École polytechnique fédérale de Lausanne

Biomedical Imaging Group

Higher-Order Regularization Methods for Supervised Learning

Author:
Joaquim Campos

Supervisor:
Shayan Aziznejad

Professor:
Prof. Michael Unser

Master Thesis Report

MSc in Communication Systems

January 17, 2020

Abstract

In the past few decades, the amount of available data has been growing exponentially - a fact which is symbolized by the appearance of the "Big Data" industry [1]. Devices ranging from biosensors to smartphones turn almost every aspect of our lives into digital information, which is now continuosly shared over internet and easily stored. Moreover, processing power continues to follow the famous "Moore's law", reliably doubling every $\sim$ 2 years [1]. This combination of vast amounts of data with computing power makes the study of algorithms which can effectively and reliably extract information from data as relevant as ever.

To answer the question of how to learn from data is precisely the goal of Machine Learning and, specifically, a class of algorithms known as Supervised Learning [2]. In the parallel world of Signal Processing, the connection between signals in continuous and discrete domain has been well studied [3]. The Nyquist-Shannon sampling theorem is an example of this, which establishes that all the information of a limited bandwidth signal can be captured with enough samples. More recently, the field of compressed sensing has been analysing under-determined systems, with the goal of acquiring and reconstructing signals when very few samples are available (below the Nyquist-Shannon theorem requirements). These algorithms exploit the sparsity of the signal in some domain to provide higher-quality results [4]. Because sparsity introduces simpler and more interpretable solutions, it is a desirable feature for models [5].

This two-part thesis develops higher-order ($f : \mathbb{R}^N \mapsto \mathbb{R},\ N > 1$) regularization methods for supervised learning, while exploring model sparsity and the use of splines in merging the continuous and discrete worlds [6].

In the first part (deep splines), based on the work of Unser *et al.*[7], Gupta *et al.*[8] and Debarre *et al.*[9], we learn the 1D activation functions $\sigma : \mathbb{R} \mapsto \mathbb{R}$ of a neural network, together with the rest of the parameters. The deep spline module will be evaluated in an area classification problem and a model sparsification method will be introduced.

In the second part, we develop a novel 2D learning framework ($f : \mathbb{R}^2 \mapsto \mathbb{R}$), using a Hessian-Schatten regularization term. Unlike in Lefkimmiatis *et al.*[10], the problem will be treated in its continuous formulation and the advantages of learning over function spaces will be discussed. Finally, this framework will be applied to the tasks of reconstruction, data fitting and "2D super-resolution".

[1] `https://pages.experts-exchange.com/processing-power-compared`

Acknowledgements

I would like to thank my close family, Filipa, Luis and Pedro, for their love and for keeping me grounded; my friends, who shared this journey with me; Mariana, for her support and the lessons she taught me; Shayan, for his valuable guidance in this project; and Professor Unser, for receiving me in his lab.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter introduces important concepts used throughout the thesis. We begin with an overview of supervised learning, in the first section; in the second section, we examine classical methods to solve parameterized learning problems; in the following two sections, we introduce the topic of learning over function spaces, where we discuss learning over reproducing kernel Hilbert Spaces (RKHS) and generalized Total Variation regularization (gTV), which is closely linked to the content of this thesis; finally, we focus on the more recent field of Deep Learning, which again tackles problems with a parametric formulation, and review the theory on deep splines, a method to learn the activations of a neural network in which the first part of the project is based.

1.1 Supervised Learning

In mathematical terms, we can describe the setup of supervised learning as follows. We have a dataset consisting of a set S_X of inputs and a corresponding set S_Y of labels/observations. Each pair in the dataset is assumed to be independently sampled from an underlying distribution $\mathcal{D}$ with range $\mathcal{X} \times \mathcal{Y}$ [11]:

$$\begin{aligned} \text{data}: \ S_X &= \{\boldsymbol{x}_1, ..., \boldsymbol{x}_M\} \\ \text{observations}: \ S_Y &= \{\boldsymbol{y}_1, ..., \boldsymbol{y}_M\} \end{aligned}$$

$$S = \{(\boldsymbol{x}_m, \boldsymbol{y}_m) \text{ i.i.d.} \sim \mathcal{D}\}_{m=1}^{M}. \tag{1.1}$$

As an example, the dataset might consist of several images of cats and dogs, together with their corresponding binary labels - 0 when the image represents a cat and 1 when it represents a dog - and the goal of the supervised learning algorithm (denoted $\boldsymbol{\mathcal{A}}$) is to learn to distinguish between images of these two types. This is known as a classification problem since the range of the output consists of a finite set of discrete values. Each image in the set S_X is sampled according to some distribution of natural images of cats and dogs (*i.e.*, $X_m \sim p_X, \ m = 1, \ldots, M$) and the corresponding label is determined from the image sample:

$$\mathbf{P}(Y_m = 0 \mid X_m = \boldsymbol{x}_m) = \begin{cases} 1, & \text{if } \boldsymbol{x}_m \text{ is an image of a cat.} \\ 0, & \text{if } \boldsymbol{x}_m \text{ is an image of a dog.} \end{cases}$$

$$\mathbf{P}(Y_m = 1 \mid X_m = \boldsymbol{x}_m) = 1 - \mathbf{P}(Y_m = 0 \mid X_m = \boldsymbol{x}_m)$$

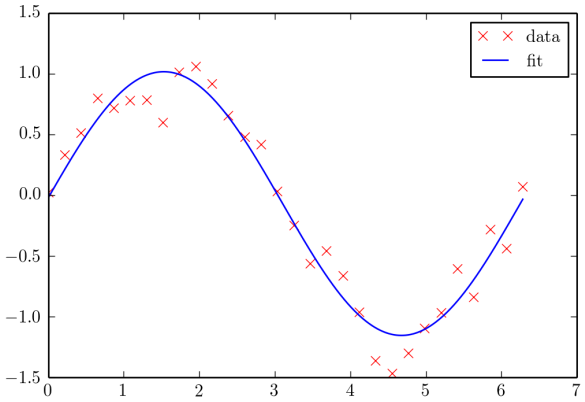


Figure 1.1: Example of data fitting;
blue line is the underlying function f;
red crosses are the noisy observations. [1]

Our objective is to construct a model (denoted f_S) which can classify correctly, as often as possible, images sampled from the distribution p_X.

In another example, a function $f : \mathbb{R}^M \mapsto \mathbb{R}$ is applied to each input $\boldsymbol{x}$, producing a corresponding real-valued output y, but whose observations are noisy due to noise present in the measurement system:

$$y_m = f(\boldsymbol{x}_m) + \epsilon_m, \quad m = 1, ..., M. \tag{1.2}$$

where ϵ is the noise. We can assume the noise is independent and uniformly distributed (i.i.d.) and follows a gaussian distribution. Then, the random variable $Y_m \mid X_m$ follows a normal distribution with mean $f(\boldsymbol{x_m})$ and standard deviation σ:

$$f_{Y_m|X_m}(y_m \mid \boldsymbol{x_m}) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(y_m - f(\boldsymbol{x}_m))^2/2\sigma^2} \tag{1.3}$$

In this case, we would like to have a model f_S which is close to the underlying f, in some sense (*e.g.*, minimize their L_2 difference). Figure 1.1 shows this data fitting task for the 1D case $f : \mathbb{R} \mapsto \mathbb{R}$. This is known as a regression problem since the output can assume values in a continous domain.

In more general terms, ideally, we would like to find a family of functions $\mathcal{F}_T$ for which the expected loss over the true distribution is sufficiently low:

$$L_{\mathcal{D}}(f) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}[\ell(\boldsymbol{y}, f(\boldsymbol{x}))] \tag{1.4}$$

$$\mathcal{F}_{T,\epsilon} = \{f : L_{\mathcal{D}}(f) < \epsilon\} \tag{1.5}$$

where ℓ, the loss function, quantifies how good the function f is for the task. We call $\mathcal{F}_{T,\epsilon}$ the target family for requirement ϵ. Notice that, if the requirement ϵ is very strict, the target family might be an empty set. For example, in problem (1.2), the expected loss is lower bounded by the variance of the noise σ^2, which we could achieve only if we model perfectly the underlying function f (*i.e.*, $\mathcal{F}_{T,\epsilon} = \emptyset$, for $\epsilon < \sigma^2$).

Note, however, that our model is the result of some algorithm $\boldsymbol{\mathcal{A}}$ and is constrained by its architecture and parameters. Hence, it is likely that our model

[1] `https://i.stack.imgur.com/0e8Fp.png`

family (denoted $\mathcal{F}_S$) does not contain some of the functions in the target family $\mathcal{F}_{T,\epsilon}$ (or any, depending on the ϵ constraint), *i.e.*, $f \in \mathcal{F}_{T,\epsilon}$ may be not realizable. In addition, we cannot minimize the true loss (1.4) in practice, since we do not know the underlying distribution $\mathcal{D}$ of the data. Instead, we only have access to a sampled dataset S on which we can minimize the so-called empirical (training) loss, and which we hope is a good enough approximation of the the true loss (1.4):

$$L_S(f_S) = \frac{1}{|S|} \sum_{(\boldsymbol{x}_m, \boldsymbol{y}_m) \in S} \ell(\boldsymbol{y}_m, f_S(\boldsymbol{x}_m)) \tag{1.6}$$

To emphasize that our model $f_S \in \mathcal{F}_S$ has parameters $\boldsymbol{\theta}$ and is a result of a supervised learning algorithm $\boldsymbol{\mathcal{A}}$, which learns on the sampled training dataset S, we write:

$$f_{S,\boldsymbol{\theta}} = \boldsymbol{\mathcal{A}}(S) \tag{1.7}$$

1.1.1 Regularization

Minimizing the empirical loss (1.6) instead of the true loss (1.4) comes with its downsides. First, due to the stochastic nature of sampling a dataset, even if the empirical (training) loss $L_S(f_S)$ is small enough, when the learned model f_S is applied to another freshly sampled dataset S_{test} from the same distribution $\mathcal{D}$, the test loss $L_{S_{\text{test}}}(f_S)$ might be much higher. Second, apart from the generalization problems inherent to training on a sampled dataset, making the minimization of the empirical loss our sole objective is counter-productive when the samples are noisy, since our goal is to model the underlying target function and not to fit the noise present in the observations.

The problem described in both of these points is called overfitting, named as such because it depicts the unfortunate situation of *over-fitting* to the training set and not modeling the underlying function. It is ilustrated in figure 1.2. The solution for this is to have models which can sufficiently minimize the empirical loss, but which are kept simple enough so as to not overfit the training data and to generalize well to other datasets. This can be done with regularization, which constrains the capacity of the family of model functions; one example is ridge regularization, where a term $\lambda \|\boldsymbol{\theta}\|^2$, which penalizes large parameters, is added to the minimization objective (1.6) (*lambda* is the regularization weight).

Lastly, there might be several models which give low empirical losses (we say that we have an under-determined system). Therefore, in line with the framework of compressed sensing, we might use regularization to incorporate prior knowledge that we have on the target functions to restrict the model search space. For example, if we know that we are trying to model natural images, our model can reflect their smoothness by penalizing large variations of the function. Likewise, if we are measuring the noisy sound signal of a tuning fork, we know that the underlying signal is sinusoial. Since the Fourier transform of a sinusoid consists of only two diracs whose location depends on its frequency, *i.e.*, $\mathcal{F}\{cos(2\pi f_0 t)\} = \frac{1}{2}(\delta(f - f_0) - \delta(f + f_0))$, we expect sparsity in fourier domain.

In this thesis, sparsity-promoting regularization plays a central role. It relates to the more general paradigm in signal processing of using as less parameters as

[2] `https://commons.wikimedia.org/wiki/File:Overfitting.svg`

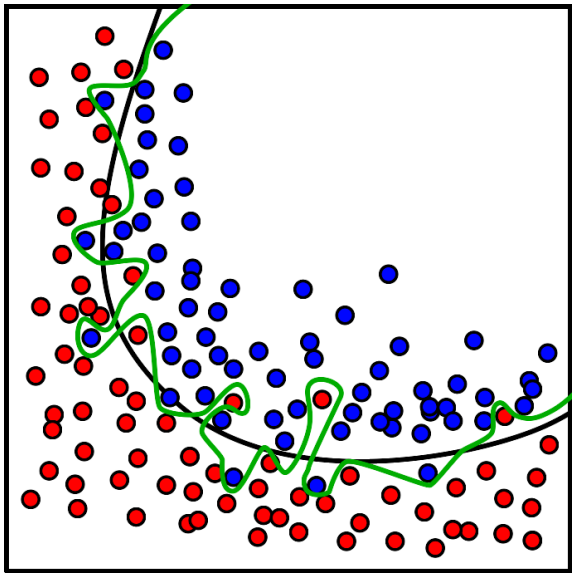


Figure 1.2: Example of overfitting;
black line is the underlying function f;
green line is the overfitted model. [2]

possible to explain the data, so as to have simpler and more interpretable solutions [5].

The next points summarize the usefulness of regularization:

- Reducing overfitting - regularization constrains the model so that it is less likely to fit the noise inherent to sampling or explicitly present in the observations and more likely to model underlying distribution.
- Solving under-determined systems - when several functions would satisfy the approximation constraints, regularization reduces the search space by enforcing prior knowledge when we have information about the underlying target function f.

1.2 Classical Methods

In this section, we introduce classical supervised learning methods; the first subsection shows how regularization can deal with under-determined or ill-conditioned systems and introduces the family of gradient-based optimization methods; the second subsection shows how regularization can be used to exploit model sparsity. To simplify notation, from now on, we drop the S in f_S when referring to the model function.

1.2.1 Linear Ridge Regression

Linear Ridge regression consists of a linear regression problem with an ℓ_2 regularization on the parameters. It is mathematically expressed as:

$$\min_{\boldsymbol{w}} \frac{1}{2M} \sum_{m=1}^{M} (f(\boldsymbol{x}_m) - y_m)^2 + \lambda \|\boldsymbol{w}\|_2^2 \tag{1.8}$$

where $\boldsymbol{x_m} \in \mathbb{R}^D$, $f : \mathbb{R}^D \mapsto \mathbb{R}$, and $f(\boldsymbol{x}_m) = \boldsymbol{x}_m^T \boldsymbol{w}$, *i.e.*, the search space is restricted to functions which are a linear combination of the input $\boldsymbol{x}_m$ with learnable weights (parameters) $\boldsymbol{w}$. The first term is the data fidelity term (how well can the model explain the data) and the second one is the regularization term. The previous equation can be written in matrix form as:

$$\min_{\boldsymbol{w}} \frac{1}{2M} \|\mathbf{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \lambda \|\boldsymbol{w}\|_2^2 \tag{1.9}$$

where $\mathbf{X} \in \mathbb{R}^{M \times D}$ is the matrix whose row i is the vector $\boldsymbol{x}_i$. It is easy to see that the expression above is convex in $\boldsymbol{w}$, so it has a single global minimizer and we can find it explicitly by equating the gradient wrt to the parameters $\boldsymbol{w}$ to zero:

$$\nabla \mathcal{L}(\boldsymbol{w}^*) = -\frac{1}{M}\mathbf{X}^T(\boldsymbol{y} - \mathbf{X}\boldsymbol{w}^*) + 2\lambda \boldsymbol{w}^* = 0 \Leftrightarrow \tag{1.10}$$

$$\Leftrightarrow \boldsymbol{w}^* = (\mathbf{X}^T\mathbf{X} + \lambda'\mathbf{I})^{-1}\mathbf{X}^T\boldsymbol{y} \tag{1.11}$$

where $\frac{\lambda'}{2M} = \lambda$.

With (1.11), we can analyse the effect of the regularization on the result. First, note that the eigenvalues of $\mathbf{X}^T\mathbf{X} + \lambda'\mathbf{I}$ are at least λ' (> 0). So, the regularization essentially "lifts" the eigenvalues of the system, making $\mathbf{X}^T\mathbf{X} + \lambda'\mathbf{I}$ always invertible, which leads to a unique $\boldsymbol{w}^*$ solution (since a matrix is invertible iff none of its eigenvalues is zero). On the other hand, even if the matrix is invertible, it can be ill-conditioned if some of its columns are nearly collinear. This situation can lead to numerical issues when inverting the matrix and is quantified by the condition number, which for a symmetric matrix $\mathbf{A}$ (like in our case) is equal to:

$$\kappa(\mathbf{A}) = \frac{|\lambda_{\max}|}{|\lambda_{\min}|} \geq \frac{|\lambda_{\max}| + \lambda'}{|\lambda_{\min}| + \lambda'} \tag{1.12}$$

We can verify that, by "lifting" the eigenvalues, regularization can reduce the condition number, thus improving numerically stability.

Notice that the complexity of (1.11) is $\mathcal{O}(D^2M)$ and the computations are not easily paralellized. Because of this, a class of faster methods which do not involve matrix inversions and are based on the gradient (1.10) (gradient-based methods) have gained more traction in supervised learning.

The full gradient descent algorithm takes a step in the opposite direction of the gradient to minimize the loss:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \gamma \nabla \mathcal{L}(\boldsymbol{w}^{(t)}) \tag{1.13}$$

where γ is the step size. This is essentially a local approach to a global problem: if the function to optimize is convex, using an appropriate step size, we are guaranteed to reach the optimal solution; if not, we can still reach a local minima which is close to the optimal one, but there are no guarantees. In practice, the full gradient descent algorithm takes a lot of time to compute and leads to a small number of updates in a certain time frame. For this reason, batch gradient descent (BGD) updates the weights after the gradient of the loss is calculated for a batch of b samples (*e.g.*, 100). The gradient for a batch B_i is the average of the gradients of its $b_i = |B_i|$ randomized samples taken from the training set:

$$\nabla \mathcal{L}_{B_i}(\boldsymbol{w}^{(t)}) = \frac{1}{|B_i|} \sum_{n \in B_i} \nabla \mathcal{L}_n(\boldsymbol{w}^{(t)}) \tag{1.14}$$

An important point is that the batch gradient is an unbiased estimator of the full gradient, *i.e.*, $\mathbb{E}[\nabla \mathcal{L}_B(\boldsymbol{w}^{(t)}] = \nabla \mathcal{L}(\boldsymbol{w}^{(t)})$. Nowadays, batch gradient descent is the core of deep learning methods, since the computation of the gradients for each batch can be paralellized easily in a GPU.

1.2.2 Linear LASSO Regression

Linear LASSO regression is a linear regression problem where the ℓ_2 norm regularization in (1.8) is replaced by an ℓ_1 norm regularization [12]:

$$\min_{\boldsymbol{w}} \frac{1}{2M} \sum_{m=1}^{M} (f(\boldsymbol{x}_m) - y_m)^2 + \lambda \left\| \boldsymbol{w} \right\|_1 \tag{1.15}$$

This is very well studied problem in signal processing, especially in the field of compressed sensing. Since the ℓ_1 regularization is known to promote sparse solutions [12][13], it can be used to reduce the search space and solve under-determined systems. For example, in magnetic ressonance imaging (MRI), acquisitions are known to be sparse in a given transform domain [14], so this knowledge can be incroporated in the loss function with an ℓ_1 regularization.

1.2.3 Classification Algorithms

Unlike in regression problems, in classification tasks the output assumes values in a finite set. Each possible value $y \in \{0, 1, 2, \ldots, K-1\}$ represents a class. In the cats/dogs example, $K = 2$, and the label 0 is used for cats and 1 for dogs (binary classification). We are going to present a high-level overview of two classical classification algorithms: Logistic regression and Support-vector machines.

Logistic Regression

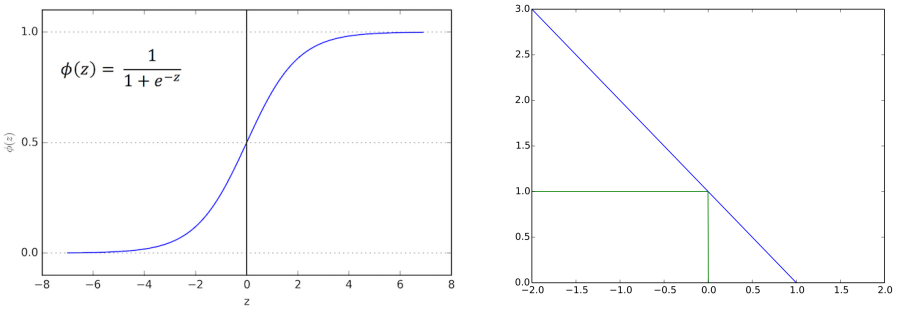
Logistic regression is based on the logistic/softmax function (figure 1.3a):

$$\sigma(z) := \frac{e^z}{1 + e^z}$$

which is used to transform $(-\infty, \infty)$ predictions from a linear model $\boldsymbol{x}^T \boldsymbol{w} + w_0$ into a probability. Since it predicts a real probability value, this method contains "regression" in its name, despite being used for classification. We refer to the simpler case of binary classification (*e.g.*, cats/dogs), although it can be extended to multi-class learning:

$$\begin{aligned} p(1 \mid \boldsymbol{x}, \boldsymbol{w}) &= \sigma(\boldsymbol{x}^T \boldsymbol{w} + w_0) \\ p(0 \mid \boldsymbol{x}, \boldsymbol{w}) &= 1 - \sigma(\boldsymbol{x}^T \boldsymbol{w} + w_0) \end{aligned} \tag{1.16}$$

The goal is to choose the parameters $\boldsymbol{w}$ which explain the data (1.1), *i.e.*, maximize the likelihood $p(\boldsymbol{y} \mid \mathbf{X}, \boldsymbol{w}) = \prod_{m=1}^{M} p(y_m \mid \boldsymbol{x}_m)$ (assuming i.i.d. samples). Instead,



(a) Logistic loss. (b) Hinge loss.

