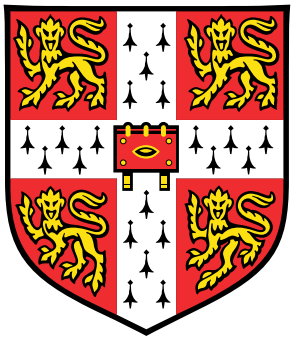
Compressive Sensing in Video Reconstruction



Brian Azizi

Department of Physics
Centre for Scientific Computing
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy

Selwyn College August 2016

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other University. This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text. This dissertation contains less than 15,000 words including appendices, bibliography, footnotes, tables and equations.

Brian Azizi
August 2016

Acknowledgements

Abstract

Table of contents

List of figures

List of tables

Nomenclature

Roman Symbols

M	Number of basis functions
N	Number of training examples
$\boldsymbol{w}$	RVM weights vector
$\boldsymbol{x}^{(i)}$	ith input vector
$y^{(i)}$	ith target

Greek Symbols

$\boldsymbol{\phi}_j$	jth basis vector
$\phi_j(\cdot)$	jth basis function

Chapter 1

Introduction

There are three parts: A signal processing framework called *Compressive Sensing (CS)*, a pre-processing step in form of a basis transformation based on discrete wavelet transforms and a Machine Learning algorithm called *Sparse Bayesian Learning*.

The key notion that ties in these three areas is the notion of *sparsity*.

The ℓ_p -norm of a vector $\boldsymbol{z}\mathbb{R}^n$ is defined as follows for $p > 0$:

$$||\boldsymbol{z}||_p \equiv \left(\sum_{i=1}^{n} |z_i|^p \right)^{\frac{1}{p}}. \tag{1.1}$$

If $p = 0$ we can define the ℓ_0 "norm" of $\boldsymbol{z}$ to be the number of its non-zero entries:

$$||\boldsymbol{z}||_0 \equiv \sum_{i=1}^{n} \mathbb{1}\{z_j \neq 0\} \tag{1.2}$$

Our contributions are three-fold:

1. An extension of the MSCE-algorithm in [5] to video data.
2. An extension to MSCE to deal with additional sensing mechanisms.
3. A parameter study and performance comparison of different wavelet representations, sensing modalities,

Scope

Thesis Organization

Chapter 2

Background

In this chapter, we will introduce the theory of *Compressive Sensing* (also known as *Compressed Sensing, Compressive Sampling* or simply, *CS*). CS is a framework within signal processing that allows for acquiring signals (i.e. measure or *sense*) directly in a *compressed* format.

To motivate the discussion, we will first review the conventional approach to signal acquisition and compression.

2.1 Coventional Signal Processing

2.1.1 Signal Acquisition

In order to work with information within analog signals (continuous streams of data) such as sounds, images or video, we rely on reducing the analog signals to digital (discrete) signals that can be processed with computers. This digitization is done by taking discrete measurements of the analog signal at certain points in time or space, a process known as *sampling*.

Conventional approaches to sampling are based on the *Shannon/Nyquist Sampling Theorem* [6]: When sampling a band-limited signal uniformly, we are able to *perfectly reconstruct* the signal from its samples if the sampling rate is at least twice the bandwidth of the signal.

Consider an analog signal $x(t)$ that varies with time, such as an audio wave. Let f be the highest frequency present in $x(t)$.

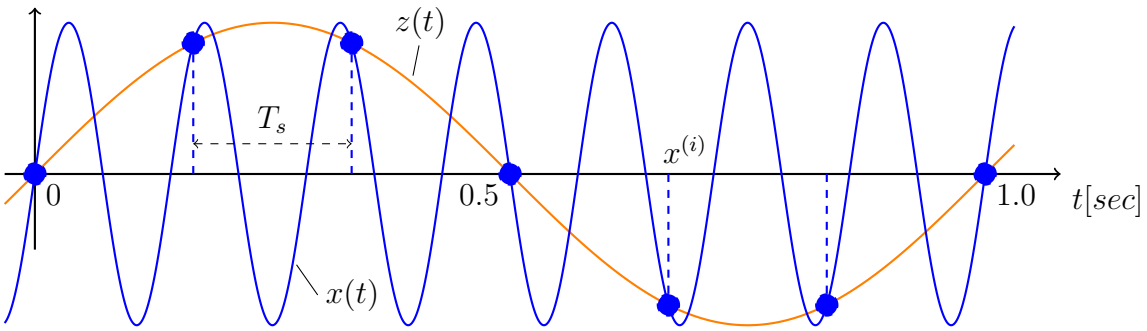


Fig. 2.1 Illustration of the Shannon/Nyquist Sampling Theorem. The orange curve is the original signal $x(t)$ which is a sinusoid with frequency $f = 7$ Hz. The blue points are discrete samples $x^{(i)}$ taken from $x(t)$ at a sampling rate $f_s = 7$ Hz, which is below the Nyquist Rate $2f = 14$ Hz. Thus, aliasing occurs and interpolation algorithms will reconstruct an alias $z(t)$ of $x(t)$.

