F(\boldsymbol{A}\boldsymbol{x}) + G(\boldsymbol{x}) \tag{6.8}$$

Where $\boldsymbol{A} \in \mathbb{C}^{m\times N}$ and where $F\colon \mathbb{C}^m \to (-\infty, \infty]$ and $G\colon \mathbb{C}^N \to (-\infty, \infty]$ are extended real-valued lower semicontinuous convex functions.

We set

$$\begin{aligned} G(\boldsymbol{x}) &= \|\boldsymbol{x}\|_1 \\ F(\boldsymbol{x}) &= \chi_{B(\boldsymbol{y},\eta)}(\boldsymbol{x}) = \begin{cases} 0 & \text{if } \|\boldsymbol{x}-\boldsymbol{y}\|_2 \leq \eta \\ \infty & \text{otherwise} \end{cases} \end{aligned}$$

We see that G is continuous, and therefore lower semicontinuous. Also, $F(\boldsymbol{x})$ is lower semicontinuous because the ball $B(\boldsymbol{y}, \eta)$ is closed [FR13, p. 485].

This algorithm also utilizes proximal operators. For a function G and $\tau \in \mathbb{R}$, we introduce the notation $P_{\tau G}(\boldsymbol{x}) = P_G(\tau; \boldsymbol{x})$

In addition, the Chambolle–Pock algorithm needs the proximal operator for the convex conjugate function of F, defined to be

$$F^*(\boldsymbol{y}) = \sup_{\boldsymbol{x}\in\mathbb{C}^N} \{\langle \boldsymbol{x}, \boldsymbol{y}\rangle - F(\boldsymbol{x})\}$$

The proximal operator P_{F^*} is given by the following lemma.

Lemma 6.14 ([FR13, p. 485])**.** *The proximal operator* $P_{F^*}(\sigma, \boldsymbol{\xi})$ *is given by*

$$P_{F^*}(\sigma;\boldsymbol{\xi}) = \begin{cases} \boldsymbol{0} & \text{if } \|\boldsymbol{\xi}-\sigma\boldsymbol{y}\|_2 \leq \eta\sigma \\ \left(1 - \frac{\eta\sigma}{\|\boldsymbol{\xi}-\sigma\boldsymbol{y}\|_2}\right)(\boldsymbol{\xi}-\sigma\boldsymbol{y}) & \text{otherwise} \end{cases} \tag{6.9}$$

Proof. We first observe that

$$P_F(\sigma, \boldsymbol{\xi}) = \operatorname{argmin}_{\boldsymbol{\zeta}\in\mathbb{C}^m : \|\boldsymbol{\zeta}-\boldsymbol{\xi}\|_2\leq\eta} \|\boldsymbol{\zeta}-\boldsymbol{\xi}\|_2$$

We see that this is the orthogonal projection onto the ball $B(\boldsymbol{y}, \eta)$. We write this as

$$P_F(\sigma, \boldsymbol{\xi}) = \begin{cases} \boldsymbol{\xi} & \text{if } \|\boldsymbol{\xi}-\boldsymbol{y}\|_2 \leq \eta \\ \boldsymbol{y} + \frac{\eta}{\|\boldsymbol{\xi}-\boldsymbol{y}\|_2}(\boldsymbol{\xi}-\boldsymbol{y}) & \text{otherwise} \end{cases}$$

To find P_{F^*}, we apply Moreau's identity [FR13]

$$P_{\tau F^*}(\boldsymbol{z}) + \tau P_{\tau^{-1}F}(\boldsymbol{z}/\tau) = \boldsymbol{z}$$

and get

$$P_{F^*}(\sigma;\boldsymbol{\xi}) = P_{\sigma F^*}(\boldsymbol{\xi}) = \begin{cases} \mathbf{0} & \text{if } \|\boldsymbol{\xi}-\sigma\boldsymbol{y}\|_2 \le \eta\sigma \\ \left(1 - \frac{\eta\sigma}{\|\boldsymbol{\xi}-\sigma\boldsymbol{y}\|_2}\right)(\boldsymbol{\xi}-\sigma\boldsymbol{y}) & \text{otherwise} \end{cases}$$

■

In addition, the proximal operator

$$P_G(\tau;\boldsymbol{z}) = \mathcal{S}_\tau(\boldsymbol{z}) \tag{6.10}$$

is derived similarly to the proximal operator in Section 6.1, by omitting the part concerning $\boldsymbol{y} - L^{-1}\nabla f(\boldsymbol{y})$.

We now have what we need to describe the algorithm.

Algorithm 6.2 Chambolle and Pock's primal-dual algorithm

```
Input: $\boldsymbol{A} \in \mathbb{C}^{m\times N}$, convex functions $F, G$
Parameters: $\theta \in [0,1]$, $\tau,\sigma > 0$ such that $\tau\sigma\|\boldsymbol{A}\|_2^2 < 1$
Initialization: $\boldsymbol{x}_0 \in \mathbb{C}^N$, $\boldsymbol{\xi}_0 \in \mathbb{C}^m$, $\overline{\boldsymbol{x}}_0 = \boldsymbol{x}_0$
Step $k \ge 1$: Compute
    $\boldsymbol{\xi}_{k+1} = P_{F^*}(\sigma;\boldsymbol{\xi}_k + \sigma\boldsymbol{A}\overline{\boldsymbol{x}}_k)$        (PD$_1$)
    $\boldsymbol{x}_{k+1} = P_G(\tau;\boldsymbol{x}_k - \tau\boldsymbol{A}^*\boldsymbol{\xi}_{k+1})$        (PD$_2$)
    $\overline{\boldsymbol{x}}_{k+1} = \boldsymbol{x}_{k+1} + \theta(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k)$        (PD$_3$)
```

Duality theory

Before diving into proving that this algorithm works, we recall the basics of duality theory. We consider our problem in a more general sense

$$\min_{\boldsymbol{x}\in\mathbb{C}^N} F_0(\boldsymbol{x}) \quad \text{subject to } \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y} \tag{6.11}$$

We refer to this as the primal problem. Generally, we also consider inequality constraints, but because we don't need these for our setup, they are omitted.

An important component in duality theory is the Lagrange function, which gives us a similar optimization problem to Equation (6.11)

The key to duality theory is to consider this function as an optimization problem in terms of the Lagrange multiplier $\boldsymbol{\xi}$. We define

$$H(\boldsymbol{\xi}) = \inf_{\boldsymbol{x}\in\mathbb{C}^n} L(\boldsymbol{x},\boldsymbol{\xi}) = \inf_{\boldsymbol{x}\in\mathbb{C}^n} F_0(\boldsymbol{x}) - \langle\boldsymbol{x},\boldsymbol{\xi}\rangle$$

For the optimal value $F_0(\boldsymbol{x}^\sharp)$ of Equation (6.11), we have

$$H(\boldsymbol{\xi}) \le F_0(\boldsymbol{x}^\sharp)$$

The function H provides a lower bound of the primal problem (6.11). This motivates looking at the problem

$$\max_{\boldsymbol{\xi}\in\mathbb{C}^m} H(\boldsymbol{\xi}) \tag{6.12}$$

to get an optimal lower bound of the minimum of the primal problem. We call this optimization problem the dual problem. If $\boldsymbol{x}^\sharp$ is primal optimal, and $\boldsymbol{\xi}^\sharp$ is dual optimal, i.e. optimal for their respective problems, we call the tuple $(\boldsymbol{x}^\sharp, \boldsymbol{\xi}^\sharp)$ primal-dual optimal, and we have weak duality, i.e.

$$H(\boldsymbol{\xi}^\sharp) \leq F(\boldsymbol{x}^\sharp)$$

If we have equality in the above equation, we say that we have strong duality.

We will show that with our setup, we have strong duality. The dual problem is

$$\max_{\boldsymbol{\xi}\in\mathbb{C}^m} -F^*(\boldsymbol{\xi}) - G^*(-\boldsymbol{A}^*\boldsymbol{\xi}) \tag{6.13}$$

To see that this is indeed the case, we set $\boldsymbol{z} = \boldsymbol{Ax}$, and rewrite Equation (6.8) as

$$\min_{\boldsymbol{x},\boldsymbol{z}} F(\boldsymbol{z}) + G(\boldsymbol{x}) \quad \text{subject to } \boldsymbol{Ax} - \boldsymbol{z} = \boldsymbol{0}$$

Because we have the equality constraint, the Lagrangian function is therefore

$$L(\boldsymbol{x},\boldsymbol{z}) = F(\boldsymbol{z}) + G(\boldsymbol{x}) + \langle \boldsymbol{\xi}, \boldsymbol{Ax} - \boldsymbol{z}\rangle$$

The dual Lagrangian becomes, by definition of the convex conjugation

$$\begin{aligned} H(\boldsymbol{\xi}) &= \inf_{\boldsymbol{x},\boldsymbol{z}}\{L(\boldsymbol{x},\boldsymbol{z},\boldsymbol{\xi})\} \\ &= -\sup_{\boldsymbol{z}}\{\langle \boldsymbol{\xi},\boldsymbol{z}\rangle - F(\boldsymbol{z})\} - \sup_{\boldsymbol{x}}\{\langle \boldsymbol{x}, -\boldsymbol{A}^*\boldsymbol{\xi}\rangle - G(\boldsymbol{x})\} \\ &= -F^*(\boldsymbol{\xi}) - G^*(-\boldsymbol{A}^*\boldsymbol{\xi}) \end{aligned}$$

Minimizing the function $H(\boldsymbol{\xi})$ thus gives us the proposed dual optimization problem (6.13).