Figure 1.3: Classification losses.

we can minimize the additive inverse of the log-likelihood, transforming it into a loss function. Deriving the formulas (see [2]), we get:

$$\mathcal{L}(\boldsymbol{w}) = \sum_{m=1}^{M} ln(1 + e^{\boldsymbol{x}_m^T \boldsymbol{w}}) - y_m \boldsymbol{x}_m^T \boldsymbol{w} \tag{1.17}$$

This cost function can be proven to be convex and so it can be optimized using the batch gradient descent method described in the previous section. An ℓ_2 regularization with weight $\frac{\lambda}{2}$ is often also added to the the loss function [11].

After training the model, for a new sampled dataset (test set), we can simply apply it and predict the label 1 if $\sigma(\boldsymbol{x}^T \boldsymbol{w} + w_0) > 0.5$ and 0 otherwise.

Support-Vector Machines

Consider again the case of binary classification but where now the two classes are represented by -1 and 1. SVM optimizes the following loss, containing an ℓ_2 regularization term:

$$\min_{\boldsymbol{w}} \sum_{m=1}^{M} [1 - y_m \boldsymbol{x}_m^T \boldsymbol{w}]_+ + \frac{\lambda}{2} \|\boldsymbol{w}\|_2^2 \tag{1.18}$$

The first term is called the Hinge loss, $[z]_+ := \max\{0, z\}$ (figure 1.3b). The essential characteristic of this loss function is that it does not penalize datapoints which are already "well" classified: if the label is $y_m = 1$ and the prediction $\boldsymbol{x}_m^T \boldsymbol{w} \geq 1$ then the data point $\boldsymbol{x}_m$ does not contribute to the loss; and similarly if $y_m = -1$ and $\boldsymbol{x}_m^T \boldsymbol{w} \leq -1$. The vectors $\boldsymbol{x}_m$ for which this happens are called non-support vectors (hence the name of the method). The loss (1.18) can be again optimized with a gradient descent algorithm. However, duality theory is often used to derive the following dual problem formulation:

$$\max_{\boldsymbol{\alpha} \in [0,1]^N} \boldsymbol{\alpha}^T \mathbf{1} - \frac{1}{2\lambda} \boldsymbol{\alpha}^T \mathbf{Q} \boldsymbol{\alpha}, \tag{1.19}$$

where $\mathbf{Q} := \text{diag}(\boldsymbol{y}) \mathbf{X} \mathbf{X}^T \text{diag}(\boldsymbol{y})$, and $\boldsymbol{w}$ relates to $\boldsymbol{\alpha}$ through the relation $\boldsymbol{w}(\boldsymbol{\alpha}) = \frac{1}{\lambda} \mathbf{X}^T \text{diag}(\boldsymbol{y}) \boldsymbol{\alpha}$. This problem can be solved by coordinate ascent on $\boldsymbol{\alpha}$ and has some advantages: the dual is kernelized,

$$\mathbf{K} = \mathbf{X} \mathbf{X}^T \tag{1.20}$$

so we can use the kernel trick by specifying $\mathbf{K}$ directly without the need to specify $\mathbf{X}$ that originated it; and the solution $\boldsymbol{\alpha}$ is only non-zero in positions corresponding to vectors which are essential in determining the decision boundary (essential support vectors).

1.3 Learning Over Function Spaces

In contrast to the classical methods we have seen so far, which solve minimization problems formulated in terms of the parameters $\boldsymbol{w}$ of the model f, another class of methods exists which tackle problems expressed directly in terms of f and not its parameters (learning over function spaces). This shift in perspective will be important in the rest of the thesis.

In this section, we briefly address the role of reproducing kernel Hilbert Spaces (RKHS) in Machine Learning [15], and then discuss generalized Total Variation regularization (gTV), which promotes sparsity in continuous domain. In particular, we will examine an algorithm based on B-splines which exactly discretizes the continuous problem formulation.

1.3.1 Learning Over Reproducing Kernel Hilbert Spaces

Suppose H is a Hilbert space on a set X. H is a reproducing kernel Hilbert space if, for each $x \in X$, there exists a unique function $k_x \in H$ such that [16]:

1. k has the reproducing property, $f(x) = \langle f, k_x \rangle, \ \forall f \in H$.
2. k spans H, $H = \overline{\text{span}\{k_x(\cdot) : x \in X\}}$.

From this, we can define the reproducing kernel of H, $k : X \times X \mapsto \mathbb{R}$, as:

$$k(x, y) = \langle k_x, k_y \rangle \ . \tag{1.21}$$

Scholkopf *et al.* proved a general version of the following Representer Theorem, which created a more unified framework for learning.

Suppose we have a kernel $k : X \times X \mapsto \mathbb{R}$, and which has a corresponding RKHS H_k. Consider a problem of the form:

$$\arg\min_{f \in H_k} \sum_{m=1}^{M} (f(\boldsymbol{x_m}) - y_m)^2 + \lambda g(\|f\|) \tag{1.22}$$

where $\|\cdot\|$ is the Hilbert space norm and g is a strictly monotonically increasing function, *e.g.*, $g(\|f\|) = \lambda \|f\|^2$. Then, the minimizer f^* is of the form:

$$f^*(\boldsymbol{x}) = \sum_{i=1}^{M} \alpha_i k(\boldsymbol{x}, \boldsymbol{x}_i) \tag{1.23}$$

where $\alpha_i \in \mathbb{R}$. Note how the problem is formulated in terms of f and not its parameters. Several algorithms can be seen in the light of this theorem. For example, in SVM (1.20), we have a kernel of the form $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^T \boldsymbol{x}_j$.

In the next section, we introduce a more specific problem which will be solved by spanning the space of model functions with a B-spline basis dictionary, motivated by a corresponding Representer Theorem.

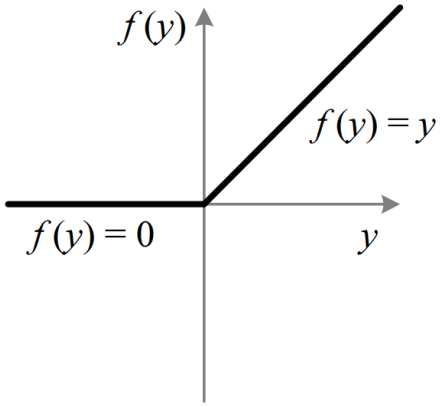


Figure 1.4: Canonical Green's function for the D^2 operator (ReLU). (source: [22])

1.3.2 Generalized Total Variation

The problem introduced in this section deals with important concepts which will be used in the rest of the thesis; it is of the following form:

$$\arg\min_{f} \sum_{m=1}^{M} (f(x_m) - y_m)^2 + \lambda \left\| \mathrm{L}\{f\} \right\|_{\mathcal{M}} \tag{1.24}$$

where $f : \mathbb{R} \mapsto \mathbb{R}$, L is a continuous linear shift-invariant (LSI) operator and the $\mathcal{M}$ norm is associated with $\mathcal{M}(\mathbb{R})$, the space of Radon measures (1.26). Notice here that the regularization is applied to the function itself, like in the RKHS case.

Regularization

Let us now explain the meaning of this regularization term. L is a spline-admissible operator, which is defined as an LSI operator which satisfies the following properties:

1. There is a locally integrable function of slow growth $\rho_{\mathrm{L}} : \mathbb{R} \mapsto \mathbb{R}$ s.t. $\mathrm{L}\{\rho_{\mathrm{L}}\} = \delta$ (L admits a Green's function).
2. Its null space $\mathcal{N}_{\mathrm{L}}$ has finite dimension.

We will only focus on linear differential operators L (which admit a Green's function and have a finite null-space). One important example is $\mathrm{L} = \mathrm{D}^2$, in which case the notation $\mathrm{TV}^{(2)}$ (total variation-2) is used for the regularization term; the nullspace of this operator consists of all linear functions, *i.e.*, $\mathcal{N}_{\mathrm{D}^2} = \mathrm{span}\{1, x\}$, and its corresponding Green's functions (non-unique) can be written as:

$$\rho_{\mathrm{D}^2} = a[x]_+ \; + \; u, \quad u \in \mathcal{N}_{\mathrm{D}^2}. \tag{1.25}$$

The first term $a[x]_+$ is a ReLU function with slope a; the fact that there are several Green's functions for D^2 and that they differ by a linear term arises from the nullspace $\mathcal{N}_{\mathrm{D}^2}$. We call $\rho_{\mathrm{D}^2} = [x]_+$ the canonical Green's function (a.k.a. ReLU), which is shown in figure 1.4.

The regularization term then consists of the $\mathcal{M}$ norm of the operator applied to function. The $\mathcal{M}(\mathbb{R})$ space of radon measures, associated with the norm $\|\cdot\|_{\mathcal{M}}$, is

defined as:

$$\mathcal{M}(\mathbb{R}) = \left\{ f \in \mathcal{S}'(\mathbb{R}) : \|f\|_{\mathcal{M}} \triangleq \sup_{\rho \in \mathcal{S}(\mathbb{R}) : \|\rho\|_\infty \leq 1} \langle f, \rho \rangle < \infty \right\} \tag{1.26}$$

This formula is shown here for the sake of completeness (for more details, see [7]). For the purposes of this thesis, it suffices to mention that $\mathcal{M}(\mathbb{R})$ is a slightly larger space than $L_1(\mathbb{R})$, since it also includes the dirac delta function $\|\delta(\cdot - x_m)\|_{\mathcal{M}} = 1$, while $\delta(\cdot - x_m) \notin L_1(\mathbb{R})$. Moreover, the associated norms have a strong connection in the sense that functions with a finite L_1 norm have the same $\mathcal{M}$ norm, *i.e.*, $\|f\|_{L_1} = \|f\|_{\mathcal{M}}$ for any $f \in L_1(\mathbb{R})$. This property hints that the $\mathcal{M}$ norm can also be used to enforce sparsity, as will become clearer soon.

Now, we are in a position to understand the effect of the regularization term $\|\mathrm{L}\{f\}\|_{\mathcal{M}}$: it promotes sparsity of the function in the range of the L operator. Take again the example of the $\mathrm{TV}^{(2)}$, $\mathrm{L} = \mathrm{D}^2$; in this case, the regularization term promotes sparsity of the second-derivative of the function, which will favor piece-wise linear solutions. Notice that a ReLU (canonical Green's function of D^2) is sparse in this sense, since $\mathrm{TV}^{(2)}\{\mathrm{ReLU}\} = 1$, because $\|\delta(x)\|_{\mathcal{M}} = 1$.

Search Space

The continuous domain Representer Theorem in [9] (*Thm. 1*) states that the extreme points of the solution set S of problem (1.24) are a sum of shifted Green's functions of the operator L, with a bound on K:

$$f(x) = \sum_{k=1}^{K} a_k \rho_{\mathrm{L}}(x - x_k) + \sum_{n=1}^{N_0} b_n p_n(x) \tag{1.27}$$

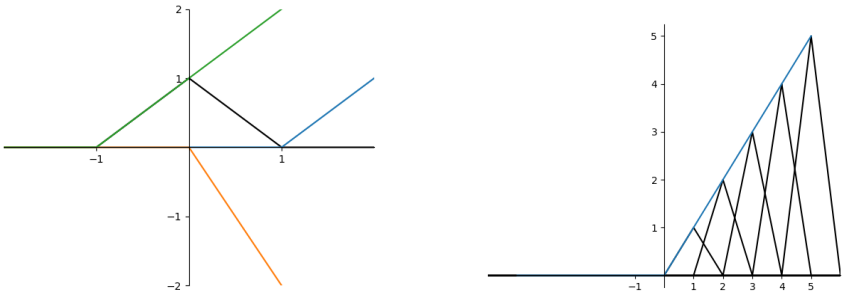
where $a_k, x_k \in \mathbb{R}$, and $\{p_n\}_{n=1,\ldots,N_0}$ form a basis of $\mathcal{N}_L$. The dictionary basis $\rho_L(\cdot - \tau)_{\tau \in \mathbb{R}}$ only depends on the operator L and not on the measurements.

Therefore, we can reduce our search space to functions of this form (1.27). This has the advantage of parameterizing the learning problem, given that these functions are entirely defined by the location of the knots $\{x_k\}_{k=1}^K$, the Green's functions coefficients $\{a_k\}_{k=1}^K$, and the null space basis coefficients $\{b_n\}_{n=1}^N$, with a bound on K.

To make the task of finding a solution more practical, two ideas have been proposed. The first one, in [8], is the elimination of knot discovery by restricting the knots of the Green's functions to a grid with spacing h. This makes the problem computationally more treatable because discovering the knots $\{x_k\}_{k=1}^K$ is an expensive process. Moreover, as the grid becomes finer, we are able to represent any function. With the addition of this (infinite) grid, (1.27) becomes:

$$f(x) = \sum_{k \in \mathbb{Z}} a_k \rho_{\mathrm{L}}(x - kh) + \sum_{n=1}^{N_0} b_n p_n(x) \tag{1.28}$$

The second contribution, in [9], is the realization that functions of the form (1.28) have an alternative formulation in terms of a B-spline dictionary basis. Here, we will focus on the D^2 operator (for a general statement, see [9] *proposition 1*). In fact, (1.28) is a piece-wise linear function, which is also spanned by a B1-spline



(a) Constructing a B1-spline basis with ReLUs.

(b) Constructing a ReLU with B1-splines.

Figure 1.5: B1-splines/ReLU formulation equivalence.

dictionary basis. An intuitive way to see why this is the case is shown in figure 1.5 - we can represent a ReLU with B1-splines and vice-versa. Notice that a linear term can also be spanned by a B1-spline dictionary. This alternative formulation increases numerical stability, since small perturbations in the coefficients lead to small perturbations in the measurements; in the case of the Green's function formulation, the system is poorly-conditioned: a perturbation of the coefficient of a ReLU with knot at x_0 propagates to all measurements at $x > x_0$, and the greater $x - x_0$, the greater the change in measurements.

With these new basis functions, (1.28) becomes:

$$f(x) = \sum_{k \in \mathbb{Z}} c_k \beta^1(x - kh) \tag{1.29}$$

Note that this basis is interpolatory, which means that $c_k = \beta^1(kh)$. We can derive a useful relationship between a_k and c_k; assuming a grid spacing $h = 1$, for simplicity, we start from the definition of the canonical Green's function for the D^2 operator, $\rho(x) = [x]_+$:

$$D^2 \rho = \delta \xrightarrow{\mathcal{F}} \hat{\rho} = \frac{1}{\omega^2} \tag{1.30}$$

Using now the B1-splines Fourier definition [6]:

$$\hat{\beta}^n(\omega) = \left(\frac{sin(\omega/2)}{(\omega/2)} \right)^{n+1} = \frac{\left(e^{j\omega/2} - e^{-j\omega/2}\right)^{n+1}}{(j\omega)^{n+1}} \tag{1.31}$$

$$\hat{\beta}^1(\omega) \triangleq \hat{\Delta}(\omega) = \left(e^{j\omega} - 2 + e^{-j\omega}\right) \cdot \hat{\rho} \tag{1.32}$$

Transforming this expression into time domain, gives the result expressed in figure 1.5 (a):

$$\Delta(x) = [1, -2, 1] * \rho(x). \tag{1.33}$$

Replacing this result into the spline expansion above (1.29), we get

$$\begin{aligned}
\sum_{k\in\mathbb{Z}} a_k\rho(x-k) + b_0 + b_1 x &= \sum_{k\in\mathbb{Z}} c_k\Delta(x-k) \\
\Leftrightarrow \quad \sum_{k\in\mathbb{Z}} a_k\rho(x-k) &= \sum_{k\in\mathbb{Z}} c_k\Delta(x-k) - b_0 - b_1 x \\
\Leftrightarrow \quad \sum_{k\in\mathbb{Z}} a_k\rho(x-k) &= \sum_{k\in\mathbb{Z}} (c_k - b_0 - b_{1,k})\Delta(x-k) \\
&= \sum_{k\in\mathbb{Z}} d_k\Delta(x-k) \\
&= \sum_{k\in\mathbb{Z}} d_k\rho(x-k+1) - 2\sum_{k\in\mathbb{Z}} d_k\rho(x-k) + \sum_{k\in\mathbb{Z}} d_k\rho(x-k-1) \\
&= \sum_{k\in\mathbb{Z}} d_{k+1}\rho(x-k) - 2\sum_{k\in\mathbb{Z}} d_k\rho(x-k) + \sum_{k\in\mathbb{Z}} d_{k-1}\rho(x-k) \\
&= \sum_{k\in\mathbb{Z}} (d_{k+1} - 2d_k + d_{k-1})\rho(x-k). \qquad (1.34)
\end{aligned}$$

where $d_k = c_k - b_0 - b_{1,k}$. Now notice that $a_k = d_{k+1} - 2d_k + d_{k-1} = c_{k+1} - 2c_k + c_{k-1}$, because the linear term $b_0 + b_{1,k}$ lies in the nullspace of the second-order finite difference operator. Finally, we get the following relationship between the Green's function and B1-spline expansion coefficients:

$$\boldsymbol{a} = [1, -2, 1] * \boldsymbol{c} \qquad (1.35)$$

where $\boldsymbol{a}$ is the ordered list of a_k's and * is the convolution operation. In the general case, the filter coefficients are divided by the grid spacing h.

Exact Discretization

The advantage of this formulation is that it allows us to discretize **exactly** the problem (1.24). We continue to focus on $\mathrm{L} = \mathrm{D}^2$ (recall that $\left\|\mathrm{D}^2\{f\}\right\|_{\mathcal{M}} := \mathrm{TV}^{(2)}\{f\}$). Replacing (1.28) into the minimization objective (1.24), given that $\mathrm{D}^2\{\rho_{\mathrm{D}^2}\} = \delta$ and $\|\delta(\cdot - x_m)\|_{\mathcal{M}} = 1$, we get

$$\mathrm{TV}^{(2)}\{f\} = \sum_{k\in\mathbb{Z}} |a_k| = \|\boldsymbol{a}\|_1 = \left\| \frac{1}{h}[1, -2, 1] * \boldsymbol{c} \right\|_1 \qquad (1.36)$$

At the end, reducing the search space led to a simple formulation: the $\mathrm{TV}^{(2)}$ of any function in our solution family is simply the ℓ_1 norm of its Green's function coefficients. This result also makes a direct connection between the $\mathcal{M}$ norm and the ℓ_1 norm and makes it clear how the $\mathrm{TV}^{(2)}$ leads to sparsity of the Green's function coefficients, given the sparsifying effect of ℓ_1 regularizers. The exact discretization of the problem, after the search space restriction, becomes:

$$\arg\min_{\boldsymbol{c}} \sum_{m=1}^{M} (f(x_m, \boldsymbol{c}) - y_m)^2 + \lambda \left\| \frac{1}{h}[1, -2, 1] * \boldsymbol{c} \right\|_1 \qquad (1.37)$$

$$f(x, \boldsymbol{c}) = \sum_{k\in\mathbb{Z}} c_k \beta^1(x - kh)$$

Finite-Dimensional Problem

In practice, we can assume that the signal we are trying to model has a finite support $[-L, L]$, so we can have a finite-dimensional coefficient vector $\boldsymbol{c}$ whose coefficients are within this range. Then, by imposing linear boundary conditions, we can represent the linear term $b_0 + b_1 x$ and construct an unbounded domain function which is not penalized by the regularization outside $[-L, L]$. The formula (1.35) is still valid in this case. We will discuss this in more detail in section 1.5.

Finally, we have the following finite-dimensional discrete optimization problem:

$$\arg\min_{\boldsymbol{c}\in\mathbb{R}^N} \|\mathbf{Hc} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{Lc}\|_1 \tag{1.38}$$

This problem can be efficiently solved through an ADMM-simplex mixed algorithm [9][8]. More details on this will be provided in the second part of the thesis, where this algorithm is applied.

To summarize the important ideas of this section: we started from a continuous formulation in (1.24) for $f : \mathbb{R} \mapsto \mathbb{R}$; motivated by the continuous domain (1D) Representer Theorem, we reduced the search space to the family of functions spanned by a sum of shifted Green's functions, parameterizing the continuous-domain problem (1.24). We then restricted the search space by placing the knots of the Green's functions in a grid (eliminating knot discovery) and equivalently representing the functions in our model family with a B1-spline expansion. At the end, we were able to exactly discretize the continuous domain problem, transforming it into a discrete one of finding finitely-many coefficients $\boldsymbol{c}$.

This algorithm is elegant, has an interpretable result and is mathematically motivated, but it only allows learning of functions $f : \mathbb{R} \mapsto \mathbb{R}$. The rest of the thesis will explore higher-order methods $f : \mathbb{R}^N \mapsto \mathbb{R}$.

One of the machine learning objects which allows us to go to a higher-order is a neural network, the object of study of Deep Learning. Since they will be used in the rest of the thesis, they are briefly discussed in the next section. Notice that they fit in the parameteric framework, being therefore more closely linked to the algorithms we saw in the section 1.2.