In order to digitise $x(t)$, we measure x at discrete points in time $t^{(0)}, \cdots, t^{(n)}$ and store the samples $x^{(i)} \equiv x(t^{(i)})$. We sample x uniformly, measuring a sample every T_s seconds, so that $t^{(i)} = iT_s$. The sampling rate is therefore $f_s = 1/T_s$.

Suppose we wish to reconstruct $x(t)$ by interpolating the samples. There is an infinite number of continuous functions that fit this set of samples. However, it can be shown that only one of them has a bandwidth of no more than $f_s/2$. Thus, if $f < f_s/2$ (the *Nyquist Criterion*), then $x(t)$ is the unique function that will be approximated by interpolation algorithm such as the *Whittaker-Shannon interpolation formula* [6].

In Figure 2.1, we have a sinusoidal signal $x(t)$ with frequency f. The sampling rate is $f_s = f$ and therefore below the signal's *Nyquist rate* $2f$. Thus, we are unable to reconstruct $x(t)$ from the samples. Instead, we will reconstruct an *alias* $z(t)$ which, in this case, is another sinusoid with frequency $f/7$. The original signal x is lost.

We have illustrated Nyquist sampling in the 1-dimensional case. The same principles hold for higher dimensional signals such as images and videos.

For signals that vary with space, the sampling rate is governed by the desired spatial resolution. In order to recover the finer details (the high-frequency components) of an image, we require higher pixel density (i.e. a larger number of pixels per centimeter (ppcm)).

Nyquist sampling underlies almost all signal acquisition protocols that are found in practice. It is the basis of medical imaging, audio and video recording and radio receivers.

2.1.2 Signal Compression

The sampling theorem imposes a lower bound on the sampling rate above which we are able to perfectly reconstruct the desired signal. This lower bound is often very high and we end up with a very large number of measurements. Storage and transfer of such signals becomes prohibitavely expensive as the size of the signal grows. Thus, a need for *data compression* arises.

We will discuss a particular type compression algorithm known as *transform coding*. It is the standard compression method for "natural" and manmade signals such as audio, photos, and video and is the basis of many common signal formats such as JPEG (images), MPEG (video) and MP3 (audio).

Let $\boldsymbol{v}$ by a real-valued digital signal of length M, $\boldsymbol{v} \in \mathbb{R}^M$. Without loss of generality, $\boldsymbol{v}$ is assumed to be a one-dimensional signals. If we are working with a multi-dimensional signals, we may first vectorize it into a long vector. When compressing digital signals, we are usually interested in *lossy compression*.

Any vector in $\mathbb{R}^M$ can be expressed as a linear combination of M *basis vectors* $\boldsymbol{\psi}_j \in \mathbb{R}^M$:

$$\boldsymbol{v} = \sum_{j=1}^{M} w_j \boldsymbol{\psi}_j \tag{2.1}$$

where w_j is the coefficient (or weight) associated with $\boldsymbol{\psi}_j$.

By forming the *basis matrix* $\boldsymbol{\Psi} = [\boldsymbol{\psi}_1 \cdots \boldsymbol{\psi}_M]$, we can express equation (2.1) in matrix form

$$\boldsymbol{v} = \boldsymbol{\Psi}\boldsymbol{w}$$

where $\boldsymbol{w} = (w_1, \cdots, w_M)^T$. For simplicity, we assume that the basis $\boldsymbol{\Psi}$ is orthonormal, so that $\boldsymbol{\Psi}\boldsymbol{\Psi}^T = \boldsymbol{I}_M$ and $\boldsymbol{\psi}_i^T \boldsymbol{\psi}_j$ is 1 if $i = j$ and 0 otherwise. Thus, the coefficient w_j is given by $w_j = \boldsymbol{v}^T \boldsymbol{\psi}_j$.