Because of strong duality, solving the primal and dual optimization problems simultaneously is the same as finding the solution of

$$\min_{\boldsymbol{x}\in\mathbb{C}^N}\max_{\boldsymbol{\xi}\in\mathbb{C}^m} \operatorname{Re}\langle \boldsymbol{Ax},\boldsymbol{\xi}\rangle + G(\boldsymbol{x}) - F^*(\boldsymbol{\xi}) \tag{6.14}$$

which is known as the saddle-point problem. We know that this property holds from Proposition 6.15

Proposition 6.15 ([FR13, p. 564]). *Let $\boldsymbol{A} \in \mathbb{C}^{m\times N}$ and $F\colon \mathbb{C}^m \to (-\infty,\infty]$ $G\colon \mathbb{C}^N \to (-\infty,\infty]$ be proper convex functions such that there exist $\boldsymbol{x}\in\mathbb{C}^N$ such that $\boldsymbol{Ax}$ is in the domain of F. Assume that both the primal and the dual problem have optimal solutions. Then we have strong duality, i.e.*

$$\min_{\boldsymbol{x}\in\mathbb{C}^N}(F(\boldsymbol{Ax}) + G(\boldsymbol{x})) = \max_{\boldsymbol{\xi}\in\mathbb{C}^m}\left(-F^*(\boldsymbol{\xi}) - G^*(-\boldsymbol{A}^*\boldsymbol{\xi})\right)$$

Furthermore, a primal-dual solution $(\boldsymbol{x}^\sharp,\boldsymbol{\xi}^\sharp)$ is a solution to the saddle-point problem

$$\min_{\boldsymbol{x}\in\mathbb{C}^N}\max_{\boldsymbol{\xi}\in\mathbb{C}^m}\langle \boldsymbol{Ax},\boldsymbol{\xi}\rangle + G(\boldsymbol{x}) - F^*(\boldsymbol{\xi})$$

Correctness of the algorithm

The main result with regards to showing the correctness of the Chambolle–Pock Primal-Dual algorithm is given below

Theorem 6.16 ([FR13])**.** *Assume that Equation* (6.14) *has a saddle point. Pick* $\theta = 1$ *and* $\sigma, \tau > 0$ *such that* $\sigma\tau\|\boldsymbol{A}\|_2^2 < 1$. *Let* $\{\boldsymbol{x}_n, \overline{\boldsymbol{x}}_n, \boldsymbol{\xi}_n\}_{n=0}^{\infty}$ *be the sequence generated by the primal-dual algorithm. Then the sequence* $(\boldsymbol{x}_n, \boldsymbol{\xi}_n)$ *converges to a saddle point* $(\boldsymbol{x}_n^\sharp, \boldsymbol{\xi}_n^\sharp)$ *of Equation* (6.14)*. In particular, the sequence of* $\boldsymbol{x}_n$ *converges to a minimizer of Equation* (6.8)*.*

The proof of this theorem is rather convoluted. We will structure it by showing a series of lemmas. The proofs of these are also omitted, as they involve more heavy computations than difficult theory.

The first involves the discrete derivate

$$\Delta_\tau \boldsymbol{u}_n = \frac{\boldsymbol{u}_n - \boldsymbol{u}_{n1}}{\tau}, \quad n \in \mathbb{N}$$

We further extend this to other sequences

$$\Delta_\tau \|\boldsymbol{u}_n\|_2^2 = \frac{\|\boldsymbol{u}_n\|_2^2 - \|\boldsymbol{u}_{n1}\|_2^2}{\tau}, \quad n \in \mathbb{N}$$

and multiple orders, e.g. $\Delta_\tau^2 \boldsymbol{u}_n \coloneqq \Delta_\tau \Delta_\tau \boldsymbol{u}_n$

The following lemma proves some desirable properties

Lemma 6.17. *Let* $\boldsymbol{u}, \boldsymbol{u}_n \in \mathbb{C}^N, n \geq 0$. *Then*

$$2\operatorname{Re}\langle \Delta_\tau \boldsymbol{u}_n, \boldsymbol{u}_n - \boldsymbol{u}\rangle = \Delta_\tau \|\boldsymbol{u} - \boldsymbol{u}_n\|_2^2 + \tau \|\Delta_\tau \boldsymbol{u}^2\|_2^2 \tag{6.15}$$

In addition, if the sequence $\{\boldsymbol{v}_n\}$ *is another sequence of vectors, then we have discrete integration formula holds for* $M \in \mathbb{N}$.

$$\tau \sum_{n=1}^{M} (\langle \Delta_\tau \boldsymbol{u}_n, \boldsymbol{v}_n\rangle + \langle \boldsymbol{u}_{n-1}, \Delta_\tau \boldsymbol{v}_n\rangle) = \langle \boldsymbol{u}_M, \boldsymbol{v}_M\rangle - \langle \boldsymbol{u}_0, \boldsymbol{v}_0\rangle \tag{6.16}$$

Next, we need some properties of the sequences generated by the algorithm.

Lemma 6.18. *Let* $\{\boldsymbol{x}_n, \overline{\boldsymbol{x}}_n, \boldsymbol{\xi}_n\}_{n=0}^{\infty}$ *be the sequence generated by the Primal-Dual algorithm, and let* $\boldsymbol{x} \in \mathbb{C}^N$ *and* $\boldsymbol{\xi} \in \mathbb{C}^m$ *be arbitrary. Then, for any* $n \geq 1$

$$\begin{aligned}
&\frac{1}{2}\Delta_\sigma \|\boldsymbol{\xi} - \boldsymbol{\xi}_n\|_2^2 + \frac{1}{2}\Delta_\tau \|\boldsymbol{x} - \boldsymbol{x}_n\|_2^2 + \frac{\sigma}{2}\|\Delta_\sigma \boldsymbol{\xi}_n\|_2^2 + \frac{\tau}{2}\|\Delta_\tau \boldsymbol{x}_n\|_2^2 \\
&\leq L(\boldsymbol{x}, \boldsymbol{\xi}_n) - L(\boldsymbol{x}_n, \boldsymbol{\xi}) + \operatorname{Re}\langle \boldsymbol{A}(\boldsymbol{x}_n - \overline{\boldsymbol{x}}_{n-1}), \boldsymbol{\xi} - \boldsymbol{\xi}_n\rangle
\end{aligned} \tag{6.17}$$

Summing (6.17) in Lemma 6.18 from $n = 1$ to $n = M$, we arrive at the next lemma

Lemma 6.19. *Let $\{\boldsymbol{x}_n, \overline{\boldsymbol{x}}_n, \boldsymbol{\xi}_n\}_{n=0}^{\infty}$ be the sequence generated by the primal-dual algorithm, with $\theta = 1$, and let $\boldsymbol{x} \in \mathbb{C}^N$, $\boldsymbol{\xi} \in \mathbb{C}^m$ be arbitrary. Then, for $M \geq 1$,*

$$\begin{aligned}
&\sum_{n=1}^{M}(L(\boldsymbol{x}_n, \boldsymbol{\xi}) - L(\boldsymbol{x}, \boldsymbol{\xi}_n)) + \frac{1}{2\tau}\|\boldsymbol{x} - \boldsymbol{x}^M\|_2^2 + \frac{1-\sigma\tau\|\boldsymbol{A}\|_2^2}{2\sigma}\|\boldsymbol{\xi} - \boldsymbol{\xi}_M\|_2^2 \\
&+ \frac{1-\sqrt{\sigma\tau}\|\boldsymbol{A}\|_2}{2\tau}\sum_{n=1}^{M-1}\|\boldsymbol{x}_n - \boldsymbol{x}_{n-1}\|_2^2 + \frac{1-\sqrt{\sigma\tau}\|\boldsymbol{A}\|_2}{2\sigma}\sum_{n=1}^{M}\|\boldsymbol{\xi}_n - \boldsymbol{\xi}_{n-1}\|_2^2 \\
&\leq \frac{1}{2\tau}\|\boldsymbol{x} - \boldsymbol{x}_0\|_2^2 + \frac{1}{2\sigma}\|\boldsymbol{\xi} - \boldsymbol{\xi}_0\|_2^2
\end{aligned} \tag{6.18}$$

A corollary of Lemma 6.19 shows that the sequence in Theorem 6.16 is bounded

Corollary 6.20. *Let $(\boldsymbol{x}^\sharp, \boldsymbol{\xi}^\sharp)$ be a primal-dual optimum/saddle point of Equation (6.14). Then the iterates with $\theta = 1$ and $\sigma\tau\|\boldsymbol{A}\|_2 < 1$ satisfy*

$$\frac{1}{2\sigma}\|\boldsymbol{\xi}^\sharp - \boldsymbol{\xi}_M\|_2^2 + \frac{1}{2\tau}\|\boldsymbol{x}^\sharp - \boldsymbol{x}_M\|_2^2 \leq C\left(\frac{1}{2\sigma}\|\boldsymbol{\xi}^\sharp - \boldsymbol{\xi}_0\|_2^2 + \frac{1}{2\tau}\|\boldsymbol{x}^\sharp - \boldsymbol{x}_0\|_2^2\right)$$

Where $C = (1 - \sigma\tau\|\boldsymbol{A}\|_2^2)^{-1}$. In particular, the iterates $(\boldsymbol{x}_n, \boldsymbol{\xi}_n)$ are bounded.