1.4 Deep Learning

Deep Learning started gaining traction in 2014, after AlexNet [17] shook the computer vision community by outperforming all the classical computer vision algorithms by a large margin. It is now used for diverse tasks such as inverse problems in imaging [18], style transfer [19], face generation [20], amongst others. Mathematically, a neural network, the object of Deep Learning, is a sequence of linear and non-linear functions [7], i.e., L layers (see figure 1.6):

$$\boldsymbol{f}(\boldsymbol{x}) = (\boldsymbol{\sigma}_L \circ \boldsymbol{f}_L \circ \boldsymbol{\sigma}_{L-1} \circ \cdots \circ \boldsymbol{\sigma}_2 \circ \boldsymbol{f}_2 \circ \boldsymbol{\sigma}_1 \circ \boldsymbol{f}_1)(\boldsymbol{x}) \tag{1.39}$$

1. $\boldsymbol{f}_\ell : \mathbb{R}^{N_{\ell-1}} \to \mathbb{R}^{N_\ell}$ (linear)

$$\boldsymbol{f}_\ell : x \mapsto \boldsymbol{f}_\ell(x) = \boldsymbol{W}_\ell \boldsymbol{x} + \boldsymbol{b}_\ell \tag{1.40}$$

with weight matrix $\boldsymbol{W}_\ell = [\boldsymbol{w}_{1,\ell} \cdots \boldsymbol{w}_{N_\ell,\ell}] \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ and bias vector $\boldsymbol{b}_\ell = (b_{1,\ell} \cdots b_{N_\ell,\ell}) \in \mathbb{R}^{N_\ell}$.

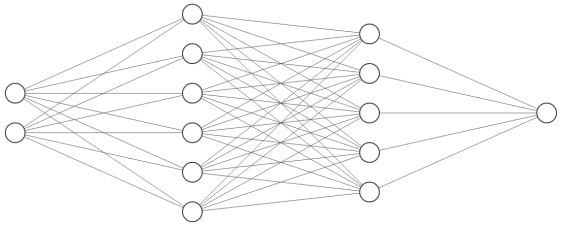


Figure 1.6: Example of a neural network.

2. $\boldsymbol{\sigma}_l : \mathbb{R}^{N_\ell} \to \mathbb{R}^{N_\ell}$ (nonlinear)

$$\boldsymbol{\sigma}_\ell : x = (x_1, \cdots, x_{N_\ell}) \mapsto \boldsymbol{\sigma}_\ell(x) = (\sigma_{1,\ell}(x_1), \cdots, \sigma_{N_\ell,\ell}(x_{N_\ell})), \tag{1.41}$$

They can be used to tackle problems of the form:

$$\arg\min_{\boldsymbol{\theta_w}} \sum_{m=1}^{M} (\boldsymbol{f}(\boldsymbol{x_m}, \boldsymbol{\theta_w}) - \boldsymbol{y_m})^2 + \mu R(\boldsymbol{\theta_w}) \tag{1.42}$$

where $R(\boldsymbol{\theta_w})$ is any regularization term applied to the parameters $\boldsymbol{\theta_w}$. One common type of regularization, called weight decay, is an L_2 penalty on the weight parameters of the network - it is the neural net equivalent of ridge regularization. Neural nets are parametric models and the regularization is again applied to their parameters, in contrast to the previous algorithm in which regularization was applied directly to the function. The MSE loss is given in (1.42) as an example but other losses such as cross-entropy can be used.

One family of activation functions which has been particularly successful is the ReLU family [21][22][23]; its most basic form, a ReLU activation, is the function shown in figure 1.4. ReLU-based networks have an important link with continuous piece-wise linear functions (CPWL), which were the focus of the algorithm in the previous section. Specifically, these networks construct CPWL functions [24], and any CPWL function can be represented by a neural network of this kind [25].

Given the CPWL behaviour of this sucessful family of activation functions, if our goal is to learn the activations of a neural network, reducing the search space to piece-wise linear functions seems to be a sensible option. Unser justifies this intuition mathematically with a Representer Theorem of deep neural networks [7]. Much like the Representer Theorem alluded to in section 1.3.2, which motivated reducing the search space to the span of a Green's function basis dictionary, in 1D, Unser proved a similar result in the context of deep learning, again by using a $\mathrm{TV}^{(2)}$ regularization to promote piece-wise linear solutions [7]. We will briefly present this fundamental result (*Thm.* 3 in [7]), to provide the foundations for the first part of the thesis.

1.5 Deep Splines

Given a neural network with overal composition function as in (1.39), linear weights $\boldsymbol{U_\ell} = [\boldsymbol{u}_{1,\ell} \cdots \boldsymbol{u}_{N_\ell,\ell}]^T \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ which are normalized ($\|\boldsymbol{u}_{n,\ell}\| = 1$), nonlinear activations $\boldsymbol{\sigma}_\ell : \mathbb{R}^{N_\ell} \to \mathbb{R}^{N_\ell}$, and which gets as input a series of data points $(\boldsymbol{x}_m, \boldsymbol{y}_m)_{m=1}^M$,

consider the following $\mathrm{TV}^{(2)}$-regularized problem:

$$\arg \min_{(\boldsymbol{U}_\ell),(\sigma_{n,\ell}\in \mathrm{BV}^{(2)}(\mathbb{R}))} \left(\sum_{m=1}^{M} E(\boldsymbol{y}_m, \boldsymbol{f}(\boldsymbol{x}_m)) + \mu \sum_{\ell=1}^{N} R_\ell(\boldsymbol{U}_\ell) + \lambda \sum_{\ell=1}^{L} \sum_{n=1}^{N_\ell} \mathrm{TV}^{(2)}(\sigma_{n,\ell}) \right) \tag{1.43}$$

where $E : \mathbb{R}^{N_L} \times \mathbb{R}^{N_L} \to \mathbb{R}^+$ can be any convex loss function, $R_\ell : \mathbb{R}^{N_\ell \times N_\ell} \to \mathbb{R}^+$ can be any convex cost on the weights, such as weight decay, and $\lambda_{\mathrm{TV}}, \mu \in \mathbb{R}^+$ are two adjustable regularization parameters. If a solution of (1.43) exists, then one of the solutions is a "deep spline" network with individual activations of the form

$$\sigma_{n,\ell}(x) = b_{1,n,\ell} + b_{2,n,\ell} x + \sum_{k=1}^{K_{n,\ell}} a_{k,n,\ell} (x - \tau_{k,n,\ell})_+, \tag{1.44}$$

with adaptive parameters $K_{n,\ell} \leq M - 2, \tau_{1,n,\ell}, \cdots, \tau_{K_{n,\ell},n,\ell} \in \mathbb{R}$, and $b_{1,n,\ell}, b_{2,n,\ell}$, $a_{1,n,\ell}, \cdots, a_{K_{n,\ell},n,\ell} \in \mathbb{R}$.

So, in a similar fashion to section 1.3.2, this theorem states that, if a solution to (1.43) exists, one of them is of the form (1.44), *i.e.*, a sum of a bounded number K of ReLUs with uknown slopes and locations, plus a linear term. As we will see in the experimental section, the strength of the $\mathrm{TV}^{(2)}$ regularization can be adjusted to produce solutions far from the upper bound, *i.e.*, $K << M$. Again, the linear term is not penalized by the regularization; however, it can still contribute to the reduction of the data-fidelity part of the loss.

1.5.1 Discretization

Notice that the the activation functions in (1.44) are of the same form as the functions in our reduced search space (1.27), for $\mathrm{L} = \mathrm{D}^2$, from the gTV method in section 1.3.2. Hence, the same exact discretization applies here:

$$\mathrm{TV}^{(2)}\{\sigma_{n,\ell}\} = \left\| \sigma''_{n,\ell} \right\|_{\mathcal{M}} = \sum_{k=1}^{K_{n,\ell}} |a_{k,n,\ell}| = \|\boldsymbol{a}_{n,\ell}\|_1 \tag{1.45}$$

And we reach the following discrete optimization objective, which can be minimized by training a neural net in a GPU, using batch gradient descent:

$$\arg \min_{(\boldsymbol{U}_\ell),(\sigma_{n,\ell}\in \mathrm{BV}^{(2)}(\mathbb{R}))} \left(\sum_{m=1}^{M} E(\boldsymbol{y}_m, \boldsymbol{f}(\boldsymbol{x}_m)) + \mu \sum_{\ell=1}^{N} R_\ell(\boldsymbol{U}_\ell) + \lambda \sum_{\ell=1}^{L} \sum_{n=1}^{N_\ell} \|\boldsymbol{a}_{n,\ell}\|_1 \right) \tag{1.46}$$

Chapter 2

Learning Deepsplines

This chapter puts in practice the Representer Theorem of deep neural networks, which was the focus of the previous section, by learning the activations of a neural network using deep splines. In the first section, we formulate the problem in terms of B-splines in a finite-dimensional setting. In the second section, we present the numerical results.

2.1 B-splines Formulation

From the result of the Representer Theorem (1.44), the same questions as in the gTV section 1.3.2 are raised.

First, discovering the location of the knots of the ReLUs is slow. To solve this issue, we can again restrict the knots to a grid (with the grid spacing going to zero we can still approximate any function in the limit). If the solution is composed of only a few knots, we hope that the regularization $\mathrm{TV}^{(2)}$ will eliminate most of the non-relevant knots through its sparsifying effect (or make negligible). This will be seen in the experimental validation.

Second, the formulation in terms of ReLUs is poorly-conditioned; in order to compute the output of the activation we need to know the value at that position of all ReLUs which have knots before, which makes the cost of backpropagation increase exponentially with the number of knots - the linear increase in a single layer is exponentially propagated through layers-. This was experimentally verified (but not shown here). This issue can be circumvented by using the alternative B-splines formulation, whose details were given in section 1.3.2. With this formulation - without considering the computational cost of the regularization term, which is neglible for sufficiently large networks - the cost of the backpropagation becomes $\mathcal{O}(1)$ with respect to the number of coefficients, since we only require the two closest coefficients to determine the output value at a given location, through linear interpolation; this is the advantage of using localized basis functions.

2.1.1 Finite-Dimensional Problem

As mentioned in section 1.3.2, for a practical implementation, we need to transform this problem into a finite-dimensional one. If we assume that the large majority of data points falls within a region of interest $[-L, L]$, we can restrict the B-spline

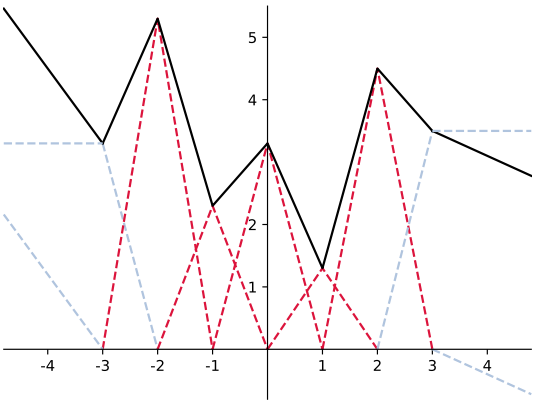


Figure 2.1: Finite spline representation;
dashed red: B1-spline basis functions;
dashed gray: boundary basis functions.

coefficients c_k to be in that region. Outside this domain, we can perform linear extrapolation, so that the function is not penalized by the $\mathrm{TV}^{(2)}$ there.

The constructed function is shown in figure 2.1 with $L = 4$. Mathematically, these linear boundary conditions can be done with carefully chosen boundary basis, shown in dashed gray. In the implementation, we use the first two coefficients (not shown in 2.1, but corresponding to $x = -4$ and $x = -3$) to perform the left extrapolation, and similarly for the right extrapolation. Note that, in this finite representation, the linear term is determined by the left extrapolation and the ReLU knots are restricted to the region of interest (the first and last ReLU knots are at $L + 1$ and $L - 1$, respectively). We can summarize this statement and express the relationship with the Green's function expansion as follows:

$$\begin{aligned}\sigma_{spline}(x) &= \begin{cases} \sum_{k=-L}^{L} c_k \Delta(x-k) & \text{for } x \in [-L, L] \\ c_{-L} + (c_{-L} - c_{-L+1}) \cdot ((-L) - x) & \text{for } x \in (-\infty, L) \\ c_{L-1} + (c_L - c_{L-1}) \cdot (x - (L-1)) & \text{for } x \in (L, \infty) \end{cases} \\ &= b_1 + b_2 x + \sum_{k=-(L-1)}^{(L-1)} a_k [x-k]_+ \end{aligned} \tag{2.1}$$

The relationship between the coefficients is given by $\boldsymbol{a} = \mathbf{L}\mathbf{c}$ (see 1.38), where $\mathbf{L}$ is a $(2L-2) \times (2L)$ Toeplitz matrix and the two degrees of freedom correspond to the linear term (null space):

$$\begin{bmatrix} a_{-L+1} \\ \vdots \\ a_{L-1} \end{bmatrix} = \begin{bmatrix} 1 & -2 & 1 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & -2 & 1 & 0 & \cdots & 0 \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} c_{-L} \\ c_{-L+1} \\ \vdots \\ c_{L-1} \\ c_L \end{bmatrix} \tag{2.2}$$

In summary, the regularization term in (1.46) is computed through a valid convolution of the B-splines coefficients $\boldsymbol{c}$ with the discrete D^2 filter and then taking the $\|\cdot\|_1$ of the result.

2.2 Numerical Experiments

In this section, we use deep splines in an area classification problem. The aim of these experiments is to better understand the behaviour of these objects by restricting ourselves to learning functions which map a two-dimensional real input to a discrete value. Their performance will be benchmarked against other common activations, the ReLU and the PReLU.

The task at hand consists of classifying points in a 2D space as being inside or outside a circle which occupies half the area of the square $[-1,1]^2$. The underlying function $f : [-1,1]^2 \mapsto \{0,1\}$ maps a 2D data point to the value 1 if it is inside the circle, and to the value 0 otherwise:

$$f(x,y) = \begin{cases} 1, & \text{for } x^2 + y^2 < \frac{2}{\pi} \\ 0, & \text{otherwise} \end{cases} \tag{2.3}$$

A training set S_i consists of M $(= 1000)$, position/label pairs: each (x,y) position is randomly chosen within the square $[-1,1]^2$ and the corresponding label is assigned according to (2.3). One training set is shown in figure 2.2.

To be able to interpret the results, we tackle the problem using a simple architecture, shown in figure 2.3a. This network takes a 2D input, which is fed to a single hidden layer of N_{hidden} neurons and outputs a real-value. A sigmoid function (figure 1.3a) is applied to convert this real value into a probability (of the input belonging to the circle), such that $f_S(x,y) \in [0,1]$. During training, binary-cross entropy is used to update the network parameters:

$$\mathcal{L}_S(f_S) = \frac{1}{M} \sum_{i=1}^{M} -\log(f_S(x,y)) \tag{2.4}$$

The test set consists of a fine grid with spacing 0.01 covering the $[-1,1]^2$ square (200·200 datapoints). During testing, the output value is quantized into a prediction:

$$\hat{f}_S(x,y) = \begin{cases} 1, & \text{if } f_S(x,y) > 0.5 \\ 0, & \text{otherwise} \end{cases} \tag{2.5}$$

And the test accuracy is computed as:

$$\text{accuracy } (\%) = \frac{\#\text{ correct predictions}}{\#\text{ total predictions}} \times 100 \tag{2.6}$$

We will benchmark the deep spline against two different activations: the ReLU and the PReLU. The PReLU, shown in figure 2.3b, is a piece-wise linear function with a single knot at zero, like the ReLU, but which has learnable slope a in the negative part of the domain.

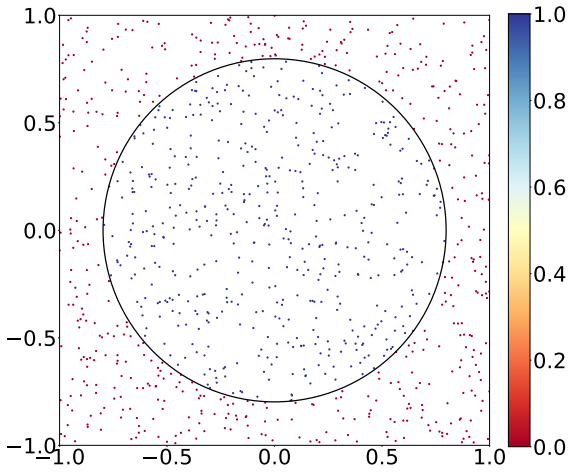
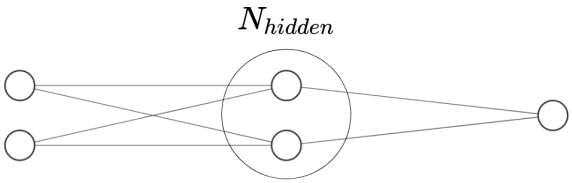
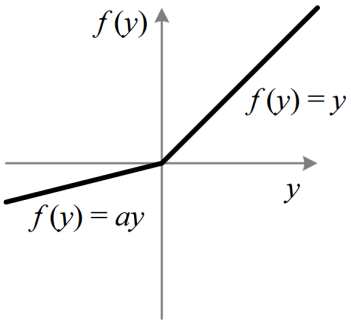


Figure 2.2: Example of a training dataset S;
red dots: label 0; blue dots: label 1.



(a) Deep spline network architecture;
N_{hidden} is the number of hidden neurons.



(b) PReLU activation.
(source: [22])

Figure 2.3: deep spline network architecture; PReLU.

An L_2 penalty (weight decay) is applied to the weight parameters of the network. The weight decay, μ, is gridsearched with a logarithmic step in the range $[10^{-10}, 10^{-2}]$.

Five training datasets $\{S_1, \ldots, S_5\}$ are independently sampled for training. For each weight decay, one model is trained on each dataset and then the corresponding test accuracies are computed, including the median accuracy of the five models. The weight decay which achieved the best median accuracy is selected and, for that weight decay, the test results of the model trained on the median accuracy dataset are reported.

For the deep spline network, the $\mathrm{TV}^{(2)}$ regularization weight (λ) is computed from the weight decay according to (theoretical work in preparation):

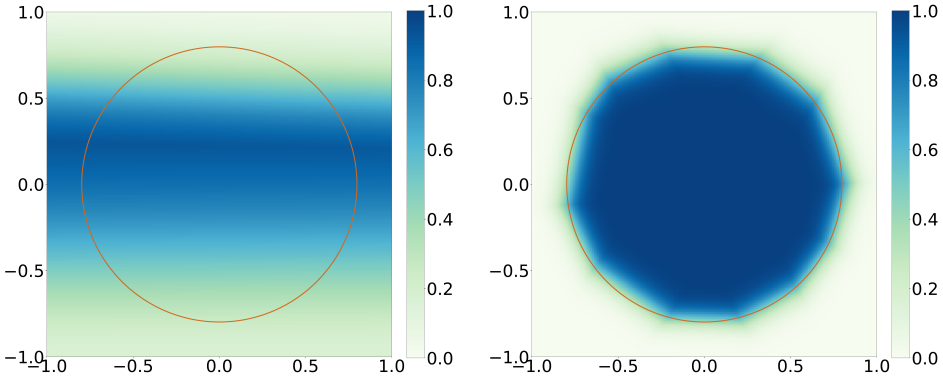
$$\lambda = \frac{16}{33}\mu \tag{2.7}$$

The Pytorch framework [26] was used for the experiments. The models were trained for a total of 250 epochs with an Adam optimizer [27], which uses the gradients in a smarter way than the vanilla-BGD discussed in the introduction; the initial learning rate was set to 10^{-3} and was decreased by 10 at epochs 175 and 225.

A small batch size of 10 was necessary to avoid local minima (by introducing more stochasticity in the gradients).

In the next two sections, we show the results of the experiments, first for the baseline networks and then for the deep spline. In terms of performance, what we are concerned about is the relationship between the accuracy and the size of the network/total number of parameters, so both these quantities are shown alongside the test accuracies. The color value of the plots represents the network prediction (probability map).

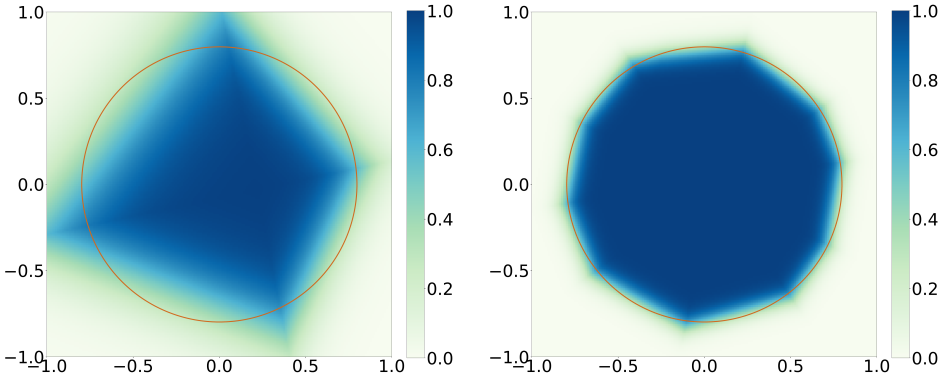
2.2.1 Baseline Networks



(a) $N_{hidden} = 2$,
test accuracy: 74.65%,
N.o parameters: 9.

(b) $N_{hidden} = 10$,
test accuracy: 98.15%,
N.o parameters: 41.