We now have two equivalent representations of the same signal, $\boldsymbol{v}$ in the original basis and $\boldsymbol{w}$ in the $\boldsymbol{\Psi}$ basis. Since $\boldsymbol{\Psi}$ is orthogonal, $\boldsymbol{v}$ and $\boldsymbol{w}$ have the same ℓ_2-norm, $||\boldsymbol{v}||_2 = ||\boldsymbol{\Psi}\boldsymbol{w}||_2 = ||\boldsymbol{w}||_2$. However, in the original signal, $\boldsymbol{v}$, the energy is typically spread over many of its components. On the other, it is possible to find a basis $\boldsymbol{\Psi}$ such that the energy of the transformed signal, $\boldsymbol{w}$, is concentrated in only a few large components w_j and a large fraction of its entries are very close to zero.

Suppose that we delete the entries w_j that are very small and replace them with zero to obtain $\hat{\boldsymbol{w}}$. Let $\hat{\boldsymbol{v}} = \boldsymbol{\Psi}\hat{\boldsymbol{w}}$ the approximate signal in the original domain. Since

(a) Uncompressed Image (b) Compressed Image via DCT

Fig. 2.2 The uncompressed image has a resolution of 512×512, i.e. 262144 pixels. We compress the image by performing a Discrete Cosine Transform and storing only the largest 27832 coefficients. The compression ratio is 9.42.

$\hat{\boldsymbol{w}}$ is very close to $\boldsymbol{w}$, it follows that

$$||\hat{\boldsymbol{v}} - \boldsymbol{v}||_2 = ||\Psi\hat{\boldsymbol{w}} - \Psi\boldsymbol{w}||_2 = ||\Psi(\hat{\boldsymbol{w}} - \boldsymbol{w})||_2 = ||\hat{\boldsymbol{w}} - \boldsymbol{w}||_2$$

is very small.

Thus, a viable method for lossy compression of the signal $\boldsymbol{v} \in \mathbb{R}^M$ would be the following:

1. Compute the full set of transform coefficients $\{w_j\}_{j=1}^M$ via $\boldsymbol{w} = \Psi^T\boldsymbol{v}$.
2. Locate all the coefficients w_j whose absolute value is above a certain threshold (suppose there are K of them).
3. Discard all the $(M - K)$ small coefficients
4. Store the values and locations of the K large coefficients

In order to view the compressed signal in the original domain, we reconstruct it via the transform: $\boldsymbol{\Psi}\hat{\boldsymbol{w}} = \hat{\boldsymbol{v}}$, where $\hat{\boldsymbol{w}}$ is $\boldsymbol{w}$ with the $(M - K)$ smallest coefficients replaced by zero.

It is possible to find basis matrices $\boldsymbol{\Psi}$ that result in very high compression ratios for a wide range of natural signals without any noticable reduction in the signal quality.

Furthermore, many of the commonly used basis transforms can be computed very efficiently.

Audio signals and a wide class of communication signals are highly compressible in the localized Fourier basis. Images and video signals, on the other hand, can often be compressed via the *Discrete Cosine Transform* (DCT) or the *Discrete Wavelet Transform* (DWT). For instance, the JPEG standard for image compression is based on the DCT, while the more modern JPEG2000 format uses the CDF 9/7 wavelet transform or the CDF 5/3 wavelet transform [8].

In Figure 2.2, we compress the standard test image "Lenna" via a DCT. We are only storing about 10% of the transform coefficients. Yet, the difference between the original image and the compressed image is hardly noticable.

We will discuss the DCT and the DWT in more detail in Chapter 4.

2.2 Compressive Sensing

The conventional approach to data acquisition and compression is very effective and has been highly influential. However, it is also extremely wasteful. We acquire a huge amount of data at the signal sampling stage and then proceed to discard a large part of it at the compression stage.

Compressive Sensing is a more general approach that lets us *acquire signals directly in a compressed format*. This is clearly more efficient as it allows us to skip the intermediate stage of taking N samples.

In this section we will formulate the Compressive Sensing problem. For simplicity, the focus will be on discrete signals such as digital images or videos.

2.2.1 The Compressive Sensing Problem

Let $\boldsymbol{v} \in \mathbb{R}^M$ be the signal of interest. As an example, $\boldsymbol{v}$ could be a digital photograph, such as Figure 2.2a, that has been unrolled into a long vector of length M, where M is the number of pixels.

Suppose $\boldsymbol{v}$ is currently unknown and we want to acquire a compressed representation of it without measuring $\boldsymbol{v}$ directly. To do so, consider the following linear sensing scheme: We measure inner products between the signal $\boldsymbol{v}$ and a collection of M-dimensional vectors $\{\boldsymbol{\theta}_i\}_{i=1}^N$ to obtain the measurements $y_j = \boldsymbol{v}^T\boldsymbol{\theta}_i$ for $i = 1, \cdots, N$.