We now have the required tools to prove Theorem 6.16

Proof of Theorem 6.16. Consider the sequence $\{(\boldsymbol{x}_n, \boldsymbol{\xi}_n)\}_{n=0}^{\infty}$. We want to show that it converges to a saddle point $(\boldsymbol{x}^\sharp, \boldsymbol{\xi}^\sharp)$. From Corollary 6.20, we know that it is bounded, and because we are working in a complete metric space, it has a convergent subsequence $(\boldsymbol{x}_{n_k}, \boldsymbol{\xi}_{n_k})$ that converges to a point $(\boldsymbol{x}^\circ, \boldsymbol{\xi}^\circ)$ as $k \to \infty$. Setting $\boldsymbol{x} = \boldsymbol{x}^\sharp$ and $\boldsymbol{\xi} = \boldsymbol{\xi}^\sharp$ in Equation (6.18) in Lemma 6.19, makes all terms nonnegative. Thus any terms on the left hand side can be discarded, and still make the inequality hold. Thus

$$\frac{1-\sqrt{\sigma\tau}\|\boldsymbol{A}\|_2}{2\sigma}\sum_{n=1}^{M-1}\|\boldsymbol{x}_n - \boldsymbol{x}_{n-1}\|_2^2 \leq \frac{1}{2\tau}\|\boldsymbol{x} - \boldsymbol{x}_0\|_2^2 + \frac{1}{2\sigma}\|\boldsymbol{\xi} - \boldsymbol{\xi}_0\|_2^2$$

Both sides of this inequality are positive. In addition, the right hand side does not depend on the number of iterations M. Increasing M will therefore keep the right hand side constant, while increasing the left hand side. Therefore, we must have $\|\boldsymbol{x}_n - \boldsymbol{x}_{n-1}\| \to 0$ as $n \to \infty$. By choosing a different term on the left hand side from Equation (6.18), we also get that $\|\boldsymbol{\xi}_n - \boldsymbol{\xi}_{n-1}\|_2 \to 0$. The point $(\boldsymbol{x}^\circ, \boldsymbol{\xi}^\circ)$ must be a fixed point of the algorithm, and Proposition 6.15 gives us that it is a saddle point.

Choosing $(\boldsymbol{x}, \boldsymbol{\xi}) = (\boldsymbol{x}^\circ, \boldsymbol{\xi}^\circ)$ in Equation (6.17), and summing from $n = n_k$ to $M > n_k$, and following a similar technique as in the proof of Lemma 6.19, we finally get

$$\begin{aligned}
&\frac{1}{2\sigma}\|\boldsymbol{\xi}^\circ - \boldsymbol{\xi}^M\|_2^2 + \frac{1}{2\tau}\|\boldsymbol{x}^\circ - \boldsymbol{x}^M\|_2^2 + \frac{1-\sqrt{\sigma\tau}\|\boldsymbol{A}\|_2}{2\sigma}\sum_{n=n_k}^{M}\|\boldsymbol{\xi}^n - \boldsymbol{\xi}^{n-1}\|_2^2 \\
&+ \frac{1-\sqrt{\sigma\tau}\|\boldsymbol{A}\|_2}{2\tau}\sum_{n=n_k}^{M-1}\|\boldsymbol{x}^n - \boldsymbol{x}^{n-1}\|_2^2 \\
&+ \frac{1}{2\tau}\Big(\|\boldsymbol{x}^M - \boldsymbol{x}^{M-1}\|_2^2 - \sqrt{\sigma\tau}\|\boldsymbol{A}\|_2\|\boldsymbol{x}^{n_k-1} - \boldsymbol{x}^{n_k-2}\|_2^2\Big) \\
&- \operatorname{Re}\langle \boldsymbol{A}(\boldsymbol{x}^M - \boldsymbol{x}^{M-1}), \boldsymbol{\xi}^\circ - \boldsymbol{\xi}^M\rangle + \operatorname{Re}\langle \boldsymbol{A}(\boldsymbol{x}^{n_k-1} - \boldsymbol{x}^{n_k-2}), \boldsymbol{\xi}^\circ - \boldsymbol{\xi}^{n_k}\rangle \\
&\le \frac{1}{2\sigma}\|\boldsymbol{\xi}^\circ - \boldsymbol{\xi}^{n_k}\|_2^2 + \frac{1}{2\tau}\|\boldsymbol{x}^\circ - \boldsymbol{x}^{n_k}\|_2^2
\end{aligned} \tag{6.19}$$

We have that

$$\begin{aligned}
\lim_{n\to\infty}\|\boldsymbol{x}_n - \boldsymbol{x}_{n-1}\|_2 &= \lim_{n\to\infty}\|\boldsymbol{\xi}_n - \boldsymbol{\xi}_{n-1}\|_2 = 0 \\
\lim_{k\to\infty}\|\boldsymbol{x}^\circ - \boldsymbol{x}_{n_k}\|_2 &= \lim_{k\to\infty}\|\boldsymbol{\xi}^\circ - \boldsymbol{\xi}_{n_k}\|_2 = 0
\end{aligned}$$

And it follows that

$$\lim_{M\to\infty}\|\boldsymbol{x}^\circ - \boldsymbol{x}_M\|_2 = \lim_{M\to\infty}\|\boldsymbol{\xi}^\circ - \boldsymbol{\xi}_M\|_2 = 0$$

because the expression on the right hand side of the equation converges to zero. ■

CHAPTER 7

Writing compressive sensing algorithms as neural networks

In this section, we are going to investigate the structure of the iterative algorithms from Chapter 6 when they are unrolled. We will see that affine maps and non-linear functions are important for the algorithms. Therefore, we are going to see that they resemble a neural network.

It turns out that writing these algorithms as neural networks by strictly following Definition 5.1, is not possible. However, we are going to allow ourselves to perform some slight modifications to the definition. The changes will be made as small as possible, in an attempt to maintain the notion of what a neural network is.

Changing the definition as we see fit, might of course seem counterproductive, as the existing definition may be useful for e.g. theoretical analysis. We allow ourselves to still call the modified definition neural networks because modern neural networks rarely follow Definition 5.1. For instance, they often allow *skip connections*, *running several networks in parallel*, *max pooling* and other techniques we cannot express using the classical definition, while still having "neural network-like" structure.

The motivation behind the results in this chapter is that it makes an implementations in deep learning frameworks feasible, and as a consequence, we can test the stability conjecture supported by Theorem 4.15.

7.1 FISTA as a neural network

Before showing the result for FISTA, we are first going to consider an easier case, namely ISTA, which FISTA was inspired from [BT09; FR13]. The algorithm consist of a single iteration step, given below

Algorithm 7.1 ISTA with constant stepsize

```
Input:  $L := L(f)$ – A Lipschitz constant of $\nabla f$
Step 0:  Take $\boldsymbol{x}_0 \in \mathbb{R}^N$. Typically $\boldsymbol{x}_0 := \boldsymbol{A}^*\boldsymbol{y}$
Step $k$:  $k \geq 1$
    $\boldsymbol{x}_{k+1} = p_L(\boldsymbol{x}_k)$    (7.1)
```

ISTA solves the same problem as FISTA, so we will use the same setup as we did in Section 6.1, i.e. we have $f(\boldsymbol{z}) = \|\boldsymbol{A}\boldsymbol{z} - \boldsymbol{b}\|_2^2$, $g(\boldsymbol{z}) = \lambda\|\boldsymbol{z}\|_1$. We recall that this gives the proximal operator

$$p_L(\boldsymbol{x}) = \mathcal{S}_{\lambda/L}(\boldsymbol{x} - L^{-1}\nabla f(\boldsymbol{x}))$$

We want to write the iteration step (7.1) as a one-layered neural network. When that is established, ISTA generalizes to a multi-layered neural network by increasing the number of iterations. Indeed, the desired result is given in the following lemma:

Lemma 7.1. *The application of the proximal operator in Equation* (7.1) *can be written as forward propagation through a layer in a neural network. In other words*

$$p_L = (\rho_{ISTA} \circ \mathcal{W}_{ISTA})$$

where

$$\begin{aligned}
\mathcal{W}_{ISTA}(\boldsymbol{x}) &= W_{ISTA}\boldsymbol{x} + b_{ISTA} \\
W_{ISTA} &= (\boldsymbol{I} - 2L^{-1}\boldsymbol{A}^*\boldsymbol{A}) \\
b_{ISTA} &= 2L^{-1}\boldsymbol{A}^*\boldsymbol{y} \\
\rho_{ISTA} &= \mathcal{S}_{\lambda/L}
\end{aligned}$$

If we allow the bias term b_{ISTA} to depend on the measurement vector $\boldsymbol{y}$.

Proof. We have to show that

$$\mathcal{S}_{\lambda/L}(\boldsymbol{x} - L^{-1}\nabla f(\boldsymbol{x})) = \rho_{\text{ISTA}}(\mathcal{W}_{\text{ISTA}}(\boldsymbol{x}))$$

By the definition of ρ_{ISTA}, we have that

$$\rho_{\text{ISTA}}(\mathcal{W}_{\text{ISTA}}(\boldsymbol{x})) = \mathcal{S}_{\lambda/L}(\mathcal{W}_{\text{ISTA}}(\boldsymbol{x}))$$

By definition, this acts elementwise on its input, and is nonlinear. We also have that $\mathcal{W}_{\text{ISTA}}(\boldsymbol{x}) = \boldsymbol{W}_{\text{ISTA}}\boldsymbol{x} + \boldsymbol{b}_{\text{ISTA}}$ is an affine map, because $\boldsymbol{W}_{\text{ISTA}}$ is a matrix, and $\boldsymbol{b}_{\text{ISTA}}$ is a vector.

It therefore remains to show that $\mathcal{W}_{\text{ISTA}} = \boldsymbol{x} - L^{-1}\nabla f(\boldsymbol{x})$. Indeed, by applying the definitions and factoring the result, we get

$$\begin{aligned}
\mathcal{W}_{\text{ISTA}}(\boldsymbol{x}) &= \boldsymbol{W}_{\text{ISTA}}\boldsymbol{x} + \boldsymbol{b}_{\text{ISTA}} \\
&= (\boldsymbol{I} - 2L^{-1}\boldsymbol{A}^*\boldsymbol{A})\boldsymbol{x} + 2L^{-1}\boldsymbol{A}^*\boldsymbol{y} \\
&= \boldsymbol{x} - 2L^{-1}\boldsymbol{A}^*\boldsymbol{A}\boldsymbol{x} + 2L^{-1}\boldsymbol{A}^*\boldsymbol{y} \\
&= \boldsymbol{x} - 2L^{-1}\boldsymbol{A}^*(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}) \\
&= \boldsymbol{x} - L^{-1}\nabla f(\boldsymbol{x})
\end{aligned}$$

Which completes the proof. ■

ISTA fits rather nicely into the classical definition of neural networks given in Definition 5.1. Lemma 7.1 provides appropriate affine mapping and nonlinear functions that will lets us write ISTA as a neural network.