Figure 2.4: ReLU; median results for the best weight decay.



(a) $N_{hidden} = 2$,
test accuracy: 89.42%,
N.o parameters: 11.

(b) $N_{hidden} = 10$,
test accuracy: 98.19%,
N.o parameters: 51.

Figure 2.5: PReLU; median results for the best weight decay.

The results for the baseline networks are shown in figures 2.4 and 2.5. As we can observe, with only two hidden neurons, both the ReLU and PReLU networks do not perform very well (although the PReLU performs much better). To achieve an accuracy above 98%, we need 10 hidden neurons and a total number of 41/51

network parameters. Notice that the increased number of parameters in the PReLU starts to make less of a difference as the network size increases.

2.2.2 Deep Spline Network

For the deep spline network, we initialize the activations with an even/odd initialization: half of the activations in the hidden layer are initialized with the absolute value function and the other half are initialized with the soft-threshold/shrink function, shown in figure 2.6, from compressed sensing [28]. This initialization is more intelectually satisfying than a random one since any function can be represented as a sum of an even and an odd function and this idea is encountered throughout signal processing (*e.g.*, Fourier series).

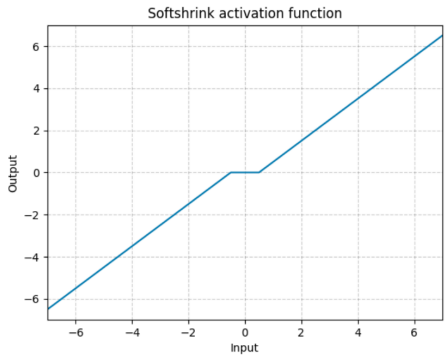


Figure 2.6: Softshrink. [1]

As mentioned previously, the $\mathrm{TV}^{(2)}$ regularization of the activations promotes piece-wise linear solutions which have a reduced number of non-zero slopes (few knots). In practice, negligible slopes are never fully eliminated. As an attempt to have a tight control on the number of knots, we propose a "sparsification" algorithm. This algorithm goes from the B-spline expansion coefficients to the Green's function expansion coefficients ($\boldsymbol{c} \to \boldsymbol{a}$), "kills" the knots with negligible slope ($\boldsymbol{a} \to \hat{\boldsymbol{a}}$), and finally goes back to B-spline expansion coefficients again ($\hat{\boldsymbol{a}} \to \hat{\boldsymbol{c}}$). The algorithm is as follows (assuming grid spacing $h = 1$, WLOG):

1. $\boldsymbol{a} = \mathbf{L}\mathbf{c}$, as in equation (2.2).

2. Apply threshold: for $j \in \{-L+1, \ldots, L-1\}$,

$$\hat{a}_j = \begin{cases} 0, & \text{if } |a_j| < \text{atol} \\ a_j, & \text{otherwise} \end{cases} \tag{2.8}$$

[1] `https://pytorch.org/docs/master/nn.html`

3. $\hat{\boldsymbol{c}} = g(\hat{\boldsymbol{a}})$:

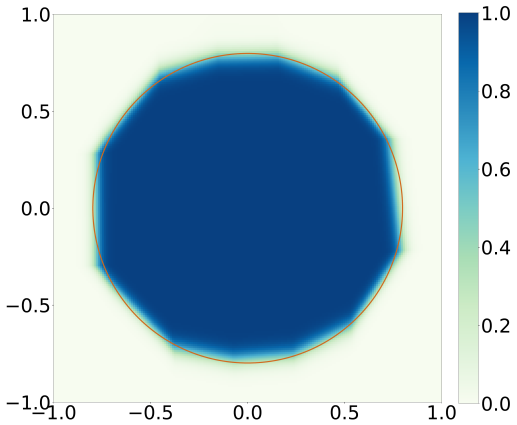
$$\begin{bmatrix} \hat{c}_{-L} \\ \hat{c}_{-L+1} \end{bmatrix} = \begin{bmatrix} c_{-L} \\ c_{-L+1} \end{bmatrix}$$

$$\begin{bmatrix} \hat{c}_{-L+2} \\ \vdots \\ \hat{c}_{L} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 & 0 \\ 2 & 1 & \cdots & \cdots & 0 & 0 \\ \vdots & \vdots & & & \vdots & \vdots \\ 2L-3 & 2L-4 & \cdots & \cdots & 1 & 0 \\ 2L-2 & 2L-3 & \cdots & \cdots & 2 & 1 \end{bmatrix} \begin{bmatrix} \hat{a}_{-L+1} \\ \hat{a}_{-L+2} \\ \vdots \\ \hat{a}_{L-2} \\ \hat{a}_{L-1} \end{bmatrix} + \begin{bmatrix} c_{-L+1} \\ \vdots \\ c_{-L+1} \end{bmatrix} \tag{2.9}$$

Step 2 of the algorithm eliminates all slopes whose absolute value is smaller than atol. To choose the absolute threshold we procede as follows: after training the model, we observe the training accuracy; then, we start from a low atol value and slowly increase it until the train accuracy drops by more than 1%; finally, the highest threshold for which the accuracy drops by less than 1% is chosen.

Step 3 goes back to the B-spline representation, after the elimination of the negligible slopes. Since the linear term vanishes in the previous transformation, c_{-L} and c_{-L+1}, which define the linear term, are saved before the last step and are assigned the same values. The rest of the coefficients are computed as shown, taking into account the value of c_{-L+1}.

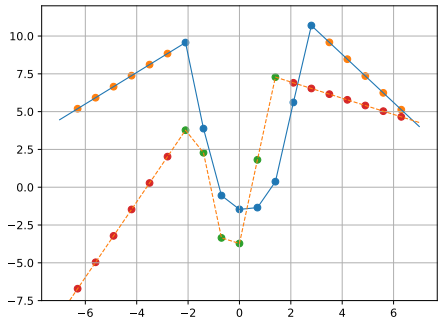
Here we present the results for deep spline network with just 2 hidden neurons and 21 B1-spline coefficients in each activation.



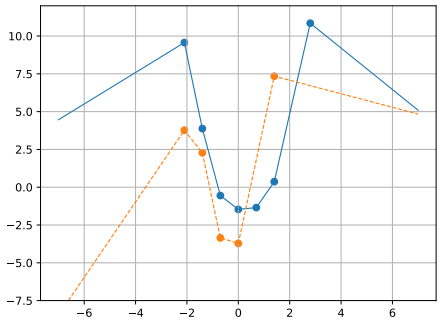
(a) $N_{hidden} = 2$,
test accuracy: 98.54%,
N.o parameters: 23

Figure 2.7: deep spline after sparsification, 21 B-spline coefficients.

The activations and the effect of sparsification are shown in figure 2.8. The dotted points correspond to knots with a non-zero slope. Note that, before sparsification, some of the present knots are visibly neglible. Sparsification provides a tighter control by "killing" these knots. The number of parameters mentioned in figure 2.7 accounts for the number of non-zero a_k slopes after sparsification (12 in total), the b_0, b_1 coefficients which define the linear term for each activation (4 in total), and the remaining network parameters. Note that one of the activations has the shape



(a) Before sparsification;
the dotted knots have a non-zero slope.



(b) After sparsification;
the dotted knots have a non-zero slope.

Figure 2.8: Activations of the model corresponding to figure 2.7.

of a parabola, which makes sense given that the decision boundary is of the form $x^2 + y^2 - c < 0$.

We observe in figure 2.7 that using a deep spline with only two hidden neurons and a total of 23 network parameters we were able to achieve a better accuracy than the ReLU and PReLU networks, with 41 and 51 parameters, respectively. We can conclude that deep splines allow us to use smaller networks and that the transfer of capacity from the network weights to the activations is beneficial.

Chapter 3

Hessian-Schatten Regularization

3.1 Introduction

The complex implicit parameterization of neural networks leads to a reduced model interpretability, so alternative higher-order methods based on a stronger mathematical foundation are worth exploring. In this Part 2 of the thesis, we approach again the ideas from the gTV B-splines method (section 1.3.2) by presenting a new regularization framework for learning functions $f : \mathbb{R}^2 \mapsto \mathbb{R}$ and proposing a method to solve the continuous domain problem via exact discretization. This framework is called Hessian-Schatten, because the regularization is based on the Schatten norm of the Hessian (more on this later).

The problem we which to solve is of the form:

$$\arg\min_{f} \sum_{m=1}^{M} (f(x_m, y_m) - z_m^{obs})^2 + \lambda \mathcal{HS}(f) \tag{3.1}$$

Note the change in notation: z_m^{obs} is now used for the observations instead of y_m and the input, which was previously denoted as $\boldsymbol{x}_m$, is now explicitly written as (x_m, y_m). This emphasizes the 2D nature of the problem. The problem formulation is non-parametric like in the RKHS and gTV cases.

Before defining $\mathcal{HS}(f)$, we first need to mention the concepts of SVD, Schatten norm and Hessian matrix, in which it is based; this is the purpose of the first three subsections. The remaining subsections of the introduction will define and discuss this regularization term and overview the work that has been done on it.

3.1.1 SVD

The SVD is a matrix decomposition of the form $\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^*$ where $\mathbf{U}$ and $\mathbf{V}$ are unitary matrices ($\mathbf{U}\mathbf{U}^* = \mathbf{U}^*\mathbf{U} = \mathbf{I}$) and $\mathbf{S}$ is diagonal. Every matrix $\mathbf{A} \in \mathbb{F}^{n \times m}$ has an SVD decomposition [29]. The elements in the diagonal of $\mathbf{S}$ are called the singular values, defined as the square roots of the non-negative eigenvalues of $\mathbf{A}\mathbf{A}^*$ (the same as of $\mathbf{A}^*\mathbf{A}$). The number of non-zero singular values determines the rank of the matrix. Note that if $\mathbf{A}$ is a normal matrix, *i.e.*, $\mathbf{A}\mathbf{A}^* = \mathbf{A}^*\mathbf{A}$ (*e.g.*, an hermitian matrix), the spectral theorem applies and $\mathbf{A}$ will have an eigen-decomposition of the form $\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}$ where $\mathbf{U}$ is unitary. Then, $\sqrt{\mathbf{A}\mathbf{A}^*} = \mathbf{U}\,|\mathbf{\Lambda}|\,\mathbf{U}^*$, which means that the singular values are equal to the absolute values of the eigenvalues of $\mathbf{A}$.

3.1.2 Schatten Norm

Given a vector space V over a vector field $\mathbb{F}$, a norm is a function $p : V \mapsto [0, +\infty)$ which satisfies the following properties for any $\boldsymbol{u}, \boldsymbol{v} \in V$ and any scalar $a \in \mathbb{F}$:

1. Triangle inequality; $p(\boldsymbol{u} + \boldsymbol{v}) \leq p(\boldsymbol{u}) + p(\boldsymbol{v})$
2. absolute homogeneity; $p(a\boldsymbol{u}) = |a|\, p(\boldsymbol{u})$
3. positive definiteness; $p(\boldsymbol{u}) = \mathbf{0}$ iff $\boldsymbol{u} = 0$.

The Schatten norm is a matrix norm, *i.e.*, it satisfies the conditions above with $V = \mathbb{F}^{m \times n}$, defined as:

$$\|\mathbf{A}\|_{S_p} = \|\boldsymbol{\sigma}\|_{\ell_p} = \left(\sum_{k=1}^{\min(n_1, n_2)} \boldsymbol{\sigma}_k^p(\mathbf{A}) \right)^{\frac{1}{p}} \tag{3.2}$$

From this definition, we see that the Schatten norm of a matrix $\mathbf{A}$ is computed by taking its SVD decomposition and then computing the ℓ_p norm of the vector of singular values. Apart from the basic norm properties, the Schatten norm also satisfies several additional properties, one of which is unitary invariance:

$$\|\mathbf{UAV}\|_{S_p} = \|\mathbf{A}\|_{S_p}, \text{ for } \mathbf{U}, \mathbf{V} \text{ unitary matrices.} \tag{3.3}$$

In this project, we expect to find symmetry in the results because of this property.

To better understand the definition of the Schatten norm, let us highlight the equivalence with other well-known norms, for specific values of p. The last case, where $p = 1$, will be the focus of this part of the project.

$\boldsymbol{p = \infty}$:

$$\|\mathbf{A}\|_{S_\infty} = \sigma_{max}(\mathbf{A}) = \sup\{\|\mathbf{A}\boldsymbol{x}\|_2 : \|\boldsymbol{x}\|_2 = 1\} = \|\mathbf{A}\|_2 \tag{3.4}$$

When $p = \infty$, the Schatten norm is equal to the operator/spectral norm, *i.e.*, the maximum singular value. Since $\mathbf{U}$ and $\mathbf{V}$ are unitary, they only perform "rotations", and $\mathbf{S}$ scales the vectors after the $\mathbf{V}$ transformation. Hence, we can view the maximum singular value as the maximum possible "stretch" of a vector after the $\mathbf{A}$ transformation.

$\boldsymbol{p = 2}$:

$$\|\mathbf{A}\|_{S_2} = \Big(\sum_{k=1}^{\min(n_1, n_2)} \boldsymbol{\sigma}_k^2(\mathbf{A}) \Big)^{\frac{1}{2}} = \sqrt{\mathrm{tr}(\mathbf{A}^*\mathbf{A})} = \sqrt{\sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \left| a_{ij}^2 \right|} = \|\mathbf{A}\|_{\mathrm{F}} \tag{3.5}$$

When $p = 2$, the Schatten norm is equal to the Frobenius norm, which is the matrix norm induced by the inner product $\langle \mathbf{A}, \mathbf{B} \rangle = \mathrm{tr}(\mathbf{A}^*\mathbf{B})$. The second equality comes from the fact that $\mathrm{tr}(\mathbf{X}^*\mathbf{X})$ is equal to the sum of the eigenvalues of $\mathbf{X}^*\mathbf{X}$ which, by definition, is equal to the sum of the square singular values. This norm arises in multiple contexts - in the context of neural nets, it is called weight decay.

$\boldsymbol{p = 1}$:

$$\|\mathbf{A}\|_{S_1} = \sum_{k=1}^{\min(n_1,n_2)} \boldsymbol{\sigma}_k(\mathbf{A}) = \mathrm{tr}(\sqrt{\mathbf{A}^*\mathbf{A}}) = \|\mathbf{A}\|_* \tag{3.6}$$

When $p = 1$, the Schatten norm is equal to the nuclear/trace norm $\|\mathbf{A}\|_*$. This norm is a convex envelope of rank($\mathbf{A}$), so it is often used in mathematical optimization to search for low rank matrices. Since the rank of a matrix is the number of non-zero singular values - a.k.a. the L_0 "norm" of the vector of singular values, which is not convex - it can be relaxed to a convex minimization problem using the L_1 norm instead [30].

As mentioned previously, for normal matrices, the singular values are equal to the absolute value of the eigenvalues. Hence,

$$\|\mathbf{A}\|_{S_1} = \sum_{k=1}^{n} |\lambda_k(\mathbf{A})| = \|\boldsymbol{\lambda}(\mathbf{A})\|_{\ell_1}, \qquad \text{for } \mathbf{A} \text{ normal} \tag{3.7}$$

So the S_1 norm of a normal matrix is equal to the ℓ_1 norm of its eigenvalues.

3.1.3 Hessian Operator

The Hessian matrix $\mathcal{H}f(x, y) \in \mathbb{R}^{2\times 2}$ of a twice-differentiable function $f : \mathbb{R}^2 \mapsto \mathbb{R}$ at (x, y) is the matrix of second partial derivatives:

$$\mathcal{H}f(x, y) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2}(x, y) & \frac{\partial^2 f}{\partial x \partial y}(x, y) \\ \frac{\partial^2 f}{\partial y \partial x}(x, y) & \frac{\partial^2 f}{\partial y^2}(x, y) \end{bmatrix} \tag{3.8}$$

Henceforth, we will use the notation $\mathbf{H} := \mathcal{H}f(x, y)$ unless it is important specify the function f or the coordinates (x, y) for which the Hessian is computed.

Schwarz's theorem states that if the second partial derivatives are continuous in the neighborhood of (x, y), then $\mathbf{H}$ is symmetric. However, there might be cases for which this condition is not satisfied but $\mathbf{H}$ is still symmetric; we will see such a case in our method. For now, it is important to get a geometric intuition of the eigen-decomposition of a symmetric Hessian. With that goal in mind, first note that the Hessian is useful in computing second directional derivatives [31]:

$$\nabla^2_{\boldsymbol{v}} f = \boldsymbol{v}^T \mathbf{H} \boldsymbol{v} \tag{3.9}$$

where $\boldsymbol{v}$ is a unit vector. If the Hessian is symmetric, according to the spectral theorem, it will have an eigen-decomposition of the form:

$$\mathbf{H} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^* \tag{3.10}$$

where the columns of $\mathbf{U}$ are formed by orthonormal eigenvectors ($\mathbf{U}$ is unitary). The eigenvectors of $\mathbf{H}$ have an important geometric interpretation: they point in the direction of minimal and maximal magnitudes of the second derivative. This can be seen as follows; note that we can write any unit vector as a linear combination of the eigenvectors, since their combination spans $\mathbb{R}^2$:

$$\boldsymbol{v} = a_1\boldsymbol{v}_1 + a_2\boldsymbol{v}_2, \quad a_1^2 + a_2^2 = 1 \tag{3.11}$$

So, the directional derivative in the direction $\boldsymbol{v}$ is:

$$\begin{aligned}\nabla^2_{\boldsymbol{v}} f &= (a_1\boldsymbol{v}_1 + a_2\boldsymbol{v}_2)^T \mathbf{H}(a_1\boldsymbol{v}_1 + a_2\boldsymbol{v}_2) = \\ &= \lambda_1 a_1^2 + \lambda_2 a_2^2.\end{aligned} \tag{3.12}$$

where we used the orthogonality of the eigenvectors ($\boldsymbol{v}_1^T\boldsymbol{v}_2 = 0$). The last expression is a convex combination of the eigenvalues so, assuming $|\lambda_1| \leq |\lambda_2|$ (WLOG), it is easy to see that:

$$0 = |\lambda_1| \leq \left|\nabla^2_v f\right| \leq |\lambda_2| \tag{3.13}$$

where the limits are attained when $a_1 = 1, a_2 = 0$ and $a_1 = 0, a_2 = 1$, respectively. From this expression we conclude that the eigenvalues of the Hessian provide the bounds for the magnitude of the second directional derivatives at (x, y), with the minimum and largest values being attained in the direction of the eigenvectors.

One final point to consider is the following: the Hessian $\mathcal{H}f(x, y)$ of a function f which is locally linear function at (x, y) is the $\mathbf{0}$ matrix. This will be important later.

3.1.4 Hessian-Schatten

We are now in position to define the $\mathcal{HS}(f)$ regularization, first introduced in [10]:

$$\begin{aligned}\mathcal{HS}(f) = \int_{(x,y)\in\Omega} \|\mathcal{H}f(x,y)\|_{S_1}\, dxdy &= \int_{(x,y)\in\Omega} \|\boldsymbol{\sigma}(x,y)\|_{\ell_1}\, dxdy \\ &= \int_{(x,y)\in\Omega} \left(\sum_{k=1}^{2} \sigma_k(x,y)\right) dxdy\end{aligned} \tag{3.14}$$

where $\sigma_k(x, y)$ is the k-th singular value of $\mathcal{H}f(x, y)$. The last equality comes from the fact that the singular values are non-negative ($\sigma_k(x, y) = |\sigma_k(x, y)|$). It is worth emphasizing the steps required to compute this regularization: we calculate the Hessian of f at each location $(x, y) \in \Omega$; we then take its SVD decomposition and compute the ℓ_1 norm of the vector of singular values; finally, we integrate over the whole space Ω to get a single value.

This regularization (and its most general S_p form) enjoys several good properties: it is scale, rotation and translation-invariant [10]; the rotation invariance is linked to the unitary invariance of Schatten norms. These properties make the Hessian-Schatten particularly suitable for natural images, which preserve their structure under these transformations.

3.1.5 The Effect of the Regularization

For future reference, let us repeat here the minimization problem we wish to solve:

$$\arg\min_f \sum_{m=1}^{M} (f(x_m, y_m) - z_m^{obs})^2 + \lambda\mathcal{HS}(f) \tag{3.15}$$

Now that we defined the Hessian-Schatten, we can discuss its effect. The presence of the ℓ_1 term in (3.14) means that the regularizer will promote solutions with fewer

non-zero singular values (sparser). For a normal matrix, $\sigma_k = |\lambda_k|$, $\|\boldsymbol{\sigma}(x,y)\|_{\ell_1} = \|\boldsymbol{\lambda}(x,y)\|_{\ell_1}$, so this is equivalent to promoting sparsity in the eigenvalue sense. This sparsity-promoting behaviour is the reason why we restrict ourselves in this project to S_1 norms, even though a more general version of (3.14) exists [10].

As mentioned before, if the function is locally linear at (x, y), the Hessian there is identically $\mathbf{0}$, which is the sparsest result possible (both singular values are zero). Therefore, this regularization will favor piece-wise linear solutions, by analogy with the D^2 operator in 1D.