This relation can be expressed more succinctly as

$$\boldsymbol{y} = \boldsymbol{\Theta}\boldsymbol{v} \tag{2.2}$$

where $\boldsymbol{y} = (y_1, \cdots, y_N)^T$ and $\boldsymbol{\Theta}$ is the $N \times M$ *sensing matrix* given by

$$\boldsymbol{\Theta} = \begin{bmatrix} \boldsymbol{\theta}_1^T \\ \vdots \\ \boldsymbol{\theta}_N^T \end{bmatrix}. \tag{2.3}$$

We are interested in the *undersampled* situation where $N < M$. In particular, we would like to acquire a compressed representation $\boldsymbol{y}$ of $\boldsymbol{v}$ using as few measurements as possible, while still being able to recover the original signal.

There are two problems that stand out at this point.

1. Given that $N << M$, how can we ensure that, during the measurement process, we do not lose any information contained in $\boldsymbol{v}$ (i.e. $\boldsymbol{y}$ captures all the information in $\boldsymbol{v}$)?

2. Given our measurements $\boldsymbol{y}$ and knowledge of the sensing matrix $\boldsymbol{\Theta}$, how do we recover the signal of interest $\boldsymbol{v}$?

At an intuitive level, we would expect at least some information to be destroyed during the measurement process, especially if the number of measurements, N, is substantially lower than the length of the desired signal, M. Moreover, even if $\boldsymbol{y}$ did contain the same information as $\boldsymbol{v}$, we are still faced with solving the linear system in equation (2.2). This system is underdetermined and hence there are infinitely many vectors $\boldsymbol{v}$ that satisfy $\boldsymbol{\Theta}\boldsymbol{v} = \boldsymbol{y}$.

2.2.2 Sparse Signals

In general, we cannot resolve these issues. However, if we restrict ourselves to a certain class of signals, it is possible to make progress.

In particular, we will focus on signals that have *sparse representations*. A signal $\boldsymbol{v} \in \mathbb{R}^M$ has a sparse representation if there exists a $M \times M$ basis matrix $\boldsymbol{\Psi}$ so that the transformed signal $\boldsymbol{w}$, where $\boldsymbol{v} = \boldsymbol{\Psi}\boldsymbol{w}$, is *sparse*.

We say that $\boldsymbol{w}$ is K-*sparse* if $\boldsymbol{w}$ has K non-zero components. Equivalently, from (1.2)[1], $\boldsymbol{w}$ is K-sparse if $||\boldsymbol{w}||_0 = K$.

In Section 2.1.2 we noted that many manmade and natural signals are compressible. They have an almost-sparse representation, meaning that, when expressed in the basis $\boldsymbol{\Psi}$, almost all of their energy is contained in only K components and the remaining components are very close to zero. Such signals are well-approximated by K-sparse representations.

2.3 Solving the Compressive Sensing Problem

So let us suppose then, that the desired signal $\boldsymbol{v}$ is K-sparse when expressed in the $\boldsymbol{\Psi}$ basis, where K is small ($K \leq M$). We can substitute $\boldsymbol{v} = \boldsymbol{\Psi w}$ into (2.2) to get

$$\boldsymbol{y} = \boldsymbol{\Theta v} = \boldsymbol{\Theta \Psi w}$$

Let us define $\boldsymbol{\Phi} = \boldsymbol{\Theta \Psi}$, so that

$$\boldsymbol{y} = \boldsymbol{\Phi w}. \tag{2.4}$$

We have arrived at another underdetermined linear system. The CS measurements $\boldsymbol{y}$ are still the same as in (2.2), but the sensing matrix $\boldsymbol{\Theta}$ has been replaced by the new sensing matrix $\boldsymbol{\Phi}$.

However, we now want to recover the signal $\boldsymbol{w}$ and we can abuse the fact that $\boldsymbol{w}$ is K-sparse for some K. This allows us to address to problems above.

The information in $\boldsymbol{w}$ is highly localized. It is fully contained in only $K << M$ of its entries. Thus, as long as $N \geq K$, it should, at least in principle, be possible for a measurement vector $\boldsymbol{y}$ of length $N << M$ to completely capture the information within $\boldsymbol{w}$.