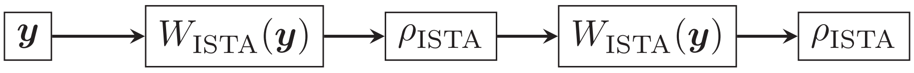

$\boldsymbol{y}$
$W_{\text{ISTA}}(\boldsymbol{y})$
ρ_{ISTA}
$W_{\text{ISTA}}(\boldsymbol{y})$
ρ_{ISTA}

Figure 7.1: Two layers of ISTA as as neural network where the weights functions of the measurements.

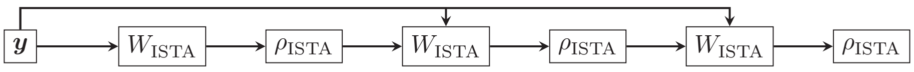


Figure 7.2: Three layers of ISTA as a neural network where the dependency is illustrated using skip connections.

However, Lemma 7.1 makes an important assumption: The bias term in $\mathcal{W}_{\text{ISTA}}$ must depend on the measurements $\boldsymbol{y}$. This leads to *skip connections*, meaning that the input, in addition to being passed through the network in the way expected, it will also skip some nodes, and be passed directly to some of the nodes. This is illustrated in Figure 7.2.

We are now ready to attempt writing FISTA as a neural network. This will be done according to the following definition

Definition 7.2 (Neural networks)**.** A neural network Φ is a function composed of N function f_i such that is on the form

$$\Phi(\boldsymbol{x}) = f_N(f_{N-1}(f_{N-2}(\dots f_1(\boldsymbol{x})\dots))) \tag{7.2}$$

Where each f_i is on one of the following forms:

- An affine map $\mathcal{W}$.
- An activation function $\boldsymbol{\xi} \mapsto s(\boldsymbol{\xi})\rho(\boldsymbol{\xi})$, where
  - ρ is a nonlinear function acting componentwise on its input.
  - $s\colon \mathbb{C}^m \to \mathbb{R}$ is a function that will scale the output of ρ
- A function $\mathcal{L}$ that computes the linear combination

  $$\mathcal{L}(\boldsymbol{x}, \boldsymbol{z}, \alpha, \beta) = g(\alpha, \beta)\boldsymbol{x} + h(\alpha, \beta)\boldsymbol{z}$$

  Where the functions $g, h : \mathbb{R}^2 \to \mathbb{R}$ compute the coefficients.

In addition, we require that if f_i is an affine map, then f_{i+1} must be an activation function

We also allow skip connections to make e.g. $\mathcal{L}$ usable in Equation (7.2). We do this by additionally allowing the functions f_i to take the output of any f_j for any $j < i$ as additional inputs.

The requirement of always having activation functions after affine maps mainly stems from the wish to preserve a similar structure as in Definition 5.1. The importance of the nonlinearity in the activation functions comes from the fact that if $\mathcal{W}_1$ and $\mathcal{W}_2$ are affine maps, then both $\mathcal{W}_1 + \mathcal{W}_2$ and $\mathcal{W}_1 \circ \mathcal{W}_2$ are affine maps. Thus, a neural network considering of only affine maps, no matter how many, will just be an affine map. In other words, having repeating affine maps reduces to a single affine map, and that approach will therefore

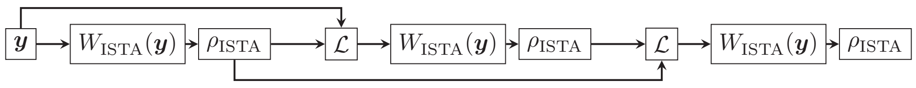


Figure 7.3: FISTA

be wasteful from a computational point of view, as expressing the series of maps as a single one will provide the same amount of expressiveness and can be computed faster.

When writing FISTA as a neural network, we are going to need the following definition

Definition 7.3 (Linear combination node for FISTA)**.** We define the following linear combination node $\mathcal{L}$

$$\begin{aligned}\mathcal{L}(x_k, x_{k-1}, t_k, t_{k-1}) &= \boldsymbol{x}_k + \left(\frac{t_k - 1}{t_{k+1}}\right)(\boldsymbol{x}_k - \boldsymbol{x}_{k-1}) \\ &= \left(1 + \frac{t_k - 1}{t_{k+1}}\right)\boldsymbol{x}_k + \left(\frac{t_k - 1}{t_{k+1}}\right)\boldsymbol{x}_{k-1}\end{aligned}$$

This is Equation (6.6).

We are now ready to describe FISTA as a neural network. We will see that the discussion about ISTA proves useful when doing this, as the proximal operator is the same in both cases, and so

Theorem 7.4 (FISTA as a neural network)**.** *The equations* (6.4) *and* (6.5) *can be written as a neural network according to Definition 7.2 with* $\mathcal{W}_{FISTA} = \mathcal{W}_{ISTA}$ *and* $\rho_{FISTA} = \rho_{ISTA}$ *given by Lemma 7.1 and* $\mathcal{L}$ *given by Definition 7.3. The composite function*

$$\rho_{FISTA}(\mathcal{W}_{FISTA}(\mathcal{L}(\boldsymbol{x}_k, \boldsymbol{x}_{k-1}, t_k, t_{k-1}))$$

Is one layer of a neural network that computes one iteration of FISTA if we set $\boldsymbol{x}_{-1} \coloneqq \boldsymbol{x}_0$ *and* $t_0 \coloneqq t_1$.

Proof. We first note that defining t_0 and $\boldsymbol{x}_{-1}$ allows the first evaluation of $\mathcal{L}$ to return its first input $\boldsymbol{x}_1$, as the algorithm does not actually perform the linear combination until the end of the first iteration.

Due to the similarities between ISTA and FISTA, we can apply Lemma 7.1 for the affine mapping and the nonlinear function. For the linear combination, we see that it computes $\overline{\boldsymbol{x}}$, according to Equation (6.6), which is the input to Equation (6.4). ■

FISTA, using $\mathcal{L}$ is illustrated in Figure 7.3. Note that the figure uses function notation for the dependency on $\boldsymbol{y}$ in $\mathcal{W}_{\text{ISTA}}$. This is done mainly to focus on the structure involving $\mathcal{L}$, and avoid clutter.

At this point, the function in Definition 7.3 does only implicitly compute the coefficients in the linear combination. These coefficients involving t_k are not arbitrary, and need to be computed.

The sequence $\{t_k\}_{k=1}^{\infty}$ in the context of FISTA has a major advantage as it does not depend on anything other that the value from the previous iteration, therefore, the sequence can be computed and set when building the network.

$$\begin{tikzcd}
\boxed{t_1} \arrow[r] \arrow[drrr] & \boxed{\mathcal{W}_{\mathrm{t}}} \arrow[r] & \boxed{\rho_{\mathrm{t}}} \arrow[r] & \boxed{t_2} \arrow[r] \arrow[d] \arrow[drrr] & \boxed{\mathcal{W}_{\mathrm{t}}} \arrow[r] & \boxed{\rho_{\mathrm{t}}} \arrow[r] & \boxed{t_3} \arrow[d] & & \\
\boxed{\boldsymbol{y}} \arrow[r] \arrow[rrr, bend left] & \boxed{\mathcal{W}_{\mathrm{ISTA}}(\boldsymbol{y})} \arrow[r] & \boxed{\rho_{\mathrm{ISTA}}} \arrow[r] \arrow[rrrr, bend right] & \boxed{\mathcal{L}} \arrow[r] & \boxed{\mathcal{W}_{\mathrm{ISTA}}(\boldsymbol{y})} \arrow[r] & \boxed{\rho_{\mathrm{ISTA}}} \arrow[r] & \boxed{\mathcal{L}} \arrow[r] & \boxed{\mathcal{W}_{\mathrm{ISTA}}(\boldsymbol{y})} \arrow[r] & \boxed{\rho_{\mathrm{ISTA}}}
\end{tikzcd}$$

Figure 7.4: FISTA with t_k computed in parallel

Another approach is to observe that the sequence $\{t_k\}$ can be computed using a neural network. By setting

$$\mathcal{W}_{\mathrm{t}}(t) = t \tag{7.3}$$

$$\rho_{\mathrm{t}}(t) = \frac{1 + \sqrt{1 + 4t^2}}{2} \tag{7.4}$$

Thus, the function $(\mathcal{W}_{\mathrm{t}} \circ \rho_{\mathrm{t}})$ composed with itself k times will give us t_k. In addition, $\mathcal{W}_{\mathrm{t}}$ is an affine map, as it is linear, and ρ_{t} is a non-linear function, because of both the square and the square root. It also acts componentwise, as t_k only have one component.

Thus, we can compute FISTA by running two neural networks in parallel. This is illustrated in Figure 7.4

We conclude this section by noting that FISTA indeed can be written as a neural network using the extended definition. We do note, however, that this is only the case when we compute $\{t_k\}$ outside of the network. The technique illustrated in Figure 7.4 does not fit the definition.

7.2 Chambolle–Pock as a neural network

Let us now consider Chambolle and Pock's primal-dual algorithm, as discussed in Section 6.2. We want to write this as a neural networking according to the above modified definition.

We recall that the iteration steps in the Primal-Dual algorithm.

$$\boldsymbol{\xi}_{k+1} = P_{F^*}(\sigma; \boldsymbol{\xi}_k + \sigma \boldsymbol{A} \overline{\boldsymbol{x}}_k) \tag{PD$_1$}$$

$$\boldsymbol{x}_{k+1} = P_G(\tau; \boldsymbol{x}_k - \tau \boldsymbol{A}^* \boldsymbol{\xi}_{k+1}) \tag{PD$_2$}$$

$$\overline{\boldsymbol{x}}_{k+1} = \boldsymbol{x}_{k+1} + \theta(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) \tag{PD$_3$}$$

This algorithm is quite similar to FISTA, but is also more complicated. Instead of just computing the primal solution, as FISTA does, the dual solution is also computed. In addition, these two sequences cannot be computed independently, because they heavily depend on each other.