3.1.6 Discrete Hessian

Lefkimmiatis *et al.*[10] addresses the problem above by descretizing the Hessian matrix using second finite differences, in a context where the underlying function corresponds to an image (takes discrete indices as input and outputs a real value); the image intensities on a $N_x \times N_y$ grid are rasterized in a vector $\boldsymbol{z}^{obs} \in \mathbb{R}^M$ ($M = N_x \cdot N_y$). Then, the discrete Hessian operator $\boldsymbol{\mathcal{H}} : \mathbb{R}^M \mapsto \mathbb{R}^{M\times 2\times 2}$ is applied to the vectorized image intensities $\boldsymbol{z}^{obs}$ as follows:

$$[\boldsymbol{\mathcal{H}}\boldsymbol{z}^{obs}]_m = \begin{bmatrix} [\Delta_{r_1r_1}\boldsymbol{z}^{obs}]_m & [\Delta_{r_1r_2}\boldsymbol{z}^{obs}]_m \\ [\Delta_{r_2r_1}\boldsymbol{z}^{obs}]_m & [\Delta_{r_2r_2}\boldsymbol{z}^{obs}]_m \end{bmatrix}, \quad m = 1, \ldots, M \tag{3.16}$$

where $\Delta_{r_ir_j}$ are second finite difference operators (see [10] for more details).

Discretizing the Hessian with finite second differences has several disavantages. First, it naturally leads to discretization errors. Second, it hides the implicit assumption that the underlying function is piece-wise constant (step function), which is unnatural and not intelectually satisfying. We can argue that images can, in fact, be seen as step functions in continuous domain, but we have to remember that they are only samples of a continuous, and usually smooth, underlying signal. More insight and greater flexibility is achieved by modeling this underlying signal instead (learning over function spaces), without changing the nature of the problem.

In this thesis, we follow an alternative path, analogous to the gTV B-splines method, where we exactly discretize the problem (3.15) by restricting the search space to the space spanned by shifts (in a grid) of hexagonal box-splines. This is the topic of the next section.

3.2 Continuous Formulation

3.2.1 Search Space

To solve the problem (3.15), much in the spirit of the gTV B-splines method, we reduce the search space to continuous piece-wise linear functions, which are already favored by the Hessian-Schatten, thus reflecting the properties of the regularization. Note that, in this case, we don't have (yet) a corresponding 2D continuous Representer Theorem stating that piece-wise linear functions are solutions of the problem (3.15), as *Thm. 1* in [9]. However, this restriction will allow us to find a simple expression for the Hessian-Schatten, while achieving an exact discretization - transforming the continuous problem into one of finding finitely many coefficients $\mathbf{z}$.

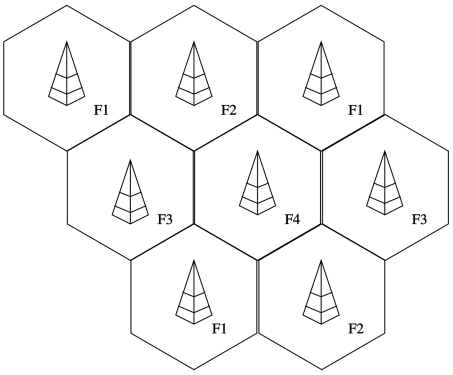


Figure 3.1: Antenna frequency reuse pattern.

A convenient way of parameterizing a CPWL function $f : \Omega \mapsto \mathbb{R}$, $\Omega \subset \mathbb{R}^2$, is by triangulating the function space. Consider the construction of a 2D lattice, centered at $(0, 0)$, formed by lattice vectors $\boldsymbol{\nu}_1$ and $\boldsymbol{\nu}_2$. The lattice is a set Λ of lattice points (standard coordinates), each corresponding to an index pair in Θ (lattice coordinates):

$$\Theta \coloneqq \left\{ (i,j) \ : i,j \in \left\{ -\frac{P}{2}, -\frac{P}{2}+1, \ldots, \frac{P}{2}-1, \frac{P}{2} \right\} \right\} \tag{3.17}$$
$$\Lambda \coloneqq \{ i\boldsymbol{\nu}_1 + j\boldsymbol{\nu}_2 \ : (i,j) \in \Theta \}$$

where $P \in \mathbb{N}^+$ is the grid size. We use the following notation for the lattice vertices:

$$(x_{i,j}, y_{i,j}) = i\boldsymbol{\nu}_1 + j\boldsymbol{\nu}_2 \tag{3.18}$$

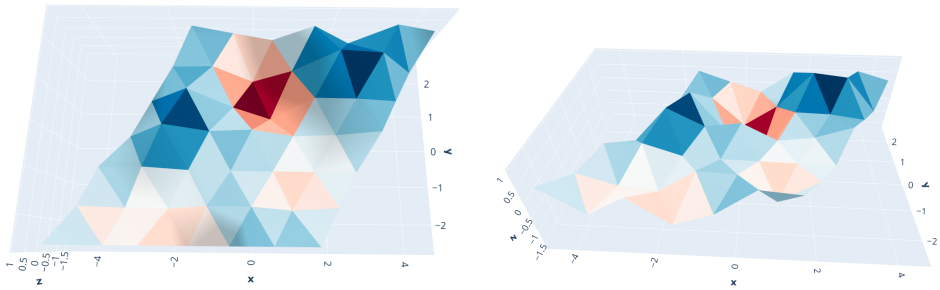
Each lattice point is assigned a value, so we have a a corresponding set $f(\Lambda)$:

$$f(\Lambda) \coloneqq \{ f(x_{i,j}, y_{i,j}) : (x_{i,j}, y_{i,j}) \in \Lambda \} \tag{3.19}$$

Since each group of three vertices can define an affine subspace, we are able to parameterize continuous piece-wise linear functions with the pair $(\Lambda, f(\Lambda))$, by analogy with the 1D case. This will become more evident in the next subsections.

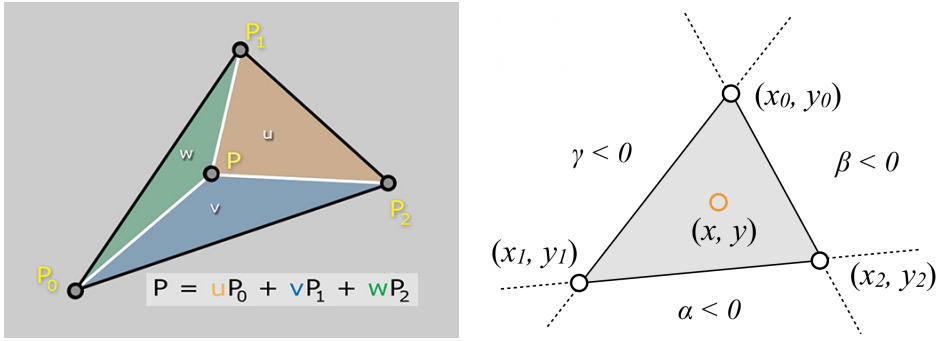
We might now ask what is a good way to tile the space, *i.e.*, choose $\boldsymbol{\nu}_1$, $\boldsymbol{\nu}_2$. One solution that is often seen in the literature is hexagonal tiling, based on a regular hexagonal structure. This tiling is used in several contexts including antenna frequency reuse patterns (figure 3.1) and is motivated by the proved Honeycomb conjecture, which states that an hexagonal grid is the optimal way of dividing a surface into equal-area regions, having the least total perimeter [32], thus providing a dense tiling of the space.

To replicate this regular hexagonal structure, we can choose $\boldsymbol{\nu}_1 = \frac{1}{h} \cdot (1, 0)$ and $\boldsymbol{\nu}_2 = \frac{1}{h} \cdot (\frac{1}{2}, \frac{\sqrt{3}}{2})$, leading to a lattice shown in figure 3.2, with $P = 6$ and $h = 1$. As we will see later, this will allow our CPWL function space to be spanned by shifts of hexagonal "box-splines". The next section will explain how the pair $(\Lambda, f(\Lambda))$ is able to parameterize a CPWL function space over Ω.



(a) Lattice, up view. (b) Lattice, side view.

Figure 3.2: Hexagonal triangulation for $P = 6$;
the vertex values are randomly initialized.



(a) Barycentric coordinate system. (b) Barycentric coordinates inside the triangle are all positive.

Figure 3.3: Barycentric coordinates.

3.2.2 Barycentric coordinates

Barycentric coordinates allow computing $f(x)$, for any $x \in \Omega$, from the pair $(\Lambda, f(\Lambda))$. They are used in computer graphics to interpolate data from vertices (vertex shading); for example, by defining a normal for each vertice, which determines how light is reflected there, we can compute the normals at other points inside the triangles by interpolating the corresponding vertex normals, so that we know how light is reflected at every location [33].

The barycentric coordinates $\boldsymbol{\lambda}$ of a point $p = (x, y)$ w.r.t. the triangle with vertices $p_1, p_2, p_3 \in \Lambda$, which we write as the set $\{p_1, p_2, p_3\} = \{(x_1, y_1), (x_2, y_2), (x_3, y_3)\}$, allow expressing that point as an affine combination of the triangle vertices:

$$\begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{bmatrix} \boldsymbol{\lambda} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.20}$$

The last row constrains this combination to be affine, *i.e.*, $\sum_i \lambda_i = \alpha + \beta + \gamma = 1$. We use the notation $\boldsymbol{\lambda} = \mathrm{Bar}(p, \{p_1, p_2, p_3\})$.

The matrix above is invertible iff the three triangle vertices do not fall along a line (the triangle is not degenerate), so we can get the barycentric coordinates of (x, y) through:

$$\boldsymbol{\lambda} = \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \left(\begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{bmatrix} \right)^{-1} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.21}$$

Barycentric coordinates are very useful in checking to which triangle a point belongs to, since they follow the property shown in figure 3.3 (a): if all the barycentric coordinates are positive, the point is inside the triangle, otherwise it is outside (or in the boundary) [33]. This is used in the algorithm for this purpose. After knowing the triangle to which a point (x, y) belongs to and its corresponding barycentric coordinates, we can compute $f(x, y)$ through interpolation. Suppose the plane equation of the triangle with vertices $(x_1, y_1), (x_2, y_2), (x_3, y_3) \in \Lambda$, and corresponding values $f(x_1, y_1), f(x_2, y_2), f(x_3, y_3) \in f(\Lambda)$, is:

$$z = f(x, y) = ax + by + d \tag{3.22}$$

- to find this plane equation we can simply define a linear system with three equations (one per vertice), $z_i = ax_i + by_i + d,\ i \in \{1, 2, 3\}$, and solve for a, b, d.- Then, the value at (x, y) of a point with barycentric coordinates $\boldsymbol{\lambda} = (\alpha, \beta, \gamma)$ is:

$$f(\alpha(x_1, y_1) + \beta(x_2, y_2) + \gamma(x_3, y_3)) = \tag{3.23}$$

$$= a(\alpha x_1 + \beta x_2 + \gamma x_3) + b(\alpha y_1 + \beta y_2 + \gamma y_3) + (\alpha + \beta + \gamma)d = \tag{3.24}$$

$$= \alpha f(x_1, y_1) + \beta f(x_2, y_2) + \gamma f(x_3, y_3) \tag{3.25}$$

where we used $\alpha + \beta + \gamma = 1$. We conclude that $f(x, y)$ can simply be obtained through an affine combination of the values at the vertices, with the barycentric coordinates of (x, y) being the weights.

An useful property of barycentric coordinates, that will be used later, is that they are invariant under affine transformations of the point p and the triangle vertices p_1, p_2, p_3 to which it belongs to, *i.e.*:

$$\begin{aligned} \boldsymbol{\lambda} &= \text{Bar}((x, y), \{(x_1, y_1), (x_2, y_2), (x_3, y_3)\}) \\ &= \text{Bar}(\mathbf{T}(x, y) + \boldsymbol{a}, \{\mathbf{T}(x_1, y_1) + \boldsymbol{a}, \mathbf{T}(x_2, y_2) + \boldsymbol{a}, \mathbf{T}(x_3, y_3) + \boldsymbol{a}\}) \end{aligned} \tag{3.26}$$

where $\mathbf{T} \in \mathbb{R}^{2\times 2}$ and $\boldsymbol{a} \in \mathbb{R}^2$.

3.2.3 Regularization With the Restricted Search Space

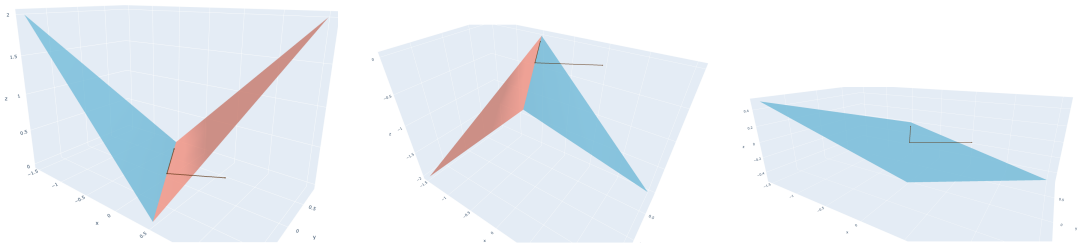
The regularization term in (3.15) can now be further discussed in light of this search space restriction. As the functions in the restricted search space are locally linear inside the triangles, there, the Hessian and the regularization cost are zero; this means that only the junctions between the triangle affine subspaces will have a non-zero Hessian and pay a regularization cost.

As mentioned in subsection 3.1.5, the regularization promotes sparsity of the eigenvalues of the Hessian matrix (if symmetry is assumed), and the eigenvalues give bounds to the second directional derivative of the function ($\nabla^2_{\boldsymbol{v}} f$) in a $\boldsymbol{v}$ direction ($\|\boldsymbol{v}\|_2 = 1$), where the minimum and maximum magnitudes are attained in the direction of the unit norm eigenvectors. Consider figure 3.4, which shows the three types of possible plane junctions. It is easy to see that the second derivative in the direction pointing along the junction of two planes, $\boldsymbol{v}_1$, is always zero ($\nabla^2_{\boldsymbol{v}_1} f = \boldsymbol{v}_1^T \mathbf{H} \boldsymbol{v}_1 = 0$), which is the minimum possible magnitude, so $\boldsymbol{v}_1$ has to correspond to one of the eigenvectors. Since $\boldsymbol{v}_1^T \mathbf{H} \boldsymbol{v}_1 = \lambda_1 \|\boldsymbol{v}_1\|^2 = 0$, then $\lambda_1 = 0$, by positive-definiteness of the norm. We thus expect to have rank-deficient Hessian matrices

along the junctions of the planes. The second eigenvector $\boldsymbol{v}_2$ is perpendicular to this one; in this direction, the magnitude of the second derivative is maximal. Two arrows are shown in figure 3.4 in the direction of each of the eigenvectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$.

Since λ_1 is always zero, the sparsity-promoting effect of the regularization can only be seen through λ_2; if it is positive, then the two contiguous planes form a convex shape ($\nabla^2_{\boldsymbol{v}_2} f > 0$); if it is negative, they form a concave shape; if it zero, then they lie in the same affine subspace, *i.e.*, have the same normal (no "2D knot"). The latter situation will be promoted by the regularization, since it increases sparsity.

Now we have a clearer geometric understanding of the effect of the Hessian-Schatten regularization. The next section will highlight that our restricted search space can be spanned by shifts (in the lattice) of symmetric, hexagonal "box-splines", in analogy to the B-splines in the 1D case.



(a) $\lambda_1 = 0$, $\lambda_2 > 0$. (b) $\lambda_1 = 0$, $\lambda_2 < 0$. (c) $\lambda_1 = 0$, $\lambda_2 = 0$.

Figure 3.4: Possible triangle junctions;
the line segments corresponds to the eigenvectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$

3.2.4 Basis function

To solve the problem, we would like to write any model in our restricted search space as a linear combination of shifted basis functions k:

$$\begin{aligned} f(x) &= \sum_{(i,j)\in\Theta} z_{i,j} k(x - x_{i,j}, y - y_{i,j}) \\ &= \sum_{(i,j)\in\Theta} f(x_{i,j}, y_{i,j}) k(x - x_{i,j}, y - y_{i,j}) \end{aligned} \tag{3.27}$$

Where the set of values $f(\Lambda) = \{f(x_{i,j}, y_{i,j}) : (x_{i,j}, y_{i,j}) \in \Lambda\}$ are our parameters.

It is easy to see that our restricted CPWL space (figure 3.2) can be spanned with the basis functions k shown in figure 3.5, called a box-splines; these are symmetric, finitely supported, piece-wise linear, and interpolatory, since $k_{i,j}$ is 1 for $(x, y) = (x_{i,j}, y_{i,j})$ and 0 for any other vertice point, by analogy with the B1-spline basis. Note that placing the centers of the shifted basis functions in a 2D lattice eliminates the need to discover the "2D knots".

In light of this basis function interpretation, it seems natural that we are able to exactly discretizate of the problem 3.15, and that, with suitable boundary conditions, we can express that problem as:

$$\arg\min_{\mathbf{z}\in\mathbb{R}^N} \left\|\mathbf{H}\mathbf{z} - \boldsymbol{z}^{obs}\right\|_2^2 + \lambda \left\|\mathbf{L}\mathbf{z}\right\|_{\ell_1}, \quad N = P \cdot P = \text{Number of lattice vertices}$$

The goal of the next section is to reach this exact discretization.

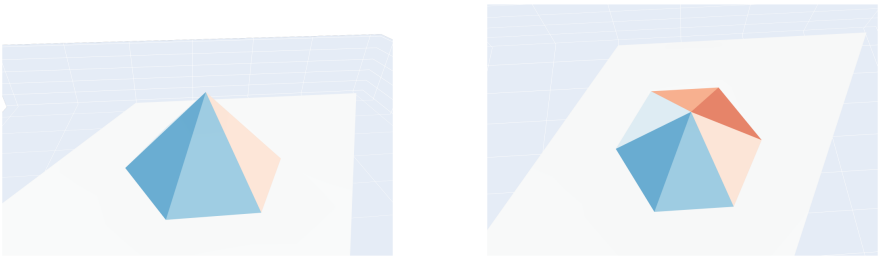


Figure 3.5: Box spline basis function.

3.3 Exact Discretization

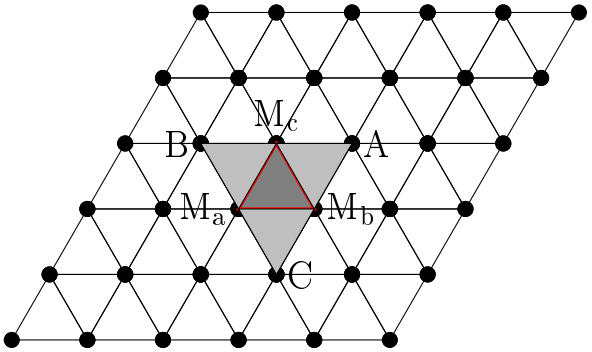


Figure 3.6: Lattice scheme, for $\boldsymbol{\nu}_1 = (1, 0)$ and $\boldsymbol{\nu}_2 = (\frac{1}{2}, \frac{\sqrt{3}}{2})$.

Consider the lattice scheme shown in figure 3.6, with lattice vectors $\boldsymbol{\nu}_1 = (1, 0)$, $\boldsymbol{\nu}_2 = (\frac{1}{2}, \frac{\sqrt{3}}{2})$. Our objective is to write the Hessian-Schatten regularization along the three junctions shown in red as a function of the vertice values (parameters). The result for the remaining junctions is written in the same way. M_a, with lattice coordinates $(k, l) \in \Theta$, will be called the reference vertex, and the list of highlighted vertices (M_a, M_b, M_c, A, B, C), which define the Hessian-Schatten along the three junctions M_bM_c, M_aM_c, M_aM_b, will be called the neighbors of the reference vertex M_a.

Define also $\square_{M_aM_bAM_c}$, $\square_{M_aM_bM_cB}$, and $\square_{CM_bM_cM_a}$, as the open sets formed by the points inside the parallelograms with vertices (M_a, M_b, A, M_c), (M_a, M_b, M_c, B), and (C, M_b, M_c, M_a), respectively.

In the appendix, we show the computations of the Hessian operator along the junction M_bM_c, and refrain from doing so for the other junctions because the mathematical reasoning behind it is the same. Here, we will summarize the Hessian results for this junction and then show the overall Hessian-Schatten results.

Let us start from the following definitions of x_{0y} and y_{0x}:

- for $y : y \in [y_{M_b}, y_{M_c}]$, define $x_{0y} = x : \ (x - x_{M_b}) = -\frac{1}{\sqrt{3}} \cdot (y - y_{M_b})$, such that (x_{0y}, y) represents a point in the line segment M_bM_c.
- for $x : x \in [x_{M_a}, x_{M_b}]$, define $y_{0x} = y : \ (y - y_{M_b}) = -\sqrt{3} \cdot (x - x_{M_b})$, such that for $x \in [x_{M_c}, x_{M_b}]$, (x, y_{0x}) represents a point in the line segment M_bM_c.