Furthermore, since $(M - K)$ entries of $\boldsymbol{w}$ are zero, it follows that in (2.4), $\boldsymbol{y}$ is a actually linear combination of only K columns of Φ. So, if $N \geq K$, we might expect to find suitable constraints that allow us to recover $\boldsymbol{w}$. Once $\boldsymbol{w}$ is recovered, we can compute $\boldsymbol{v}$ via $\boldsymbol{v} = \boldsymbol{\Psi w}$.

[1]Definition of ℓ_0-norm.

2.3.1 Constructing the Sensing Mechanism

In practice, we do not know the locations of the K nonzero entries in $\boldsymbol{w}$. So the first challenge in a compressive sensing system is to design a $N \times M$ sensing matrix $\boldsymbol{\Theta}$ that ensures we measure all the information *for any* signal $\boldsymbol{v}$ that has a K-sparse representation in the basis $\boldsymbol{\Psi}$.

We will not go into the theoretical details of designing an optimal CS measurement process, Instead we will refer the reader to [2] and simply state three particular sensing matrices that have been used successfully:

1. Form $\boldsymbol{\Theta}$ by sampling its entries θ_{ij} independently from the Gaussian distribution with mean zero and variance $1/M$: $\theta_{ij} \overset{iid}{\sim} \mathcal{N}(0, 1/M)$.

2. Sample the entries θ_{ij} independently from a symmetric Bernoulli distribution, $P(\theta_{ij} = \pm 1/\sqrt{M}) = 1/2$.

3. Form $\boldsymbol{\Theta}$ by starting with the $M \times M$ identity matrix I_M and deleting $M - N$ of its rows at random. This sensing matrix corresponds to measuring a down-sampled version of the signal $\boldsymbol{v}$ directly. If the signal of interest $\boldsymbol{v}$ is a digital image, then $\boldsymbol{\Theta}$ measures $\boldsymbol{v}$ after an *image mask* has been applied to it. This sensing mechanism has been succesfully used in [5].

2.3.2 Signal Recovery

Using the sensing mechanisms outlined above, we are able to measure a signal $\boldsymbol{v}$ directly in a compressed format $\boldsymbol{y}$. The final part of a Compressive Sensing system is a signal reconstruction algorithm that decompresses $\boldsymbol{y}$ and recovers $\boldsymbol{v}$ or, equivalently, its sparse representation $\boldsymbol{w}$.

This can be achieved by searching for the sparsest solution $\boldsymbol{w}$ that satisfies equation (2.4) [1]. That is, the desired signal is the solution to the optimization problem:

$$\min_{\boldsymbol{w}} ||\boldsymbol{w}||_0 \qquad \text{subject to} \qquad \boldsymbol{\Phi w} = \boldsymbol{y} \tag{2.5}$$

Unfortunately, (2.5) is an NP-complete problem that can only be solved by an exhaustive search through all possible combinations for the locations of the non-zero entries in $\boldsymbol{w}$.

Luckily, and somewhat surprisingly, it can be shown that, if $N = O(K \log(M/K))$, it is possible to reconstruct K-sparse signals exactly by solving the ℓ_1-optimization

problem [1, 2]

$$\min_{\boldsymbol{w}} ||\boldsymbol{w}||_1 \qquad \text{subject to} \qquad \boldsymbol{\Phi w} = \boldsymbol{y} \tag{2.6}$$

The solution $\hat{\boldsymbol{w}}$ can be expressed in the original domain via $\hat{\boldsymbol{v}} = \Psi\hat{\boldsymbol{w}}$.

In practice, signals $\boldsymbol{v}$ usually only have an almost-sparse representation. Thus, the reconstruction is not completely exact and and the solution $\hat{\boldsymbol{v}}$ will be a close approximation to $\boldsymbol{v}$.

A large part of the CS literature is focused on developing algorithms that recover a signal $\boldsymbol{v}$ from a set CS measurement $\boldsymbol{y}$, either by solving (2.6) or through some alternative route.

For a discussion and comparison of some of the most widely used algorithms, see [5]. In Chapter 5 we will discuss a particular framework for CS signal reconstruction known as *Bayesian Compressive Sensing*.

Chapter 3

Sparse Bayesian Learning

Sparse Bayesian Learning [10] is a general Bayesian framework within supervised Machine Learning. It can be applied to both regression and classification tasks. The *Relevance Vector Machine*, or *RVM*, is a particular specialisation of the Sparse Bayesian Learning model which has identical functional form to the Support Vector Machine (SVM). However, the RVM comes with a number of key advantages over the SVM. The solution produced by a RVM is typically much sparser than the solution by a comparable SVM. Furthermore, the RVM is a probabilistic model and as such, allows us to estimate error bounds in its predictions.