Because of this dependency, it is better to consider each of the two equations as one layer in the network. This way, Equation (PD$_1$) will be the calculation done in the odd numbered layers, and Equation (PD$_2$) will be the even numbered layers. After each even numbered layer, we need to compute a linear combination, namely Equation (PD$_3$), which we will define by the following function

Definition 7.5 (Linear combination node for Chambolle-Pock)**.** The function $\mathcal{L}$ is given as

$$\begin{aligned}\mathcal{L}(\boldsymbol{x}_{n+1}, \boldsymbol{x}_n, \theta) &= \boldsymbol{x}_{n+1} + \theta(\boldsymbol{x}_{n+1} - \boldsymbol{x}_n) \\ &= (1+\theta)\boldsymbol{x}_{n+1} - \theta \boldsymbol{x}_n\end{aligned}$$

Because we are mostly interested in the primal solution $\boldsymbol{x}$, it is interesting to write out Equation (PD_2) in terms of the other equations in the algorithms. We get

$$\boldsymbol{x}_{n+1} = P_G(\boldsymbol{x}_n - \tau \boldsymbol{A}^*(P_{F^*}(\boldsymbol{\xi}_n + \sigma \boldsymbol{A}\mathcal{L}(\boldsymbol{x}_n, \boldsymbol{x}_{n-1})))) \tag{7.5}$$

Which motivates the following theorem

Theorem 7.6 (Chambolle–Pock as a neural network)**.** *One iteration of the Chambolle–Pock primal-dual algorithm described in Algorithm 6.2 can be written as a two-layered neural network according to Definition 7.2, i.e.*

$$\rho_P(\mathcal{W}_P(\rho_D(\mathcal{W}_D(\mathcal{L}(\boldsymbol{x}_n, \boldsymbol{x}_{n-1}, \theta))))) \tag{7.6}$$

where

$$\begin{aligned}\rho_P &= P_G \\ \rho_D &= P_{F^*} \\ \mathcal{W}_P(\boldsymbol{z}) &= \boldsymbol{W}_P \boldsymbol{z} + \boldsymbol{b}_P \\ \mathcal{W}_D(\boldsymbol{z}) &= \boldsymbol{W}_D \boldsymbol{z} + \boldsymbol{b}_D \\ \boldsymbol{W}_P &= -\tau \boldsymbol{A}^* \\ \boldsymbol{b}_P &= \boldsymbol{x}_n \\ \boldsymbol{W}_D &= \sigma \boldsymbol{A} \\ \boldsymbol{b}_D &= \boldsymbol{\xi}_n\end{aligned}$$

Composing Equation (7.6) *with itself n times yield a neural network with $2n$ layers that computes n iterations of Algorithm 6.2. It outputs only the primal solution.*

Proof. From Equation (7.5), it is quite easy to see that Equation (7.6) computes one iteration of the primal-dual algorithm. It therefore only remains to show that it is a neural network according to Definition 7.2.

We begin by showing that the functions $\rho_{\mathrm{P}}, \rho_{\mathrm{D}}, \mathcal{W}_{\mathrm{P}}, \mathcal{W}_{\mathrm{D}}$ are functions that the definition allow. Indeed $\mathcal{W}_{\mathrm{P}}$ and $\mathcal{W}_{\mathrm{D}}$ are affine mappings. They do depend on skip connections for the bias, which is allowed.

Further, we observe that $P_G(\tau, \boldsymbol{z}) = \mathcal{S}_\tau(\boldsymbol{z})$ is similar to the activation function for FISTA, and is therefore allowed.

Consider P_{F^*}. Recall that it is defined by

$$P_{F^*}(\sigma; \boldsymbol{\xi}) = \begin{cases} \boldsymbol{0} & \text{if } \|\boldsymbol{\xi} - \sigma \boldsymbol{y}\|_2 \leq \eta\sigma \\ \left(1 - \frac{\eta\sigma}{\|\boldsymbol{\xi} - \sigma\boldsymbol{y}\|_2}\right)(\boldsymbol{\xi} - \sigma\boldsymbol{y}) & \text{otherwise} \end{cases}$$

We observe that $1 - \frac{\eta\sigma}{\|\boldsymbol{\xi} - \sigma\boldsymbol{y}\|_2} < 0$ when $\|\boldsymbol{\xi} - \sigma\boldsymbol{y}\|_2 \leq \eta\sigma$. Thus, we can write

$$P_{F^*}(\sigma; \boldsymbol{\xi}) = \max\left\{0, 1 - \frac{\eta\sigma}{\|\boldsymbol{\xi} - \sigma\boldsymbol{y}\|_2}\right\}(\boldsymbol{\xi} - \sigma\boldsymbol{y})$$

The mapping $\boldsymbol{\xi} \mapsto \boldsymbol{\xi} - \sigma \boldsymbol{y}$ is nonlinear, and it acts componentwise on $\boldsymbol{\xi}$. As the output is scaled by the value of the max function, we have that P_F^* is an activation function according to Definition 7.2.

Finally, we observe that whenever an affine map is computed, it is followed by an activation function. Thus, we also satisfy the requirement that we cannot have two affine maps in succession.

■

7.3 Adaptive neural networks

We observe that the neural networks arising from unrolling the two algorithms form networks where the weight layers $\mathcal{W}_{\mathrm{FISTA}}$, $\mathcal{W}_{\mathrm{P}}$ and $\mathcal{W}_{\mathrm{D}}$ depend on the measurement vectors $\boldsymbol{y}$ and/or the measurement matrix $\boldsymbol{A}$. In other words, these networks change their weights to accommodate the input. Therefore, they do not require any training. With the difficulties of training a neural network in mind, this might provide an advantage, especially with the theory described in Chapter 4 and 6 guaranteeing recovery. Because the weights *adapts* to to the input of the network, we refer this class of neural networks as Adaptive Neural Networks (ANNs).

CHAPTER 8

Numerical results

8.1 Setup

Treating the optimization algorithms for compressive sensing as neural networks in Chapter 7 motivates implementing them as neural networks as well. In recent years, this has typically been done by using frameworks such as Tensorflow [TF]. With these frameworks, we can program GPUs by using high-level languages such as Python [Py3]. For programmers with experience with other frameworks for scientific computing, such as SciPy [SciPy], this is done with relative ease, compared to more low-level languages such as C. Tensorflow is also designed with deep learning in mind, and can therefore calculate gradients for us, which makes us able to perform stability tests.

Every example of recovery with compressive sensing in this chapter is done with the author's Tensorflow implementations, unless otherwise noted. The sampling patterns have been generated using the cslib MATLAB library, which can be found at https://bitbucket.org/vegarant/cslib

Table 8.1 shows a legend with information about the hardware used to run experiments. To avoid cluttering our figures and tables, we refer to the hardware using the keys in the legend.

Key	Description
GPU	NVIDIA GeForce GTX 1080
CPU	Intel Core i7-4790K CPU (4.00 GHz)

Table 8.1: Legend for the keys in Table 8.2 and Table 8.3

8.2 Speed

A welcomed consequence of running the code on a GPU is that we are able to get a significant speedup, compared to running similar code on a CPU.

Table 8.2 and Table 8.2 along with the plots in Figure 8.1 and Figure 8.2 show running times for FISTA and Chambolle–Pock, implemented both in Tensorflow and with Numpy. For each of the values $256, 512, 1024$ and 2048, ten experiments have been performed, and the resulting datapoints shown are the median. We choose the median over the mean for the following reasons: The first run of a Tensorflow graph is usually slower than subsequent runs because

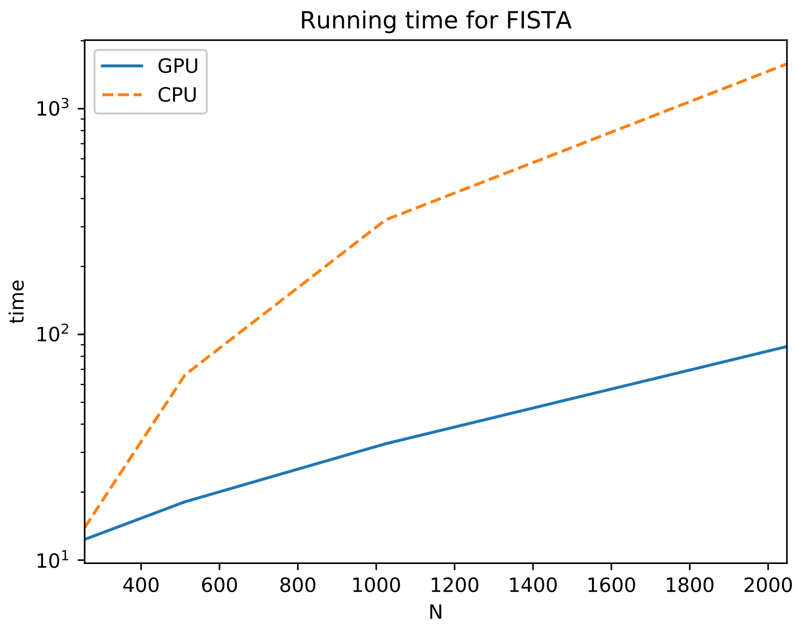


Figure 8.1: Fista time plot

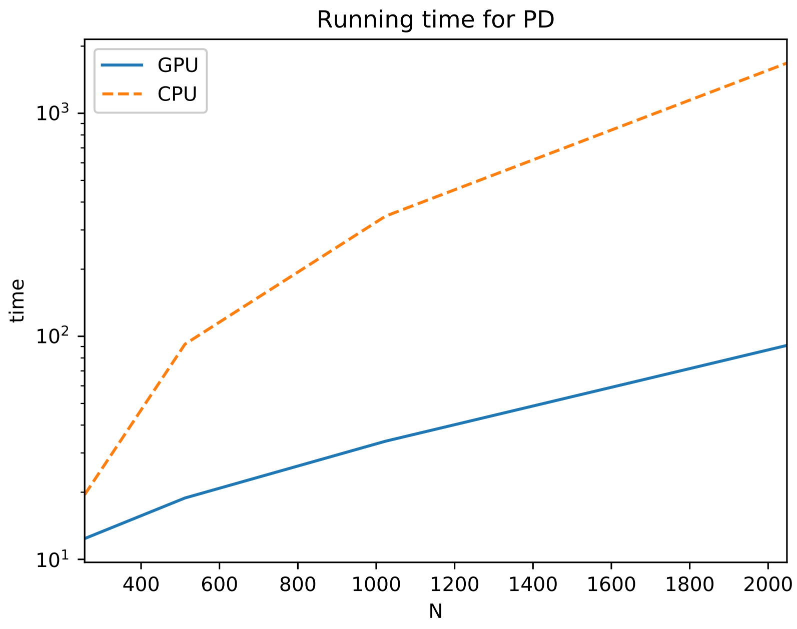


Figure 8.2: pd time plot

N	GPU	CPU
256	12.35	13.93
512	18.13	66.17
1024	32.77	321.86
2048	88.39	1579.44

Table 8.2: Running times in seconds for FISTA with different values of N

N	GPU	CPU
256	12.39	19.50
512	18.84	92.23
1024	33.87	346.01
2048	91.05	1679.67

Table 8.3: Running times in seconds for Chambolle–Pock with different values of N