And define the vector $\mathbf{z}_{k,l}$ as the vector formed by the values of the neighboring vertices of M_a, with lattice coordinates (k,l):

$$\mathbf{z}_{k,l} = (z_{M_a}, z_{M_b}, z_{M_c}, z_A, z_B, z_C) \tag{3.28}$$

After the derivation (see appendix), the results we get for the Hessian operator along the junction M_bM_c are:

$$\forall (x,y) \in \square_{M_aM_bAM_c} :$$
$$\mathcal{H}f(x,y) = \delta(y - y_{0x}) \cdot (a_2 - m_2) \cdot \begin{bmatrix} 3 & \sqrt{3} \\ \sqrt{3} & 1 \end{bmatrix} = \delta(y - y_{0x}) \cdot \mathbf{K}_1 \ ,$$

$$(\lambda_1)_{\mathbf{K}_1} = 0, \quad (\lambda_2)_{\mathbf{K}_1} = 4 \cdot (a_2 - m_2) \ ,$$

$$(a_2 - m_2) = h \cdot \frac{\sqrt{3}}{3} \cdot \begin{bmatrix} 1 & -1 & -1 & 1 & 0 & 0 \end{bmatrix} \cdot \mathbf{z}_{k,l}$$

we can see that the matrix is symmetric, even though it does not satisfy the (sufficient) Schwartz's theorem condition along the junction. Note that, as expected, the first eigenvalue is zero and the second eigenvalue is a function of the values of the neighboring vertices of M_a.

3.3.1 The Regularization Operator L

Having computed the Hessian along the junctions of the functions in our restricted search space, we now present the final results (exact discretization) of the Hessian-Schatten regularization. Define:

$$\Omega_1 = \{(x,y) : (x,y) \in \square_{M_aM_bAM_c}\}, \quad \Omega_2 = \{(x,y) : (x,y) \in \square_{M_aM_bM_cB}\}$$
$$\Omega_3 = \{(x,y) : (x,y) \in \square_{CM_bM_cM_a}\}, \quad \Omega_{ij} = \Omega_1 \cup \Omega_2 \cup \Omega_3,$$

$$\begin{aligned}
\mathcal{HS}(f)|_{\Omega_{ij}} &= \int_{(x,y)\in\Omega_{ij}} \|\mathcal{H}f(x,y)\|_{S_1} \, dxdy \\
&= \int_{(x,y)\in\Omega_1} \|\mathcal{H}f(x,y)\|_{S_1} \, dxdy + \int_{(x,y)\in\Omega_2} \|\mathcal{H}f(x,y)\|_{S_1} \, dxdy + \int_{(x,y)\in\Omega_3} \|\mathcal{H}f(x,y)\|_{S_1} \, dxdy \\
&= \frac{2\sqrt{3}}{3} \cdot \Big(\ \Big| \begin{bmatrix} 1 & -1 & -1 & 1 & 0 & 0 \end{bmatrix} \cdot \mathbf{z}_{i,j} \Big| + \Big| \begin{bmatrix} -1 & 1 & -1 & 0 & 1 & 0 \end{bmatrix} \cdot \mathbf{z}_{i,j} \Big| \\
&\qquad + \Big| \begin{bmatrix} -1 & -1 & 1 & 0 & 0 & 1 \end{bmatrix} \cdot \mathbf{z}_{i,j} \Big| \ \Big) \\
&= \left\| \frac{2\sqrt{3}}{3} \cdot \begin{bmatrix} 1 & -1 & -1 & 1 & 0 & 0 \\ -1 & 1 & -1 & 0 & 1 & 0 \\ -1 & -1 & 1 & 0 & 0 & 1 \end{bmatrix} \cdot \mathbf{z}_{i,j} \right\|_{\ell_1} = \|\mathbf{L}'\mathbf{z}_{ij}\|_{\ell_1}
\end{aligned} \tag{3.29}$$

To compute the Hessian-Schatten over the whole domain Ω, we simply have to sum

the results of $\|\mathbf{L}'\mathbf{z}_{ij}\|_{\ell_1}$ for all reference points (i,j), where $\mathbf{L}'$ is fixed and $\mathbf{z}_{ij}$ contains the values of the neighboring vertices, which for $(i,j)=(k,l)$ are given in 3.28.

$$\mathcal{HS}(f)|_{\Omega} = \sum_{(i,j)} \mathcal{HS}(f)|_{\Omega_{ij}} = \sum_{(i,j)} \|\mathbf{L}'\mathbf{z}_{ij}\|_{\ell_1} \tag{3.30}$$

Equation (3.30) shows an exact discretization of $\mathcal{HS}(f)$. Aternatively, we can write this in a more concise way:

$$\mathcal{HS}(f)|_{\Omega} = \sum_{(i,j)} \|\mathbf{L}'\mathbf{z}_{ij}\|_{\ell_1} = \|\mathbf{L}\mathbf{z}\|_{\ell_1} \tag{3.31}$$

In order to obtain this concise representation, first, we create a rasterized parameter vector $\mathbf{z}$ of size $N = P \cdot P$, where P is the size of the lattice, containing the values of the function in all the vertices; second, we construct a matrix $\mathbf{L}$ with $3 \cdot N$ rows and N columns: we can think of $\mathbf{L}$ as a vertical stacking of N submatrices, each of size $3 \times N$, with 6 columns corresponding to the columns of $\mathbf{L}'$ and $N-6$ additional zero columns; the indices of the non-zero columns correspond to the indices of the neighbors of a certain vertex, such that the correct neighboring vertices are "selected" from $\mathbf{z}$. We can conclude that $\mathbf{L}$ will be very sparse since, for each row, at most 4 out of N columns will have nonzero values.

To ensure that the function does not pay a regularization cost outside the lattice domain, zero boundary conditions are enforced. This can be done by assuming that the function is zero in the lattice boundary vertices (and outside):

$$z_{i,j} = 0, \quad \text{if } \left(|i| = \frac{P}{2}\right) \vee \left(|j| = \frac{P}{2}\right) \tag{3.32}$$

In practice, we can simply set the columns in $\mathbf{L}$ corresponding to these vertices to zero.

Finally, note how this formula reflects the properties of the Hessian-Schatten. First, we can check that (3.31) does not depend on the lattice spacing h. This reflects the scale-invariance property of the Hessian-Schatten. Second, we can observe that the rows of the matrix $\mathbf{L}'$ (3.29) are all very similar and resemble a finite difference filter. This reflects the rotation and translation-invariance of the Hessian-Schatten. If we look carefully (in conjuction with figure 3.6), we can see that there is a simple algorithm, which is the same for all junctions, to compute the Hessian-Schatten in a given junction; the steps are (take the junction M_bM_c as an example):

1. take the two triangles forming the junction and the four corresponding vertices (M_a, M_b, A, M_c);

2. multiply the values of the vertices in the junction (M_b, M_c) by -1 and sum them;

3. sum the result with the sum of the values of the other two vertices (M_a, A);

4. multiply the result by $\frac{2\sqrt{3}}{3}$ (due to the geometry of the hexagonal lattice).

3.3.2 The Forward Operator H

We will now address the exact discretization of the forward operator, after which the data fidelity term will become:

$$\sum_{m=1}^{M}(f(x_m, y_m) - z_m^{obs})^2 \rightarrow \left\|\mathbf{Hz} - \boldsymbol{z}^{obs}\right\|_2^2 \tag{3.33}$$

For this, the key will be barycentric coordinates (subsection 3.2.2). Recall that we can write any $f(x, y)$ as the convex combination of the values of the vertices to which (x, y) belongs to:

$$\begin{aligned} f(x, y) &= \alpha f(x_1, y_1) + \beta f(x_2, y_2) + \gamma f(x_3, y_3) \\ &= \alpha z_1 + \beta z_2 + \gamma z_3 \end{aligned} \tag{3.34}$$

Let us use the notation $(\hat{x}_m, \hat{y}_m)$ for the lattice coordinates of a point (x_m, y_m) represented in standard coordinates:

$$(x_m, y_m) = \hat{x}_m \boldsymbol{\nu}_1 + \hat{y}_m \boldsymbol{\nu}_2 \tag{3.35}$$

The lattice coordinates of the vertices are integer pairs, by definition.

Our first goal is to assign each (x_m, y_m) to a triangle. Knowing the lattice coordinates $(\hat{x}_m, \hat{y}_m)$ of a point already reduces the search to just two triangles; for example, if the point has lattice coordinates $(\hat{x}_m, \hat{y}_m) = (4.3, 2.5)$, then it belongs to the parallelogram $\{(4, 2), (5, 2), (5, 3), (4, 3)\}$, which contains two triangles: $\{(4, 2), (5, 2), (4, 3)\}$ and $\{(5, 2), (5, 3), (4, 3)\}$. To know exactly which of these is the correct one, we compute the barycentric coordinates of (x_m, y_m) with respect to both triangles and check for which one of them all of the coordinates are non-negative, as mentioned previously.

For this computation, the affine-invariance of barycentric coordinates is useful; the first consequence of this property is that we can compute them from the lattice coordinates of the points, since they are independent of the choice of $\boldsymbol{\nu}_1$ and $\boldsymbol{\nu}_2$, in which case the vertice positions are represented as index pairs. Second, since for large grid sizes P the triangle vertice coordinates might be almost collinear (*e.g.* $\{(232, 234), (233, 234), (234, 233)\}$), which can lead to numerical issues when inverting the matrix in (3.21), we compute instead the barycentric coordinates of $(\lfloor \hat{x}_m \rfloor, \lfloor \hat{y}_m \rfloor)$ with respect to the "origin" triangles $\{(0, 0), (0, 1), (1, 0)\}$ and $\{(0, 1), (1, 1), (1, 0)\}$ (translation-invariance).

Assigning each data point to a triangle and computing the respective barycentric coordinates is cheap and can be done "offline", before solving the minimization problem.

After this step, we can simply use the previous equation (3.34) to compute $f(x_m, y_m)$. This operation can be written concisely as $\mathbf{Hz}$; the matrix $\mathbf{H}$ will have M rows (number of data points) and $N = P \cdot P$ columns (number of vertices). Each row m of $\mathbf{H}$ will have only three non-zero values, the barycentric coordinates, in the indices corresponding to the vertices of the triangle to which it belongs to. Therefore, as is the case for the regularization operator $\mathbf{L}$, $\mathbf{H}$ is very sparse.

For example, suppose our lattice is composed of only two triangles: $\{(0, 0), (1, 0), (0, 1)\}$ and $\{(1, 0), (1, 1), (0, 1)\}$; and our training data consists of two data points: $(x_1, y_1) =$

$(0.1, 0.1)$, belonging to the first triangle, and which has barycentric coordinates $(\alpha_1, \beta_1, \gamma_1)$, and $(x_2, y_2) = (0.9, 0.9)$, belonging to the second triangle, and which has barycentric coordinates $(\alpha_2, \beta_2, \gamma_2)$. Then, the discretization of the forward operator is as follows:

$$\begin{bmatrix} f(x_1, y_1) \\ f(x_2, y_2) \end{bmatrix} = \begin{bmatrix} \alpha_1 & \beta_1 & 0 & \gamma_1 \\ 0 & \alpha_2 & \beta_2 & \gamma_2 \end{bmatrix} \begin{bmatrix} z_{0,0} \\ z_{1,0} \\ z_{1,1} \\ z_{0,1} \end{bmatrix} = \mathbf{Hz} \tag{3.36}$$

Having defined the discrete forward and regularization operators, we finally reach the exact, finite-dimensional, discretization of our continuous problem:

$$\arg\min_{\mathbf{z}\in\mathbb{R}^N} \left\|\mathbf{Hz} - \boldsymbol{z}^{obs}\right\|_2^2 + \lambda \left\|\mathbf{Lz}\right\|_{\ell_1}, \quad N = \text{Number of lattice vertices} \tag{3.37}$$

Note that the problem became of the same form as (1.38), from the B-splines method. The algorithm used to solve it will be discussed next.

3.4 Algorithm

To solve (3.37), we will use an ADMM-simplex algorithm, based on [8] and [9]. We will first discuss it in the context of the gTV B-splines method.

Theorem 2 in [9] states the solution set S_B of the B-spline optimization problem (1.38) is a compact convex set whose extreme points $\mathbf{c}^*$ are sparse, i.e., $\|\mathbf{Lc}^*\|_0 \leq M - N_0$, where M is the number of datapoints and N_0 is the dimension of the null space of the operator L. The first step of the algorithm uses the alternating direction method of multipliers (ADMM) to solve (1.38), reaching a solution $\mathbf{c}_{\text{ADMM}}$. This solution is not guaranteed to be sparse, i.e., have low $\|\mathbf{Lc}_{\text{ADMM}}\|_0$, since different ADMM solutions with the same changes in slope ($\|\mathbf{Lc}_{\text{ADMM}}\|_1$) can both be minimizers (this will be made clearer in the experimental part). To circumvent this issue, *Lemma 1* states that all solutions in the set S_B have the same measurements $\mathbf{Hc} = \boldsymbol{y}_\lambda$. So, after the ADMM, assuming it converged to a solution, we can compute $\boldsymbol{y}_\lambda = \mathbf{Hc}_{\text{ADMM}}$ and recast the problem (1.38) as:

$$\arg\min_{\mathbf{c}\in\mathbb{R}^N} \left\|\mathbf{Lc}\right\|_{\ell_1}, \quad \text{s.t. } \mathbf{Hc} = \boldsymbol{y}_\lambda \tag{3.38}$$

The problem above can be transformed into a linear program (see [9] for more details). Then, we can use the simplex algorithm to solve it, which is known to converge to the extreme points of the solution set. By *Theorem 2* in [9], the extreme points are sparse solutions. Note that (3.38) will not lower the value of $\|\mathbf{Lc}\|_{\ell_1}$ if the ADMM reached an optimal solution to problem (1.38) - by contradiction, if the simplex finds a solution $\mathbf{c}_{\text{simplex}}$ with lower $\|\mathbf{Lc}\|_{\ell_1}$, while keeping the same measurements $\mathbf{Hc}_{\text{simplex}} = \boldsymbol{y}_\lambda$, then this solution achieves a lower loss than $\mathbf{c}_{\text{ADMM}}$, hence the ADMM did not reach an optimal solution. However, it might find a new solution which has the same measurements and regularization loss, but is sparser than $\mathbf{c}_{\text{ADMM}}$.

In summary, the job of the ADMM is to find a solution to the problem and the job of the simplex is to find a (hopefully) sparser solution, which has the same measurements as the ADMM solution.

To solve our problem (3.37), we will use the same ADMM-simplex mixed algorithm, since it has the same form as (1.38).

Regarding the implementation: the code is writen in Python (available in [1]). The ADMM uses the odl library [2] while the simplex uses the cvxopt library [3]; sparse representations are used for the operators $\mathbf{L}$ and $\mathbf{H}$ (scipy.sparse [4], cvxopt.spmatrix). The values of the grid vertices $\mathbf{z}$ (parameters) are zero-initialized before the algorithm.

In the next chapter we will validate the theoretical findings with experiments.

[1] https://c4science.ch/diffusion/9422/

[2] https://odlgroup.github.io/odl/

[3] https://cvxopt.org

[4] https://docs.scipy.org/doc/scipy/reference/sparse.html

Chapter 4

Hessian-Schatten Experiments

The first section of the experiments deals with a simple problem of learning a pyramid function. Its focus is in accessing the two following points:

1. the solution reached by ADMM is not guaranteed to be sparse (have low $\|\mathbf{L}\mathbf{z}_{\text{ADMM}}\|_0$), since different ADMM solutions with the same changes in slope $\|\mathbf{L}\mathbf{z}_{\text{ADMM}}\|_{\ell_1}$ can both be minimizers;

2. the simplex reaches sparse solutions $\mathbf{z}_{\text{simplex}}$. Hopefully, we will be able to reconstruct the pyramid, which is the sparsest solution possible.

In the second section, we will attempt to reconstruct a function with large planes from a few measurements, and compare the result with ReLU networks. Both our method and the ReLU networks learn CPWL functions but, in our case, we promote sparsity with the regularization term.

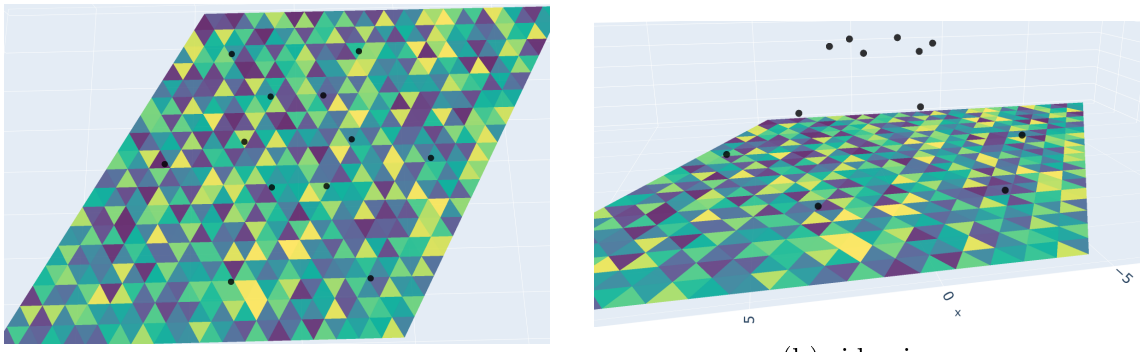
The following section tackles the problem of data fitting, $z_m^{obs} = f(x_m, y_m) + \epsilon$. Here, we will see that the sparse CPWL models that our method constructs are much better adapted for approximating smooth images. ReLU networks are very good at learning from high-dimensional signals (complete images), but produce much more arbitrary results when learning from 2D inputs.

Finally, we will see the advantages of learning over function spaces in a problem of "2D super-resolution", where we only have access to a downscaled version of an image and attempt to reconstruct the original image. Again, we will compare our method with ReLU networks.

4.1 Incomplete Pyramid

In this "incomplete pyramid" experiment, our data consists of $M = 12$ samples from a pyramid function, which are positioned in the lattice vertices; the dataset is shown in figure 4.1, together with the zero-initialized lattice of size $P = 18$ (total of 324 parameters).

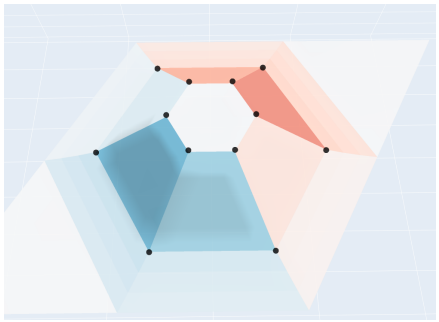
We run the algorithm for several different regularization weights λ and numbers of ADMM iterations. The learned models, after the ADMM and simplex steps, can be seen in figures 4.2, 4.3, and 4.4; the colors of the triangles are plotted according to their normal vectors, so that sparsity can be visually identified. Table 4.1 summarizes the numerical results: the two "sparsity" columns show the number (and percentage) of triangle junctions for which $|\lambda_2| \geq \epsilon$ (recall that $\lambda_1 = 0$), with



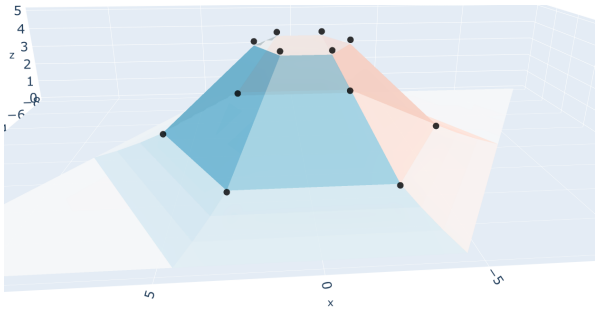
(a) up view.

(b) side view.

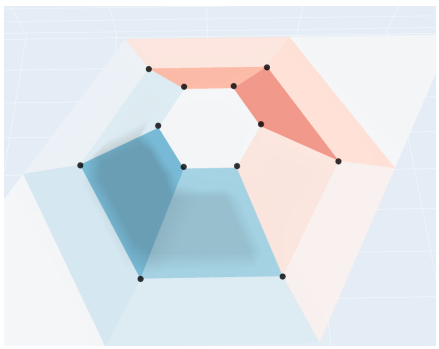
Figure 4.1: Incomplete pyramid dataset (black dots) and the lattice triangulation with zero-initialized vertices.



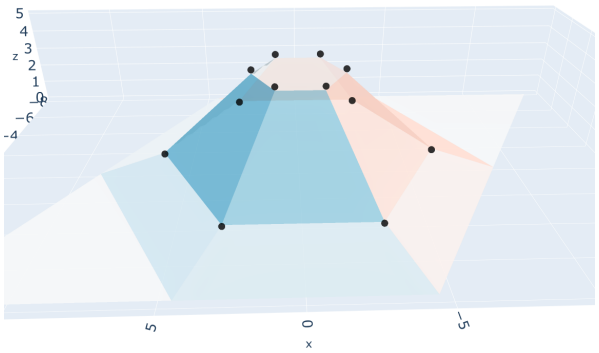
(a) ADMM up view.



(b) ADMM side view.



(c) Simplex up view.



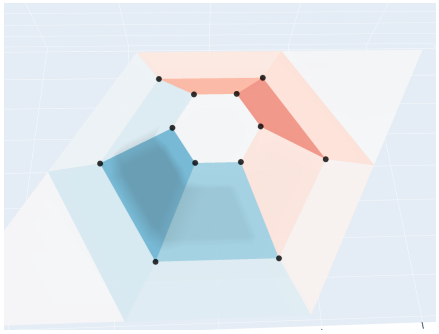
(d) Simplex side view.

Figure 4.2: ADMM-simplex with 50.000 ADMM iterations, $\lambda = 10^{-1}$.