In this chapter, we will derive the Sparse Bayesian Learning model for regression. We will summarise both the original inference algorithm [10] and also the faster "Sequential Sparse Bayesian Learning Algorithm" [11].

3.1 Model Specification

We are given a data set of N input vectors $\{\boldsymbol{x}^{(i)}\}_{i=1}^{N}$ and their associated *targets* $\{y^{(i)}\}_{i=1}^{N}$. The input vectors live in D-dimensional space, $\boldsymbol{x} \in \mathbb{R}^D$. The targets are real values, $y \in \mathbb{R}$. [1]

[1] When using the Sparse Bayesian model for regression, we assume the targets are real-valued. It is also possible to use the model for classification in which cased the targets are assumed to be discrete class labels.

We model the data using a linearly-weighted sum of M fixed basis functions $\{\phi_j(\cdot)\}_{j=1}^M$ and base our predictions on the function $f(\cdot)$ defined as

$$f(\boldsymbol{x}; \boldsymbol{w}) = \sum_{j=1}^{M} w_j \phi_j(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) \tag{3.1}$$

where $\boldsymbol{w} = [w_1, \ldots, w_M]^T$ and $\boldsymbol{\phi}(\cdot) = [\phi_1(\cdot), \ldots, \phi_M(\cdot)]^T$. Using a large number M of non-linear basis functions $\phi_j : \mathbb{R}^D \to \mathbb{R}$ allows for a highly flexible model.

The *Relevance Vector Machine*, or RVM, is a specialisation of the Sparse Bayesian Learning model in which the basis functions take the form of *kernel functions*

$$\phi_j(\cdot) \equiv K\left(\cdot\,,\, \boldsymbol{x}^{(j)}\right).$$

This defines a basis function for each training data point $\boldsymbol{x}^{(i)}$. Typically, we also include an additional *bias* term $\phi_0(\cdot) \equiv 1$, so that $M = N + 1$. The RVM has identical functional form to the popular Support Vector Machine (SVM), but superior properties. It typically gives sparser solutions than the SVM and has the additional advantage of providing confidence measures for its predictions.

However, in the following derivation, we will stick to the case of general basis functions $\phi_j : \mathbb{R}^D \to \mathbb{R}$. Thus M need not equal $N + 1$ and may, in fact, be a lot larger.

To train the model (3.1), i.e. find values for $\boldsymbol{w}$ that are optimal in some sense, we make the standard assumption that our training data are samples from the model with additive noise:

$$\begin{aligned} y^{(i)} &= f(\boldsymbol{x}^{(i)}; \boldsymbol{w}) + \epsilon^{(i)} \\ &= \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}^{(i)}) + \epsilon^{(i)} \qquad i = 1, \ldots, N. \end{aligned} \tag{3.2}$$

The errors $\{\epsilon^{(i)}\}_{i=1}^N$ are assumed to be independent samples from a zero-mean Gaussian distribution with variance σ^2

$$p(\epsilon^{(i)}) = \mathcal{N}(\epsilon^{(i)} \,|\, 0, \sigma^2) \qquad i = 1, \ldots, N. \tag{3.3}$$

Combining equation (3.2) with equation (3.1), we may express the model for the complete data using matrix notation:

$$\boldsymbol{y} = \boldsymbol{\Phi}\boldsymbol{w} + \boldsymbol{\epsilon} \tag{3.4}$$

where $\boldsymbol{\epsilon} = \left[\epsilon^{(1)}, \cdots, \epsilon^{(N)}\right]^T$. The $N \times M$ matrix $\boldsymbol{\Phi}$ is known as the design matrix. The ith row of $\boldsymbol{\Phi}$ is given by $\boldsymbol{\phi}(\boldsymbol{x}^{(i)})^T$. The jth column of $\boldsymbol{\Phi}$ is given by $\boldsymbol{\phi}_j = \left[\phi_j\left(\boldsymbol{x}^{(1)}\right), \cdots, \phi_j\left(\boldsymbol{x}^{(N)}\right)\right]^T$, which is also referred to as the jth *basis vector*. Thus

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\phi}_1 & \cdots & \boldsymbol{\phi}_M \end{bmatrix} = \begin{bmatrix} \boldsymbol{\phi}(\boldsymbol{x}^{(1)})^T \\ \vdots \\ \boldsymbol{\phi}(\boldsymbol{x}^{(N)})^T \end{bmatrix}$$

Combining equation (3.4) and equation (3.3), we find that the complete data likelihood function is given by

$$\begin{aligned} p\left(\boldsymbol{y} \,|\, \boldsymbol{w}, \sigma^2\right) &= \mathcal{N}\left(\boldsymbol{y} \,|\, \boldsymbol{w}, \sigma^2 \boldsymbol{I}_M\right) \\ &= (2\pi\sigma^2)^{-N/2} \exp\left\{-\frac{1}{2\sigma^2}||\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w}||^2\right\} \end{aligned} \tag{3.5}$$

where $\boldsymbol{I}_M$ is the $M \times M$ identity matrix.