it has not yet been optimized. The median is more robust with respect to extreme values, and will therefore not be skewed by this potential outlier. Such values might arise from simultaneous usage of the computational resources the computations have been made on.

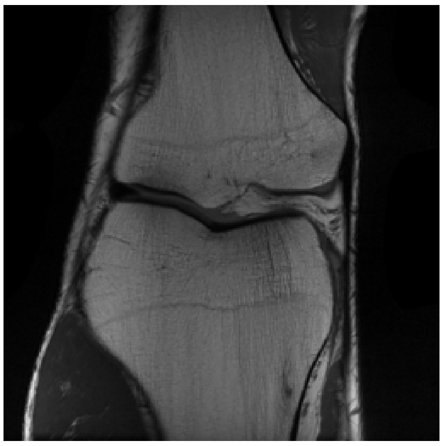
We see that the Tensorflow implementations running on GPU is faster than the Numpy implementations. However, the difference for small N is perhaps not as extreme as expected. This might be due to parallelization done by Numpy. When the problem size grows, Tensorflow outperforms Numpy with an order of magnitude for $N = 2048$. To emphasize the significance of this, we note that the combined time spend computing the experiments for Tensorflow was approximately one hour, while the CPU implementation took a total of 11 hours to complete. For running several large scale experiments, the GPU implementation is therefore clearly preferred.

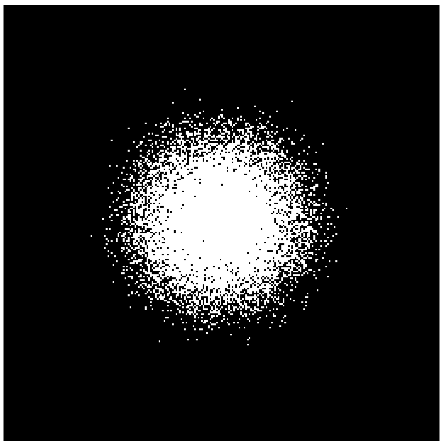
8.3 Successful recovery

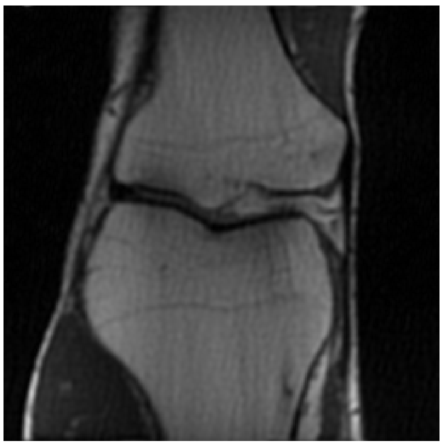
Figure 8.3 shows the result of a practical experiment using 12.5% of the original samples. We see that as a consequence of the subsampling, some details are lost when we recover by taking the adjoint in Figure 8.3c. These details are recovered in Figure 8.3d.

8.4 Superresolution

An experiment that better illustrates the usefulness of compressive sensing can be seen in Figure 8.5. In Figure 8.6, we have added text to the Shepp–Logan phantom image of size 2048×2048, which has then been rescaled to 512×512. As we can see in Figure 8.5a, we have lost a significant amount of detail, and thus the text is illegible. This corresponds to classical methods, where we have to sample the entire 512×512 grid in Figure 8.5e.


(a) Original image


(b) Sampling pattern


(c) Adjoint

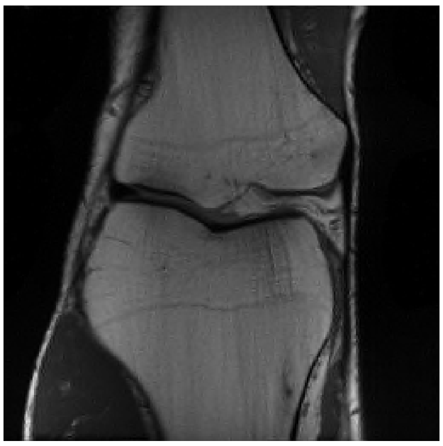
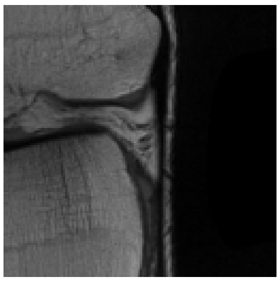
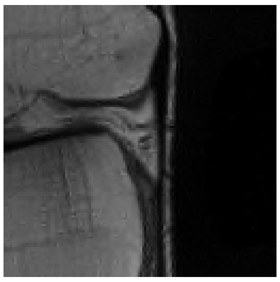

(d) Recovery

Figure 8.3: Recovery using 12.5% of the samples


(a) Original zoomed


(b) Recovered zoomed

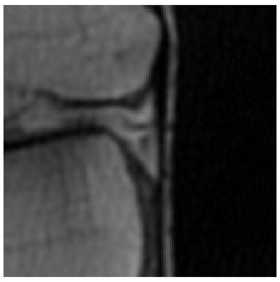

(c) Adjoint zoomed

Figure 8.4: Zooming Figure 8.3

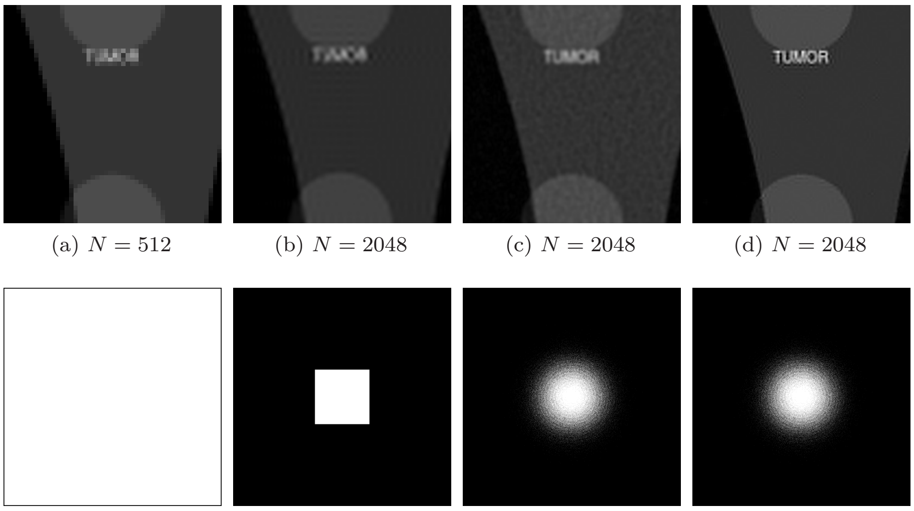


Figure 8.5: Recovery of details with compressive sensing

From the same measurements, we obtain Figure 8.5b by padding the Fourier measurements. The recovery using the IDFT still provides illegible text. By more carefully selecting our samples, we obtain Figure 8.5c, which greatly improve the adjoint. Lastly, we apply compressive sensing to the measurements in Figure 8.5c, to get an image where the text is legible Figure 8.5d. We note that this is recovery using 6.25% of the original measurements. In addition, we obtain better resolution than 100% sampling for the 512×512 case.

8.5 Stability

In Chapter 4, we looked at theoretical results regarding compressive sensing. In particular, Theorem 4.15 guarantees that the upper bound on the recovery error scales linearly. Therefore, if we perturb the input, we would not expect the error to become drastically different. In other words, compressive sensing is stable with respect to perturbations. Figure 8.7 recreates a figure from [Ant+19], where we apply a series of perturbations $\boldsymbol{r}_1, \dots, \boldsymbol{r}_4$ with increasing norm to an image to investigate if the neural network AUTOMAP from [Zhu+18] is stable. Figures (8.7a) through (8.7j) are taken from [Ant+19]. The first row of images visualizes the MRI data with the different perturbations added. The second rows displays the recovery of using AUTOMAP (AM) to recover the corresponding image in the top row from 60 % Fourier subsampling.

AUTOMAP performs well in the noiseless case, but performs poorly already when the smallest perturbation is applied. For the remaining cases, the noise causes the network to perform even worse. In few of the cases, we are unable to see the outline of the heart added as a detail, and in the worst cases, like Figure 8.7j, we also unable to separate what is the added detail, and what is artifacts arising from the added noise.