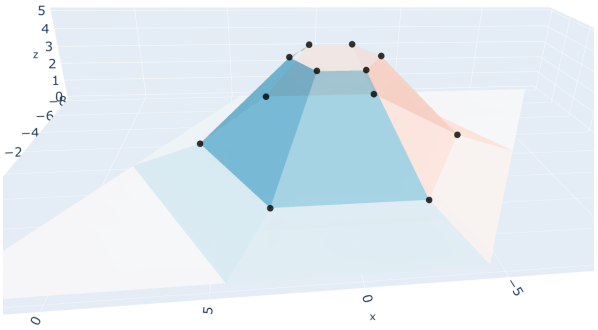
$\epsilon = 10^{-5}$, after the ADMM and simplex steps, respectively (the lower the value, the sparser the solution). We will now analyse the results, one by one.

Model	ADMM sparsity	Simplex sparsity
$\lambda = 10^{-1}$, 50.000 iter	234 (23.31%)	150 (14.94%)
$\lambda = 10^{-2}$, 100.000 iter	234 (23.31%)	132 (13.15%)
$\lambda = 10^{-2}$, 50.000 iter	234 (23.31%)	144 (14.34%)

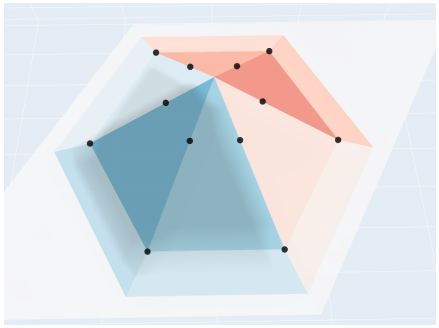
Table 4.1: Incomplete pyramid results.



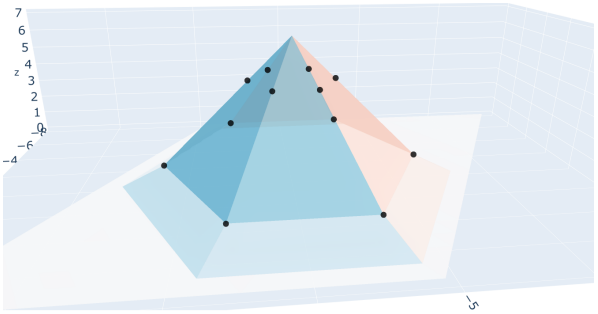
(a) ADMM up view.



(b) ADMM side view.

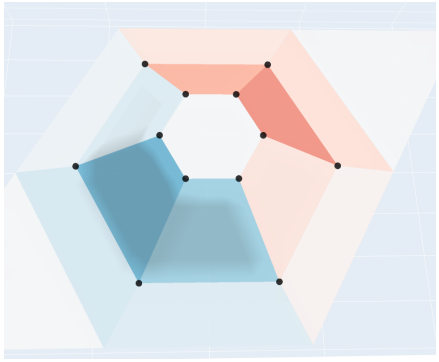


(c) Simplex up view.

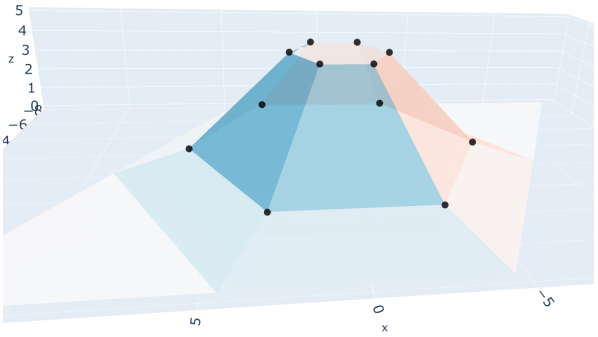


(d) Simplex up view.

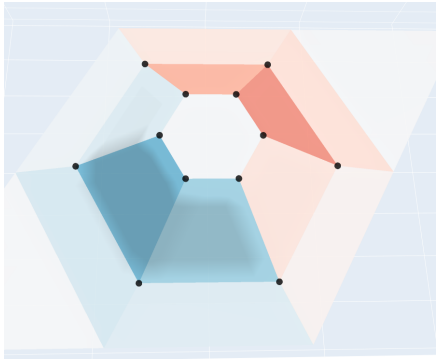
Figure 4.3: ADMM-simplex with 100.000 ADMM iterations, $\lambda = 10^{-2}$.



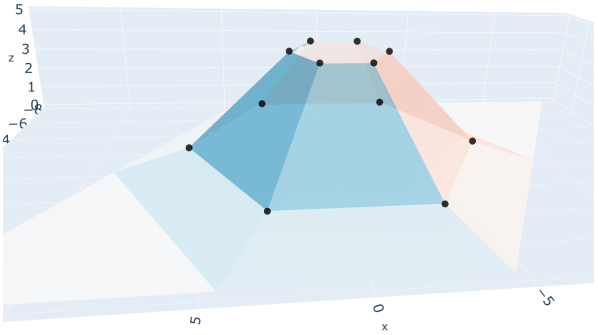
(a) ADMM up view.



(b) ADMM side view.



(c) Simplex up view.



(d) Simplex up view.

Figure 4.4: ADMM-simplex with 50.000 ADMM iterations, $\lambda = 10^{-2}$.

In the first learned model, 4.2, we can see that the ADMM reached a very good solution. Starting from the lattice grid shown in figure 4.1, it lead to a reduced number of facets, with only 23% of the "2D knots" being nonzero (table 4.1). However, the base of the exterior hexagonal polygon is not sparse, even though no datapoint is in there - notice the shades of blue color, which indicate that there are different facets -. The simplex corrects this behaviour, by making only 15% of the junctions non-zero.

In the second model, 4.3, the ADMM does not present the previous behaviour, and visually seems to converge to the same solution as the simplex in the previous example. However, the simplex was able to reach an even sparser solution (very close to the pyramid), with only 13% of the 2D knots being nonzero.

Finally, in the last example, nothing appears to visually change. However, the numbers in table 4.1 tell us something different: there were several "almost zero" junctions in the ADMM solution, where the normals of contiguous triangles are very close to each other. The simplex effectively eliminated these cases, given that the number of non-zero junctions (144) actually corresponds to what is shown visually.

We conclude by going back to our two initial points. First, the ADMM did not guarantee sparse solutions. This was especially visible in the last example where the ADMM solution looked visually the same as the simplex but the simplex had 40% less 2D knots, which means that the ADMM had several "almost zero" junctions which it was not able to eliminate. Second, we saw how the simplex reached very sparse solutions and, even though we didn't get the perfect pyramid, we got very close to it.

4.2 Incomplete Planes

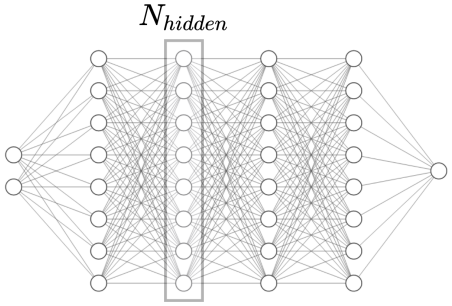
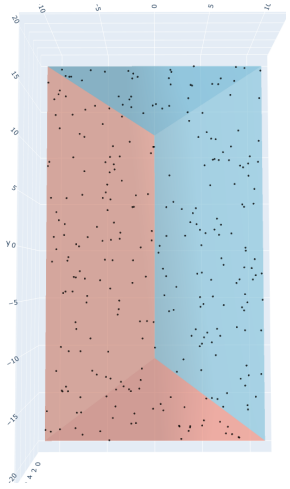


Figure 4.5: ReLU network architecture.

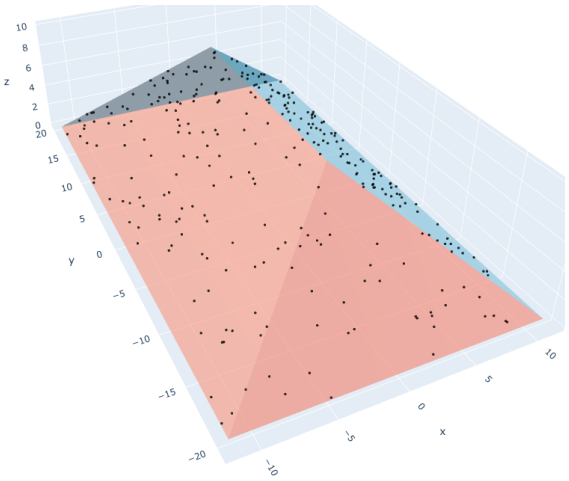
Model	MSE	Simplex sparsity
Hessian-Schatten, $\lambda = 10^{-2}$	0.0092	23.13%
ReLU network, $\mu = 10^{-4}$	0.0441	*NA*

Table 4.2: Results for the second planes experiment.

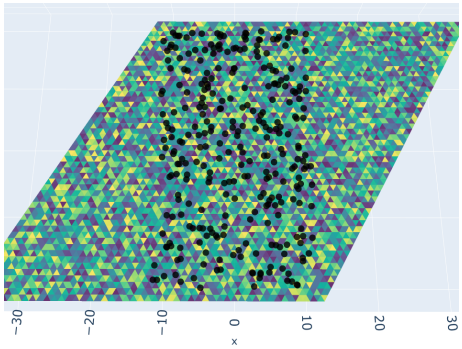
In this section, we wish to compare our algorithm with ReLU networks (which also produce CPWL functions) in the task of reconstructing a structure with large planes. Here, we will see the advantages of having a sparsity-promoting framework.



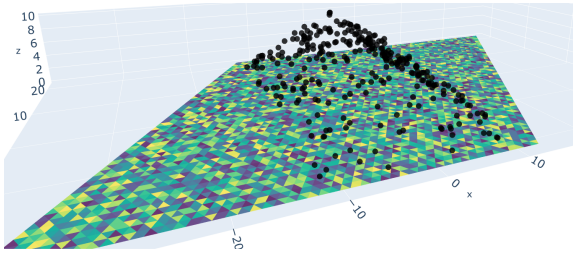
(a) Large planes data, up view.



(b) Large planes data, side view.



(c) Large planes data with lattice triangulation, up view.



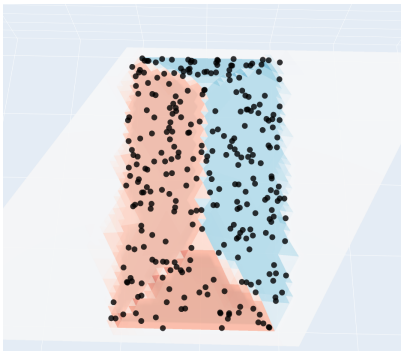
(d) Large planes data with lattice triangulation, side view.

Figure 4.6: Large planes data.

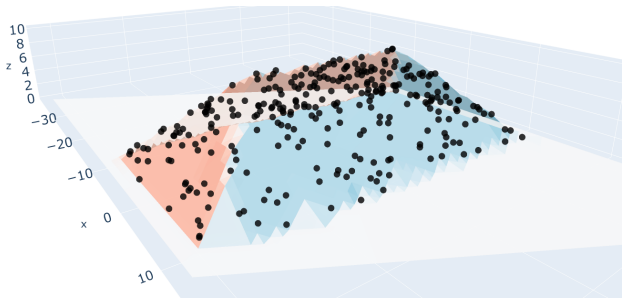
In this experiment, the test set consists of a very fine sampling of the underlying function and the training set corresponds to 2% of the test samples, with these samples being randomly chosen, as shown in figure 4.6.

For the Hessian-Schatten algorithm, we use a lattice of size $P = 50$ and run the ADMM for 50.000 iterations, with a regularization weight $\lambda = 10^{-2}$, before performing the simplex. The ReLU network architecture is shown in figure 4.5; we set $N_{hidden} = 100$, resulting in a total of 30701 parameters. The network is trained with a mean-squared-error (MSE) loss, so that the data fidelity term is of the same form as the Hessian-Schatten and set the weight decay to $\mu = 10^{-2}$. The training is ran for 250 epochs with an Adam optimizer [27] and batch size 64. The inital learning rate is 10^{-3} and is decreased by 10 at epochs 175 and 225.

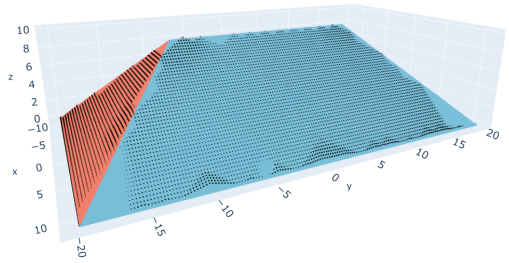
The test MSE and model sparsity are reported in table 4.2 and the visual results can be seen in figure 4.7. The first row shows the Hessian-Schatten model, after the simplex, where the colors of the triangles are again plotted according to the normals, so that sparsity can be assessed. The following two rows show the evaluation of the model on the test set (black dots), together with the ground truth, first for the Hessian-Schatten (second row) and then for the neural network (third row). For each method, the plot on the left shows an opaque ground truth, and the plot on



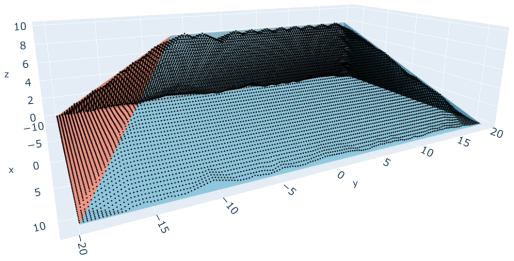
(a) Hessian-Schatten, up view.



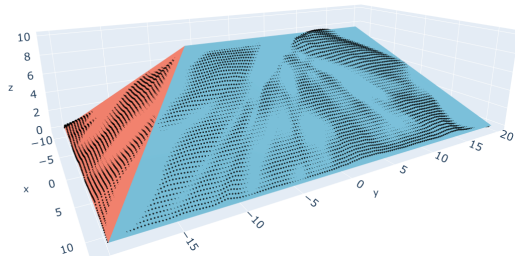
(b) Hessian-Schatten, side view.



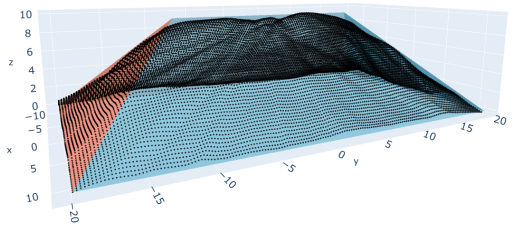
(c) Hessian-Schatten, opaque gtruth, side view.



(d) Hessian-Schatten, translucid gtruth, side view.



(e) ReLU network, opaque gtruth, side view.



(f) ReLU network, translucid gtruth, side view.

Figure 4.7: Large planes reconstruction results.

the right shows a translucid ground truth. Both are shown so that it is clearer to see that the Hessian-Schatten produces sparser and less "chaotic" solutions.

We can observe that the Hessian-Schatten created a much more smooth (and sparse) signal than the ReLU network, with only 23.13% of the 2D knots being nonzero. It also lead to a much lower test MSE (0.0092 vs 0.0441). This experiment highlights the advantage of a sparsity-promoting regularization in modeling CPWL functions.

4.3 Data Fitting

In this experiment, we address the problem of data fitting, exemplified in figure 1.1 from the introduction. We assume our dataset consists of noisy observations from an underlying function f:

$$z_m^{obs} = f(x_m, y_m) + \epsilon \tag{4.1}$$

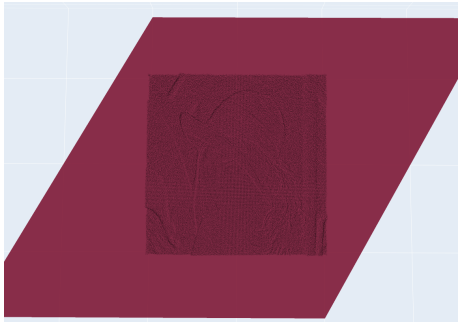


Figure 4.8: Lenna in lattice.

where $\epsilon \sim \mathcal{N}(\mu, \sigma^2)$. Our goal is to model the underlying function f.

We use the classic example of a grayscale, 512×512, Lenna image, with pixel values $\{z_{i,j}^{obs}\}_{i,j=1,\dots,512}$, which are scaled to the range $[0, 1]$. We assign each (i, j) pixel in the image to a position $(x_{i,j}, y_{i,j})$, such that the value of each pixel corresponds to sampling the function f in a grid, *i.e.*, $f(x_{i,j}, y_{i,j}) = z_{i,j}$.

The noisy image is created by adding a zero-mean gaussian noise ($\sigma = 0.1$) to the unit-range-scaled original image. The original and noisy images are shown in figure 4.9a, 4.9b. The training set corresponds to the noisy image ($M = 512 \cdot 512$ observations) and the test set to the original image. Our interest in this experiment is to show how our 2D CPWL parameterization is more fit to model smooth signals like images than a ReLU network, and that the regularization is well-suited to fight overfitting, through its sparsifying effect.

In our algorithm, we fit the square image in the lattice as shown in figure 4.8, and set $P = 820$ such that, in the region of the data, there are slightly more parameters than the number of datapoints. In this case, we should ensure that the parameters outside of the data region are not taken into account and do not increase the computational cost of the algorithm. This time, due to the extremely high number of parameters ($P \cdot P = 672.400$), which largely exceeds the limits of linear programming, simplex was not performed. The ADMM algorithm is ran for 50.000 iterations.

The ReLU network is the same as in the previous section; we grid-search the weight decay (μ) in [1e8, 1e−2] and choose the model which produces the lowest test MSE.

The results are shown in figure 4.9. We can observe that the Hessian-Schatten results are much better then those of the ReLU network. Neural networks perform well when they have access to the full image as input, thus being able to exploit its spatial structure. However, taking only a 2D value as input, they perform poorly. Observe how sparsity increases when we increase the regularization weight: from 19.41% to 14.90%, where the nonzero slopes are those such that $|\lambda_2| > 10^{-4}$. Moreover, as we increase λ, the images get smoother. The hyper-parameter λ provides a bias-variance tradeoff; if it is very small, the model will fit the noise and peform poorly; if it is very large, we restrict the model too much, such that it cannot approximate f sufficiently well.

4.4 2D Super-Resolution

Super-resolution is the task of increasing the resolution of an image. Usually, an input image $\boldsymbol{x} \in \mathbb{R}^{N \times M}$ is given as input to the algorithm and its job is to output an image of k times greater resolution $\boldsymbol{y} \in \mathbb{R}^{k \cdot N \times k \cdot M}$, where k is the expansion factor. In our case, we learn over function spaces, modeling $f : \mathbb{R}^2 \mapsto \mathbb{R}$ instead of taking a discrete $\boldsymbol{x} \in \mathbb{R}^{N \times M}$ input. This continuous domain treatment is in line with the observation that images are samples of a continuous and usually smooth signal, which we are trying to model. What we want to show here is that modelling the continuous signal f provides much greater flexibility since we can then sample it as finely as we want, knowing the value $f(x, y)$ for each position (x, y). This allows us to do super-resolution in a natural way, through learning the model f on a downscaled version (training data) of an image and then sample more finely (depending on k) to get a greater resolution image which can be compared to the original one (test data).

In these experiments, we again use the Lenna 512×512 image and downsample it by a factor of 4 to a size 128×128. The training data consists of the downscaled image samples and the test data of the original image. They are shown in figure 4.10a, 4.10b. The downsampled image was zoomed in to occupy the same space as the original one.

In the same way as the previous section, we assign each pixel (i, j) in our training data to a real-valued position $(x_{i,j}, y_{i,j})$ lying in a grid (see 4.8), and normalize the pixel values to $[0, 1]$. For testing, we sample the values in a new grid, occupying the same space as the training grid, but which is 4 times finer.

The grid size $P = 820$ is chosen such that, in the data region, there are roughly the same number of parameters as pixels in the original image (and 4 times more parameters than pixels in the subsampled image). The ADMM algorithm was ran for 50.000 iterations. Again, due to the large number of parameters $(820 \cdot 820)$, simplex is not performed.

The neural network and the training details are the same as in the previous section, and the model with the lowest test MSE for the neural network (after grid-searching μ) is chosen.

The results are shown in figure 4.10 - here, it is useful to zoom in the pictures, for better visualization -. Again, the ReLU network performs much worse than the Hessian-Schatten, since it cannot take well into account the structure of the signal. For the Hessian-Schatten, we show the results for the λ which gave the lowest test MSE, after grid-searching λ, as well as for three higher regularization weights. Notice that, as λ increases, the images get smoother given that the algorithm favors smaller signal variations, and the number of non-zero junctions is increased: from 19.46%, 15.63% and 13.46% (percentage of junctions with $|\lambda_2| > 10^{-4}$).

In this experiment, we see the benefit of learning over function spaces: we have much more flexibility since we can sample the model as finely as we want. Of course, we are restricted by the number of parameters, so the result is naturally less precise the finer you sample. Note also that the piece-wise linear assumption is more suited for images than the implicit piece-wise constant assumption which discretizing the Hessian, as in [10], entails.

(a) 512×512 grayscale Lenna. (Original)

(b) Lenna + gaussian noise, SNR = 14.49 dB.

(c) Hessian-Schatten, $\lambda = 10^{-1}$, ADMM sparsity: 19.41%, PSNR = 32.93 dB.

(d) Neural network, $\mu = 10^{-3}$, PSNR = 31.45 dB

.

(e) Hessian-Schatten, $\lambda = 2 \cdot 10^{-1}$, ADMM sparsity: 14.90%, PSNR = 32.82 dB.