So far, we have specified the general linear regression model. To get to the sparse Bayesian formulation, we define a zero-mean Gaussian prior distribution over the parameters $\boldsymbol{w}$

$$p(\boldsymbol{w} \,|\, \boldsymbol{\alpha}) = \prod_{j=1}^{M} \mathcal{N}\left(w_j \,|\, 0, \alpha_j^{-1}\right) \tag{3.6}$$

where $\boldsymbol{\alpha} = [\alpha_1, \cdots, \alpha_M]^T$ is a vector of M hyperparameters. It is important to note that each hyperparameter α_j is solely responsible for controlling the strength of the prior of its associated weight w_j. If α_j is large, the prior over w_j is very strongly peaked at zero. This form of the prior distribution is, more than anything, responsible for the dramatic sparsity in the final model.

To complete the specification, we must define a prior over the noise parameter σ^2 and the a hyperprior over the hyperparameters $\boldsymbol{\alpha}$. Following the derivation in [10], we

use the following Gamma [2] priors

$$p(\boldsymbol{\alpha} \,|\, a, b) = \prod_{j=1}^{M} \text{Gamma}(\alpha_j \,|\, a, b) \tag{3.7}$$

$$p(\beta \,|\, c, d) = \text{Gamma}(\beta \,|\, c, d) \tag{3.8}$$

where $\beta \equiv \sigma^{-2}$.

As a side note, consider the prior of $\boldsymbol{w}$ after marginalising out the dependence on the hyperpriors $\boldsymbol{\alpha}$. Since each w_j is normally distributed with an unknown precision parameter α_j and since the (hyper)prior over α_j is the Gamma distribution and therefore conjugate to $p(w_j \,|\, \alpha_j)$, it follows that the resulting integral can be evaluated analytically

$$\begin{aligned} p(\boldsymbol{w} \,|\, a, b) &= \int p(\boldsymbol{w} \,|\, \boldsymbol{\alpha}) p(\boldsymbol{\alpha} \,|\, a, b) d\boldsymbol{\alpha} \\ &= \prod_{j=1}^{M} \int \mathcal{N}(w_j \,|\, 0, \alpha_j^{-1}) \, \text{Gamma}(\alpha_j | \, a, b) d\alpha_j \\ &= \prod_{j=1}^{M} \frac{b^a \Gamma(a + \frac{1}{2})}{(2\pi)^{\frac{1}{2}} \Gamma(a)} \left(b + \frac{w_j^2}{2} \right)^{-(a+\frac{1}{2})} . \end{aligned}$$

This corresponds to a product of independent Student-t density functions over the weights w_j. The choice $a = b = 0$ implies that $p(\boldsymbol{w} \,|\, a, b) \propto \prod_{j=1}^{M} 1/|w_j|$. As discussed in [10], it is this hierarchical formulation of the weight prior that is ultimately responsible for encouraging sparse solutions.

3.2 Model Inference

We have specified the likelihood model for the data and a prior distribution over the model parameters. The next step in Bayesian inference is to compute the posterior

[2] The Gamma distribution is defined by

$$\text{Gamma}(z \,|\, a, b) = \Gamma(a)^{-1} b^a z^{a-1} \exp(-bz) \qquad z, a, b > 0$$

where $\Gamma(.)$ is the Gamma function defined by

$$\Gamma(z) = \int_0^\infty t^{z-1} \exp(-t) dt.$$

distribution of the parameters. We begin by setting up Bayes' Rule

$$p(\boldsymbol{w}, \boldsymbol{\alpha}, \sigma^2 \,|\, \boldsymbol{y}) = \frac{p(\boldsymbol{y} \,|\, \boldsymbol{w}, \boldsymbol{\alpha}, \sigma^2) p(\boldsymbol{w}, \boldsymbol{\alpha}, \sigma^2)}{\int p(\boldsymbol{y} \,|\, \boldsymbol{w}, \boldsymbol{\alpha}, \sigma^2) p(\boldsymbol{w}, \boldsymbol{\alpha}, \sigma^2) d\boldsymbol{w} d\boldsymbol{\alpha} d\sigma^2} \tag{3.9}$$