The last row shows the recovery compressive sensing (CS) using the exact

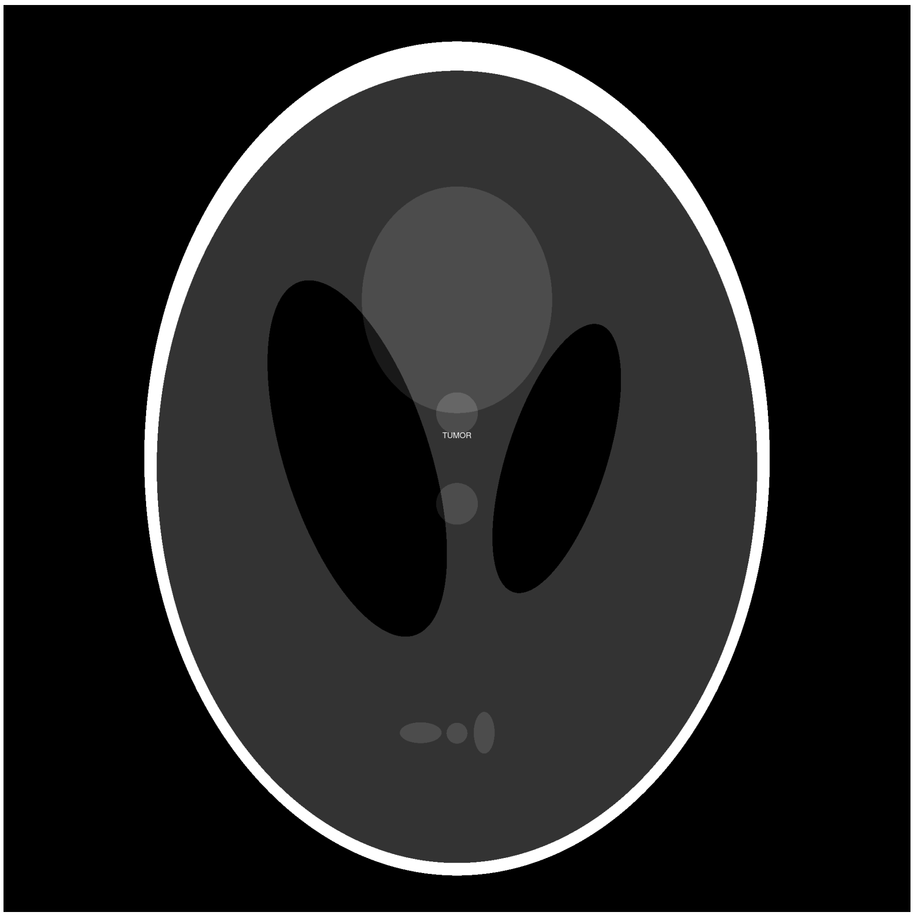


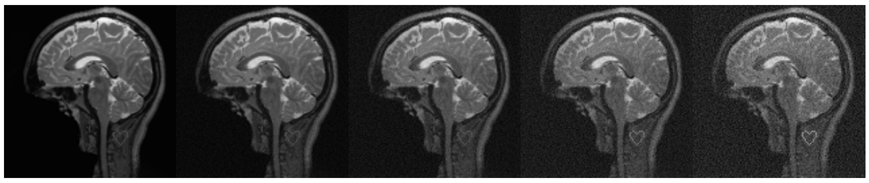
Figure 8.6: The full 2048 × 2048 Shepp–Logan phantom with text

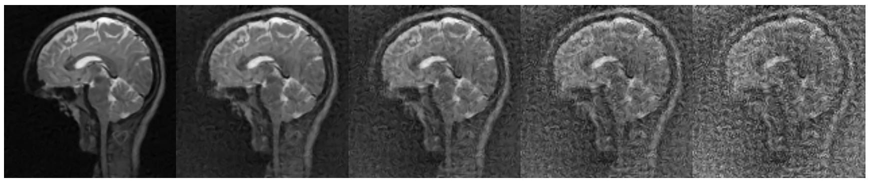
same data as was put into AUTOMAP in the second row. We see that the output is more or less identical to the input images. This is good news, as it is an indication that we in fact have stable recovery. However, this is not a fair comparison. The noise applied in the figure is generated to make the error in the output as large as possible when recovered with AUTOMAP.

Instead, we attempt to find worst-case noise for compressive sensing on the same scale as the perturbations in Figure 8.7. We consider the same problem as in [Ant+19], given in Equation (8.1)

$$\operatorname{argmax}_{\boldsymbol{r}} \quad \frac{1}{2}\|f(\boldsymbol{y}+\boldsymbol{A}\boldsymbol{r})-f(\boldsymbol{y})\|_2^2-\frac{\lambda}{2}\|\boldsymbol{r}\|_2^2 \tag{8.1}$$

When solving this optimization problem, we are seeking a small perturbation $\boldsymbol{r}$ that makes the difference between the noiseless recovery $f(\boldsymbol{y})$ and the recovery with the noise added $f(\boldsymbol{y}+\boldsymbol{A}\boldsymbol{r})$ large. We solve this problem iteratively using Nesterov accelerated gradient descent. To make the comparison as fair as possible, we terminate the algorithm when the ℓ_2-norm of the perturbation is larger than the perturbations used in Figure 8.7.


(a) $|\boldsymbol{x}|$ (b) $|\boldsymbol{x}+\boldsymbol{r}_1|$ (c) $|\boldsymbol{x}+\boldsymbol{r}_2|$ (d) $|\boldsymbol{x}+\boldsymbol{r}_3|$ (e) $|\boldsymbol{x}+\boldsymbol{r}_4|$


(f) AM $|\boldsymbol{x}|$ (g) AM $|\boldsymbol{x}+\boldsymbol{r}_1|$ (h) AM $|\boldsymbol{x}+\boldsymbol{r}_2|$ (i) AM $|\boldsymbol{x}+\boldsymbol{r}_3|$ (j) AM $|\boldsymbol{x}+\boldsymbol{r}_4|$

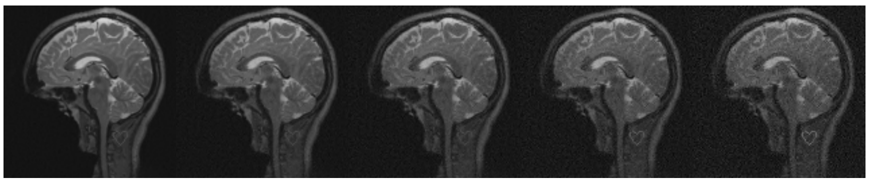
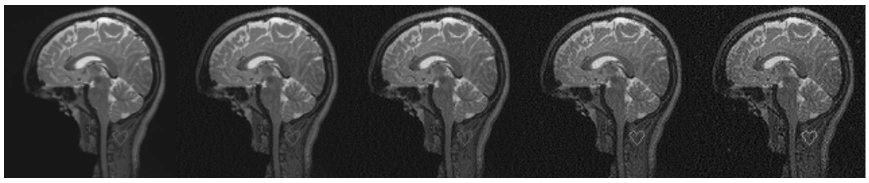

(k) CS $|\boldsymbol{x}|$ (l) CS $|\boldsymbol{x}+\boldsymbol{r}_1|$ (m) CS $|\boldsymbol{x}+\boldsymbol{r}_2|$ (n) CS $|\boldsymbol{x}+\boldsymbol{r}_3|$ (o) CS $|\boldsymbol{x}+\boldsymbol{r}_4|$

Figure 8.7: Stability compared to AUTOMAP.


(a) $|\boldsymbol{x}|$ (b) $|\boldsymbol{x}+\boldsymbol{r}_1|$ (c) $|\boldsymbol{x}+\boldsymbol{r}_2|$ (d) $|\boldsymbol{x}+\boldsymbol{r}_3|$ (e) $|\boldsymbol{x}+\boldsymbol{r}_4|$

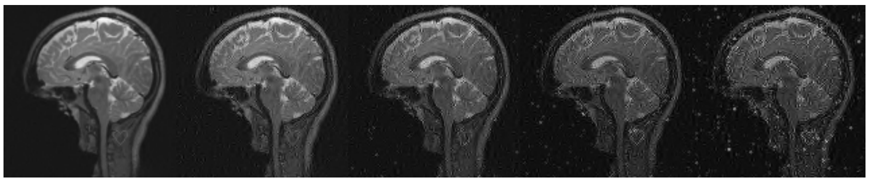
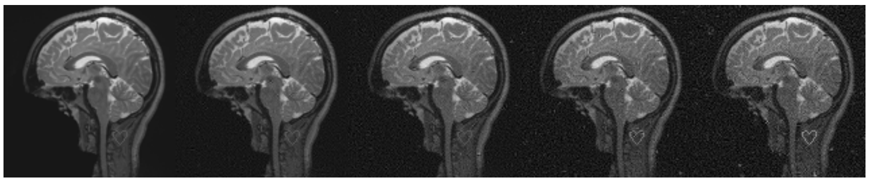
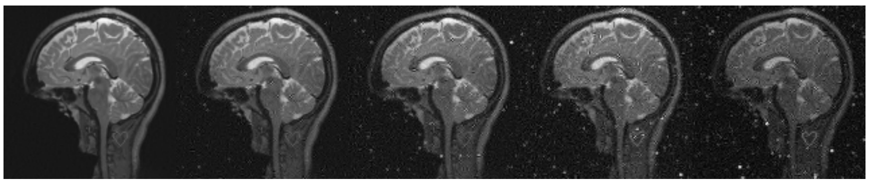

(f) CS $\boldsymbol{A}\boldsymbol{x}$ (g) CS $\boldsymbol{A}(\boldsymbol{x}+\boldsymbol{r}_1)$(h) CS $\boldsymbol{A}(\boldsymbol{x}+\boldsymbol{r}_2)$(i) CS $\boldsymbol{A}(\boldsymbol{x}+\boldsymbol{r}_3)$(j) CS $\boldsymbol{A}(\boldsymbol{x}+\boldsymbol{r}_4)$