Figure 4.9: Data fitting results.

(a) 512×512 grayscale Lenna.

(b) Downsampled $\times 4$ Lenna.

(c) Neural network, $\mu = 10^{-4}$,
PSNR = 31.09 dB.

(d) Hessian-Schatten, $\lambda = 10^{-3}$,
ADMM sparsity: 19.46%,
PSNR = 33.59 dB.

(e) Hessian-Schatten, $\lambda = 5 \cdot 10^{-3}$,
ADMM sparsity: 15.63%,
PSNR = 33.47 dB.

(f) Hessian-Schatten, $\lambda = 10^{-2}$,
ADMM sparsity: 13.46%,
PSNR = 33.07 dB.

Figure 4.10: 2D Super-resolution results.

Chapter 5

Conclusions and Future Work

In this chapter, we summarize the main conclusions of this thesis and refer to possible future work.

This project explored higher-order regularization methods whose sparsifying nature allowed us to have parametric solutions to continuous domain problems and to grid the search space, leading to simpler and more interpretable results. It also reinforced the use of splines (in particular, B1-splines and box-splines) in connecting the continuous and discrete worlds of signal processing.

In the first part, we discussed deep splines, a method to learn the activations of a neural network. This method enabled the use of smaller networks to reach the same level of performance as ReLU-based networks, with a reduced total number of parameters. This indicates that the transfer of capacity from the network weights to the activations is advantageous. The visualization of the learned activations also emerges as a new way of gaining insight into the inner workings of deep spline networks and possibly infer information about the learning task, as we saw from the parabola-shaped activation in the area classification problem.

In the second part, we developed a novel learning framework based on the Hessian-Schatten regularization. Restricting the search space to CPWL functions with 2D knots on a lattice grid lead to an exact discretization of the continuous domain problem and a simple expression for the Hessian-Schatten. Moreover, it provided a box-spline basis intepretation for our model space. Addressing the problem in its continuous formulation successfully avoided the discretization of the Hessian operator with second finite differences, which contains an implicit and unmotivated piece-wise constant assumption about the signal and introduces discretization errors.

A crucial component of this regularization framework is its sparsity-promoting effect, which can be visually interpreted as reducing the number of non-zero junctions (2D knots) of the CPWL model.

We also presented this method as an alternative to ReLU-based networks in modeling CPWL functions. Neural networks are known to perform well when learning from high-dimensional signals, but our framework was shown to be better suited for smooth CPWL 2D functions. In addition, the complex parameterization of neural networks does not provide a clear understanding of how the parameters affect the overall model function, which our framework does.

In the experimental section, we also highlighted the role of the Hessian-Schatten regularization in preventing the model from fitting the noise present in image samples, and showed how learning over function spaces leads to greater flexibility, al-

lowing us to sample the signal at any location.

Table 5.1 summarizes the methods discussed in this thesis. Neural networks solve parametric problems and the gTV B-spline and Hessian-Schatten methods learn over function spaces. In the deep splines case, since regularization is applied to both the weights of the neural network and its activation functions, it can be viewed as middle ground between neural networks and the gTV B-splines method.

	Problem formulation
gTV B-splines	$\arg\min_{f} \|f(\boldsymbol{x}) - \boldsymbol{y}\|_2^2 + \lambda \|\mathrm{L}\{f\}\|_{\mathcal{M}}$
Hessian-Schatten	$\arg\min_{f} \|f(\boldsymbol{x}) - \boldsymbol{y}\|_2^2 + \lambda \mathcal{HS}(f)$
Neural Networks	$\arg\min_{\boldsymbol{\theta_w}} \|f(\boldsymbol{x}, \boldsymbol{\theta_w}) - \boldsymbol{y}\|_2^2 + \mu R(\boldsymbol{\theta_w})$
Deepsplines	$\arg\min_{\boldsymbol{\theta_w}, \boldsymbol{\sigma}} \|f(\boldsymbol{x}, \boldsymbol{\theta_w}, \boldsymbol{\sigma}) - \boldsymbol{y}\|_2^2 + \mu R(\boldsymbol{\theta_w}) + \lambda R_{\mathrm{TV}^{(2)}}(\boldsymbol{\sigma})$

Table 5.1: Summary of methods.

In the future, there are several possible research directions which are worth exploring.

First, we would benefit from reducing the complexity of the algorithm, whose bottleneck is the simplex. Reducing the dimensionality of the simplex or finding an alternative sparsification method is an important step in expanding the domain of application of the Hessian-Schatten which, due to its two-dimensional nature, can quickly reach the limits of linear programming.

Second, we would like to compare the Hessian-Schatten with deep splines, verifying if the deep spline network could better overcome the difficulties faced by the ReLU-network in learning images in 2D.

Third, we could explore the problem of finding the knot locations by learning on an adaptive grid, which relates to the idea of bit-rate allocation for images.

Finally, we can consider using the Hessian-Schatten as a module in a neural network, replacing the deep splines, by pairing the neurons in a layer so as to have 2D inputs to the activation.

Chapter 6

Appendix

6.1 Hessian Operator along M_bM_c

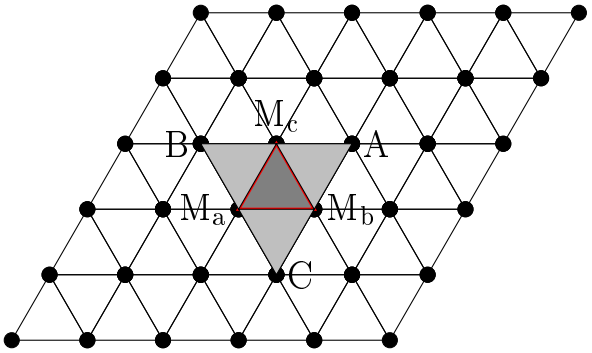


Figure 6.1: Lattice scheme, for $\boldsymbol{\nu}_1 = (1, 0)$ and $\boldsymbol{\nu}_2 = (\frac{1}{2}, \frac{\sqrt{3}}{2})$.

Let $\mathbf{X} \in \mathbb{R}^{2\times 2}$ be formed by the lattice vectors:

$$\mathbf{X} = \begin{bmatrix} | & | \\ \boldsymbol{\nu}_1 & \boldsymbol{\nu}_2 \\ | & | \end{bmatrix} = \frac{1}{h} \cdot \begin{bmatrix} 1 & \frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} \end{bmatrix} \tag{6.1}$$

where h is the grid spacing. $\mathbf{X}^{-1}$ is given by:

$$\mathbf{X}^{-1} = h \cdot \begin{bmatrix} 1 & -\frac{\sqrt{3}}{3} \\ 0 & \frac{2\sqrt{3}}{3} \end{bmatrix} \tag{6.2}$$

We use the following notations for the plane equations of each of the triangles shown in 6.1:

$$\begin{aligned} &(x, y) \mapsto m_1 x + m_2 y + m_3, && \text{for } (x, y) \in \Delta_{M_a M_b M_c} \\ &(x, y) \mapsto a_1 x + a_2 y + a_3, && \text{for } (x, y) \in \Delta_{M_b A M_c} \end{aligned} \tag{6.3}$$

Each Δ_{ijk} is the closed set formed by the points in the triangle with vertices i, j, k. We start with by defining x_{0y} and y_{0x}:

- for $y : y \in [y_{M_b}, y_{M_c}]$, define $x_{0y} = x : \ (x - x_{M_b}) = -\frac{1}{\sqrt{3}} \cdot (y - y_{M_b})$, such that (x_{0y}, y) represents a point in the line segment M_bM_c.

- for $x : x \in [x_{M_a}, x_{M_b}]$, define $y_{0x} = y : \ (y - y_{M_b}) = -\sqrt{3} \cdot (x - x_{M_b})$, such that for $x \in [x_{M_c}, x_{M_b}]$, (x, y_{0x}) represents a point in the line segment $M_b M_c$.

Define also $\square_{M_a M_b A M_c}$ as the open set formed by the the points inside the parallelogram with vertices M_a, M_b, A, M_c. Then, we have:

$\forall (x,y) \in \square_{M_a M_b A M_c}$,

$$\frac{\partial^2 f(x,y)}{\partial x^2} = (a_1 - m_1) \cdot \delta(x - x_{0y}), \quad \frac{\partial^2 f(x,y)}{\partial x \partial y} = (a_2 - m_2) \cdot \delta(x - x_{0y}),$$

$$\frac{\partial^2 f(x,y)}{\partial y \partial x} = (a_1 - m_1) \cdot \delta(y - y_{0x}), \quad \frac{\partial^2 f(x,y)}{\partial y^2} = (a_2 - m_2) \cdot \delta(y - y_{0x}).$$

We can establish a relationship between $(a_1 - m_1)$ and $(a_2 - m_2)$, using boundary conditions in the junction $M_b M_c$ connecting the planes:

$\forall \ y$ such that $y \in [y_{M_b}, y_{M_c}]$,

$$m_1 x_{0y} + m_2 y + m_3 = a_1 x_{0y} + a_2 y + a_3 \iff (a_1 - m_1) x_{0y} + (a_2 - m_2) y = a_3 - m_3$$

$$\begin{cases} (a_1 - m_1) x_{M_b} + (a_2 - m_2) y_{M_b} = a_3 - m_3 \\ (a_1 - m_1) x_{M_c} + (a_2 - m_2) y_{M_c} = a_3 - m_3 \end{cases}$$

Subtracting the two equations,

$$(a_1 - m_1)(x_{M_b} - x_{M_c}) + (a_2 - m_2)(y_{M_b} - y_{M_c}) = 0$$

$$\iff (a_1 - m_1) = (a_2 - m_2) \cdot \frac{(y_{M_c} - y_{M_b})}{(x_{M_b} - x_{M_c})} = (a_2 - m_2) \cdot tan\left(\frac{\pi}{3}\right) = (a_2 - m_2) \cdot \sqrt{3}$$

where the value $\frac{\pi}{3}$ reflects the geometry of the regular hexagon. So, the Hessian $\forall (x,y) \in \square_{M_a M_b A M_c}$ is:

$$\mathcal{H} f(x,y) = (a_2 - m_2) \cdot \begin{bmatrix} \sqrt{3} \cdot \delta(x - x_{0y}) & \delta(x - x_{0y}) \\ \sqrt{3} \cdot \delta(y - y_{0x}) & \delta(y - y_{0x}) \end{bmatrix} \tag{6.4}$$

This matrix does not seem to be symmetric, however, this is actually the case. First notice that the 1D diracs in the two directions are related:

$$\frac{|y - y_{0x}|}{|x - x_{0y}|} = tan\left(\frac{\pi}{3}\right) = \sqrt{3} \iff |x - x_{0y}| = |y - y_{0x}| \cdot \frac{1}{\sqrt{3}}$$

Using the following identity:

$$\delta(\alpha x) = \frac{\delta(x)}{|\alpha|} \tag{6.5}$$

and the fact that the dirac does not depend on the sign of the argument due to symmetry ($\delta(-x) = \delta(x)$), which also follows from the identity, we get:

$$\delta(x - x_{0y}) = \sqrt{3} \cdot \delta(y - y_{0x}) \tag{6.6}$$

Finally, we obtain a symmetric matrix, even though the function does not satisfy the Schwart'z theorem condition (continuity of the second derivative) along M_bM_c:

$$\forall\ (x,y) \in \square_{M_aM_bAM_c}:$$

$$\mathcal{H}f(x,y) = (a_2 - m_2)\begin{bmatrix} 3\cdot\delta(y-y_{0x}) & \sqrt{3}\cdot\delta(y-y_{0x}) \\ \sqrt{3}\cdot\delta(y-y_{0x}) & \delta(y-y_{0x}) \end{bmatrix}$$

$$= \delta(y-y_{0x})\cdot(a_2-m_2)\cdot\begin{bmatrix} 3 & \sqrt{3} \\ \sqrt{3} & 1 \end{bmatrix} = \delta(y-y_{0x})\cdot\mathbf{K}_1$$

where $\mathbf{K}_1$ is:

$$\mathbf{K}_1 = (a_2 - m_2)\cdot\begin{bmatrix} 3 & \sqrt{3} \\ \sqrt{3} & 1 \end{bmatrix} \tag{6.7}$$

Now, we are ready to compute the eigenvalues of the hessian, $\boldsymbol{\lambda}(\mathcal{H}f(x,y))$. The details are involved and beyond the scope of the thesis (require functional analysis), but it turns out that:

$$\boldsymbol{\lambda}(\delta(y-y_{0x})\cdot\mathbf{K}_1) = \delta(y-y_{0x})\boldsymbol{\lambda}(\mathbf{K}_1) \tag{6.8}$$

So, we can simply "take the dirac out" and compute the eigenvalues of the matrix $\mathbf{K}_1$. As we expected, the matrix has rank 1 (if $a_2 \neq m_2$, otherwise the rank is 0), thus a zero eigenvalue, since $\left[\frac{1}{2}, -\frac{\sqrt{3}}{2}\right] \in \mathcal{N}_{\mathbf{K}_1}$. Then, the second eigenvalue is equal to the trace of the matrix:

$$(\lambda_1)_{\mathbf{K}_1} = 0, \quad (\lambda_2)_{\mathbf{K}_1} = \mathrm{tr}(\mathbf{K}_1) = (a_2 - m_2)\cdot 4$$

Finally, remembering our initial objective (3.27), we want to write $(a_2 - m_2)$ as a function of the parameters:

$$\mathbf{z}_{k,l} = (z_{M_a}, z_{M_b}, z_{M_c}, z_A, z_B, z_C) \tag{6.9}$$

where $z_v = f(x_v, y_v)$, for $v \in (M_a, M_b, M_c, A, B, C)$, and (k,l) are the lattice coordinates of the reference point M_a. For $(x,y) \in \Delta_{M_aM_bM_c}$, we have:

$$(z - z_{M_a}) = m_1\cdot(x - x_{M_a}) + m_2\cdot(y - y_{M_a}),$$

$$\begin{bmatrix} z_{M_b} - z_{M_a} \\ z_{M_c} - z_{M_a} \end{bmatrix} = \begin{bmatrix} x_{M_b} - x_{M_a} & y_{M_b} - y_{M_a} \\ x_{M_c} - x_{M_a} & y_{M_c} - y_{M_a} \end{bmatrix}\cdot\begin{bmatrix} m_1 \\ m_2 \end{bmatrix} = X^T\cdot\begin{bmatrix} m_1 \\ m_2 \end{bmatrix}$$

$$\iff \begin{bmatrix} m_1 \\ m_2 \end{bmatrix} = (X^{-1})^T\cdot\begin{bmatrix} z_{M_b} - z_{M_a} \\ z_{M_c} - z_{M_a} \end{bmatrix}$$

For $(x,y) \in \Delta_{M_bAM_c}$, we have:

$$(z_A - z) = a_1\cdot(x_A - x) + a_2\cdot(y_A - y),$$

$$\begin{bmatrix} z_A - z_{M_c} \\ z_A - z_{M_b} \end{bmatrix} = \begin{bmatrix} x_A - x_{M_c} & y_A - y_{M_c} \\ x_A - x_{M_b} & y_A - y_{M_b} \end{bmatrix}\cdot\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = X^T\cdot\begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$$

$$\iff \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = (X^{-1})^T\cdot\begin{bmatrix} z_A - z_{M_c} \\ z_A - z_{M_b} \end{bmatrix}$$

$$
\begin{aligned}
(a_2 - m_2) &= ((X^{-1})^T)_{2,\bullet} \cdot \left(\begin{bmatrix} z_A - z_{M_c} \\ z_A - z_{M_b} \end{bmatrix} - \begin{bmatrix} z_{M_b} - z_{M_a} \\ z_{M_c} - z_{M_a} \end{bmatrix} \right) \\
&= ((X^{-1})_{\bullet,2})^T \cdot \begin{bmatrix} 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} z_{M_a} \\ z_{M_b} \\ z_{M_c} \\ z_A \end{bmatrix} \\
&= ((X^{-1})_{\bullet,2})^T \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & -1 & -1 & 1 & 0 & 0 \end{bmatrix} \cdot \mathbf{z}_{k,l} \\
&= \left(\sum_i (X^{-1})_{i,2} \right) \cdot \begin{bmatrix} 1 & -1 & -1 & 1 & 0 & 0 \end{bmatrix} \cdot \mathbf{z}_{k,l} \\
&= h \cdot \frac{\sqrt{3}}{3} \cdot \begin{bmatrix} 1 & -1 & -1 & 1 & 0 & 0 \end{bmatrix} \cdot \mathbf{z}_{k,l}
\end{aligned}
$$

In summary,

$$
\begin{aligned}
&\forall (x,y) \in \square_{M_a M_b A M_c} : \\
&\mathcal{H}f(x,y) = \delta(y - y_{0x}) \cdot (a_2 - m_2) \cdot \begin{bmatrix} 3 & \sqrt{3} \\ \sqrt{3} & 1 \end{bmatrix} = \delta(y - y_{0x}) \cdot \mathbf{K}_1 \ , \\
&\qquad (\lambda_1)_{\mathbf{K}_1} = 0, \quad (\lambda_2)_{\mathbf{K}_1} = 4 \cdot (a_2 - m_2) \ , \\
&\qquad (a_2 - m_2) = h \cdot \frac{\sqrt{3}}{3} \cdot \begin{bmatrix} 1 & -1 & -1 & 1 & 0 & 0 \end{bmatrix} \cdot \mathbf{z}_{k,l}.
\end{aligned}
$$

Bibliography

[1] David Reinsel, John Gantz, and John Rydning. Data age 2025: The digitization of the world. Technical report, International Data Corporation., 2018.

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, Berlin, Heidelberg, 2006.

[3] M. Vetterli, J. Kovacevic, and V. Goyal. *Foundations of Signal Processing*. Cambridge University Press, 2014.

[4] Hong Cheng. *Sparse Representation, Modeling and Learning in Visual Recognition*. Advances in Computer Vision and Pattern Recognition. Springer-Verlag, London, UK, 2015.

[5] Michael Unser and Pouya D. Tafti. *An Introduction to Sparse Stochastic Processes*. Cambridge University Press, 2014.

[6] M. Unser. Splines: a perfect fit for signal and image processing. *IEEE Signal Processing Magazine*, 16(6):22–38, Nov 1999.

[7] Michael Unser. A representer theorem for deep neural networks. *CoRR*, abs/1802.09210, 2018.

[8] H. Gupta, J. Fageot, and M. Unser. Continuous-domain solutions of linear inverse problems with Tikhonov *versus* generalized TV regularization. *IEEE Transactions on Signal Processing*, 66(17):4670–4684, September 1, 2018.

[9] Thomas Debarre, Julien Fageot, Harshit Gupta, and Michael Unser. Solving continuous-domain problems exactly with multiresolution b-splines. *IEEE Transactions on Information Theory*, pages 5122–5126, 05 2019.

[10] Stamatios Lefkimmiatis, John Paul Ward, and Michael Unser. Hessian schatten-norm regularization for linear inverse problems. *IEEE Transactions on Image Processing*, 22(5):1873–1888, May 2013.

[11] Martin Jaggi, Rudiger Urbanke, and Mohammad Emtiyaz Khan. Lecture notes in Machine Learning CS-433. `https://www.epfl.ch/labs/mlo/machine-learning-cs-433/`, October 2019.

[12] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[13] M. Unser, J. Fageot, and H. Gupta. Representer theorems for sparsity- promoting ℓ_1 regularization. *IEEE Trans. Inf. Theory*, 62(9):5167–5180, 2016.

[14] Abolfazl Mehranian, Hamidreza Saligheh Rad, Arman Rahmim, Mohammad Reza Ay, and Habib Zaidi. Smoothly clipped absolute deviation (scad) regularization for compressed sensing mri using an augmented lagrangian scheme. *Magnetic Resonance Imaging*, 31(8):1399 – 1411, 2013.

[15] Bernhard Schölkopf, Ralf Herbrich, and Alex J. Smola. A generalized representer theorem. In David Helmbold and Bob Williamson, editors, *Computational Learning Theory*, pages 416–426, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[16] Peter Bartlett. Lecture notes in Statistical Learning Theory CS 281b/Stat 241b. `https://people.eecs.berkeley.edu/~bartlett/courses/281b-sp08/7.pdf`, 2008.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[18] Kyong Hwan Jin, Michael T. McCann, Emmanuel Froustey, and Michael Unser. Deep convolutional neural network for inverse problems in imaging. *IEEE Transactions on Image Processing*, 26, 2017.

[19] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015.

[20] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *CoRR*, abs/1710.10196, 2017.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[23] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016.

[24] Guido Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks, 2014.

[25] Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding deep neural networks with rectified linear units, 2016.

[26] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[27] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[28] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Imaging Sciences*, 2:183–202, 2009.

[29] Sheldon Axler. *Linear Algebra Done Right*. Springer, Cham, 2015.

[30] Benjamin Recht, Maryam Fazel, and Pablo A. Parrilo. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *SIAM Review*, 52(3):471–501, Jan 2010.

[31] Jim Belk. Second derivatives, lecture notes in Vector Calculus Math-241. `https://faculty.bard.edu/~belk/math241/SecondDerivatives.pdf`, 2016.

[32] Thomas C. Hales. The honeycomb conjecture, 1999.

[33] Mark Pauly. Lecture notes in Computer Graphics CS-433. `https://lgg.epfl.ch/teaching_introduction_computer_graphics.php`, 2017.