Figure 8.8: Stability of FISTA

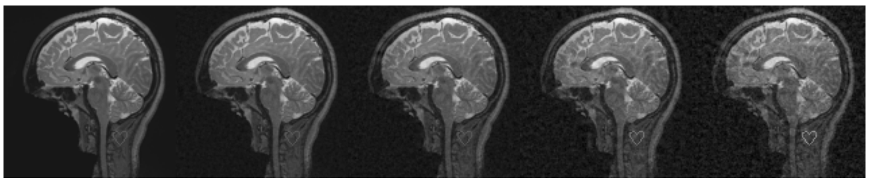


(a) $|\boldsymbol{x}|$ (b) $|\boldsymbol{x}+\boldsymbol{r}_1|$ (c) $|\boldsymbol{x}+\boldsymbol{r}_2|$ (d) $|\boldsymbol{x}+\boldsymbol{r}_3|$ (e) $|\boldsymbol{x}+\boldsymbol{r}_4|$

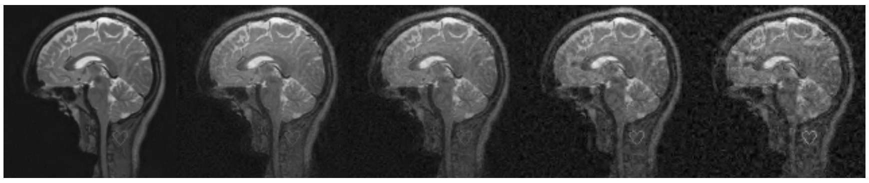


(f) CS $\boldsymbol{A}\boldsymbol{x}$ (g) CS $\boldsymbol{A}(\boldsymbol{x}+\boldsymbol{r}_1)$(h) CS $\boldsymbol{A}(\boldsymbol{x}+\boldsymbol{r}_2)$(i) CS $\boldsymbol{A}(\boldsymbol{x}+\boldsymbol{r}_3)$ (j) CS $\boldsymbol{A}(\boldsymbol{x}+\boldsymbol{r}_4)$

Figure 8.9: Stability of Chambolle–Pock



(a) $|\boldsymbol{x}|$ (b) $|\boldsymbol{x}+\boldsymbol{r}_1|$ (c) $|\boldsymbol{x}+\boldsymbol{r}_2|$ (d) $|\boldsymbol{x}+\boldsymbol{r}_3|$ (e) $|\boldsymbol{x}+\boldsymbol{r}_4|$



(f) CS $\boldsymbol{A}\boldsymbol{x}$ (g) CS $\boldsymbol{A}(\boldsymbol{x}+\boldsymbol{r}_1)$(h) CS $\boldsymbol{A}(\boldsymbol{x}+\boldsymbol{r}_2)$(i) CS $\boldsymbol{A}(\boldsymbol{x}+\boldsymbol{r}_3)$ (j) CS $\boldsymbol{A}(\boldsymbol{x}+\boldsymbol{r}_4)$

Figure 8.10: Stability of square root LASSO

Figures 8.8 and 8.9 shows the results of stability experiments for FISTA and Chambolle–Pock, respectively. The top row in each figure shows the original images with added noise, and the bottom rows shows the resulting recovery from the same 60% sampling pattern used for Figure 8.7.

We can immediately see that the algorithms do not perform as well as when we applied the noise generated for AUTOMAP. Quite surprisingly, we get a lot of artifacts. The noise is most severe for FISTA, where we lose a lot of details around the added heart detail. The noise when using Chambolle–Pock for recovery is not as severe. Similarly to FISTA, we experience that small artifacts added by the noise are amplified in the recovery. Compare for instance the brightest dots in Figure 8.9j and compare the brightness with the original image Figure 8.9e. These sorts of artifacts can obscure details, for instance in Figure 8.9i

To investigate if a more stable form of the algorithms previously considered in this thesis exist, a stability test of Chambolle–Pock with the square root LASSO (8.2) problem was also performed. We omit any theoretical analysis, derivations and discussion of this problem due to time constraints.

$$\min_{z} \quad \lambda\|\boldsymbol{z}\|_1 + \|\boldsymbol{A}\boldsymbol{z} - \boldsymbol{y}\|_2 \tag{8.2}$$

Surprisingly, we see that it performs much better than the other two algorithms. We see slight amplification of the noise, but the outputs are nearly identical to the input images. This is good news, and a clear indication of stability.

CHAPTER 9

Conclusion

This thesis started with a presentation of the basic compressive sensing problem: Recovery of sparse vectors from linear measurements. We proved some fundamental results that guarantees successful recovery. In Chapter 3 and 4, we saw that the classical theory does not perform well when applied to inverse problems in MRI. Due to computational and physical limitations, we had to modify the theory, rather than changing the sampling operator. The solution was structured sparsity, where we assume a structure in the way the non-zero components are organised in the image. Further, Chapter 5 provided an overview of the principles of deep learning with neural networks, as an alternative to compressive sensing. In Chapter 6, we presented two algorithms for solving optimization problems in compressive sensing: FISTA and Chambolle–Pock's Primal-Dual algorithm. Once their correctness was established, we were able to show in Chapter 7 that when unrolled, they formed a structure similar to that of neural networks. Therefore, with minimal modifications of the original neural network definition, we formed a new class of neural networks not requiring any training for successful image recovery. Chapter 8 was devoted to numerical results, where we demonstrated that the supplied implementations provided a significant speedup. In addition, we performed the first worst-case stability test of these algorithms.

The stability tests of the algorithms we considered in Chapter 6 showed that they were rather unstable. Relatively low levels of noise caused quite dramatic artifacts in the output. Due to these rather surprising results, we also tested Chambolle–Pock with the square root LASSO optimization problem, and it turned out that it was stable.

More exhaustive testing is required for the first two algorithms, to investigate if we can find parameters that makes them stable. We have established that they are not stable in general, because we found cases where they greatly amplified the noise. An important takeaway is that the choice of algorithm and optimization problem is not arbitrary when it comes to stability, even though the problem we are solving is stable in theory.

Bibliography

[AH16] Adcock, B. and Hansen, A. C. “Generalized sampling and infinite-dimensional compressed sensing”. In: *Found. Comput. Math.* 16.5 (2016), pp. 1263–1323.

[AH19] Adcock, B. and Hansen, A. C. *Compressive imaging (Work in progress).* 2019.

[AL] Adcock, B. and Li, C. “Compressed sensing with local structure: Uniform recovery guarantees for the sparsity in levels class”. In: ().

[Ant+19] Antun, V., Renna, F., Poon, C., Adcock, B., and Hansen, A. C. “On instabilities of deep learning in image reconstruction - Does AI come at a cost?” In: *arXiv e-prints*, arXiv:1902.05300 (Feb. 2019), arXiv:1902.05300. arXiv: 1902.05300 [cs.CV].

[BT09] Beck, A. and Teboulle, M. “A fast iterative shrinkage-thresholding algorithm for linear inverse problems”. In: *SIAM journal on imaging sciences* 2.1 (2009), pp. 183–202.

[Chu92] Chui, C. K. *An introduction to wavelets.* Elsevier, 1992.

[CRT06] Candes, E. J., Romberg, J. K., and Tao, T. In: *Communications on Pure and Applied Mathematics* 59.8 (2006), pp. 1207–1223. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpa.20124.

[Cur+13] Currie, S., Hoggard, N., Craven, I. J., Hadjivassiliou, M., and Wilkinson, I. D. “Understanding MRI: basic MR physics for physicians”. In: *Postgraduate medical journal* 89.1050 (2013), pp. 209–223.

[Cyb89] Cybenko, G. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[Don06] Donoho, D. L. “Compressed sensing”. In: *IEEE Transactions on Information Theory* 52.4 (Apr. 2006), pp. 1289–1306.

[FR13] Foucart, S. and Rauhut, H. *A mathematical introduction to compressive sensing.* Vol. 1. 3. Birkhäuser Basel, 2013.

[Goo+16] Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. *Deep learning.* Vol. 1. MIT press Cambridge, 2016.

[Hua+18] Huang, Y. et al. “Some investigations on robustness of deep learning in limited angle tomography”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention.* Springer. 2018, pp. 145–153.

[Lus+08] Lustig, M., Donoho, D. L., Santos, J. M., and Pauly, J. M. “Compressed sensing MRI”. In: *IEEE signal processing magazine* 25.2 (2008), p. 72.

[Mal09] Mallat, S. “A wavelet tour of signal processing”. In: *The Sparse Way* (2009).

[MFF16] Moosavi-Dezfooli, S.-M., Fawzi, A., and Frossard, P. “DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* June 2016.

[Pin99] Pinkus, A. “Approximation theory of the MLP model in neural networks”. In: *Acta numerica* 8 (1999), pp. 143–195.

[Py3] Rossum et al., G. van. *Python: A dynamic, open source programming language.* Available at https://www.python.org.

[RHA14] Roman, B., Hansen, A., and Adcock, B. “On asymptotic structure in compressed sensing”. In: *arXiv preprint arXiv:1406.4178* (2014).

[SCE01] Skodras, A., Christopoulos, C., and Ebrahimi, T. “The jpeg 2000 still image compression standard”. In: *IEEE Signal processing magazine* 18.5 (2001), pp. 36–58.

[Sch+18] Schlemper, J., Caballero, J., Hajnal, J. V., Price, A. N., and Rueckert, D. “A Deep Cascade of Convolutional Neural Networks for Dynamic MR Image Reconstruction”. In: *IEEE Transactions on Medical Imaging* 37.2 (Feb. 2018), pp. 491–503.

[SciPy] Jones, E., Oliphant, T., Peterson, P., et al. *SciPy: Open source scientific tools for Python.* [Online; accessed 2019-04-23]. 2001–.

[Sze+13] Szegedy, C. et al. “Intriguing properties of neural networks”. In: *CoRR* abs/1312.6199 (2013). arXiv: 1312.6199.

[TF] Abadi, M. et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* Available at https://www.tensorflow.org.

[Zhu+18] Zhu, B., Liu, J. Z., Cauley, S. F., Rosen, B. R., and Rosen, M. S. “Image reconstruction by domain-transform manifold learning”. In: *Nature* 555.7697 (2018), p. 487.