The integral in the denominator of (3.9) is computationally intractable and we must resort to an alternative strategy. First, we decompose the left-hand-side of equation (3.9) as

$$p(\boldsymbol{w}, \boldsymbol{\alpha}, \sigma^2 \,|\, \boldsymbol{y}) = p(\boldsymbol{w} \,|\, \boldsymbol{y}, \boldsymbol{\alpha}, \sigma^2) p(\boldsymbol{\alpha}, \sigma^2 \,|\, \boldsymbol{y}).$$

Next, we use Bayes' Rule to compute the posterior distribution of the weights given $\boldsymbol{\alpha}$ and σ^2

$$p(\boldsymbol{w} \,|\, \boldsymbol{y}, \boldsymbol{\alpha}, \sigma^2) = \frac{p(\boldsymbol{y} \,|\, \boldsymbol{w}, \sigma^2) p(\boldsymbol{w} \,|\, \boldsymbol{\alpha})}{p(\boldsymbol{y} \,|\, \boldsymbol{\alpha}, \sigma^2)} \tag{3.10}$$

The denominator of the right-hand-side is known as the *marginal likelihood* and given by

$$p(\boldsymbol{y} \,|\, \boldsymbol{\alpha}, \sigma^2) = \int p(\boldsymbol{y} \,|\, \boldsymbol{w}, \sigma^2) p(\boldsymbol{w} \,|\, \boldsymbol{\alpha}) d\boldsymbol{w} \tag{3.11}$$

Since $\boldsymbol{\alpha}$ and σ^2 are treated as fixed quantities in equation (3.10), the Gaussian density $p(\boldsymbol{w} \,|\, \boldsymbol{\alpha})$ is the conjugate prior to the Gaussian likelihood function $p(\boldsymbol{y} \,|\, \boldsymbol{w}, \sigma^2)$. Thus, the integral in equation (3.11) is a convolution of two Gaussians and therefore equal to another Gaussian:

$$\begin{aligned} p(\boldsymbol{y} \,|\, \boldsymbol{\alpha}, \sigma^2) &= \mathcal{N}(\boldsymbol{y} \,|\, \boldsymbol{0}, \boldsymbol{C}) \\ &= (2\pi)^{-N/2} |\boldsymbol{C}|^{-1/2} \exp\left\{ -\frac{1}{2} \boldsymbol{y}^T \boldsymbol{C}^{-1} \boldsymbol{y} \right\} \end{aligned}$$

where

$$\boldsymbol{C} = \sigma^2 \boldsymbol{I}_N + \boldsymbol{\Phi} \boldsymbol{A}^{-1} \boldsymbol{\Phi}^T. \tag{3.12}$$

The posterior distribution for $\boldsymbol{w}$ is a also a Gaussian:

$$p(\boldsymbol{w} \,|\, \boldsymbol{y}, \boldsymbol{\alpha}, \sigma^2) = \mathcal{N}(\boldsymbol{w} \,|\, \boldsymbol{\mu}, \boldsymbol{\Sigma}). \tag{3.13}$$

Its mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ are given by

$$\boldsymbol{\Sigma} = \left(\sigma^{-2} \boldsymbol{\Phi}^T \boldsymbol{\Phi} + \boldsymbol{A} \right)^{-1} \tag{3.14}$$

$$\boldsymbol{\mu} = \sigma^{-2} \boldsymbol{\Sigma} \boldsymbol{\Phi}^T \boldsymbol{y} \tag{3.15}$$

with $\boldsymbol{A} = \text{diag}(\boldsymbol{\alpha})$.

Finally, we need to find the posterior of the hyperparameters, $p(\boldsymbol{\alpha}, \sigma^2 \,|\, \boldsymbol{y})$. This part is computationally intractable, so instead we approximate the posterior by a delta-function at its mode. Hence, the problem reduces to finding the values of $\boldsymbol{\alpha}$ and σ^2 that maximise $p(\boldsymbol{\alpha}, \sigma^2 \,|\, \boldsymbol{y}) \propto p(\boldsymbol{y} \,|\, \boldsymbol{\alpha}, \sigma^2) p(\boldsymbol{\alpha}) p(\sigma^2)$.

Here, we make the simplifying assumption that $a = b = c = d = 0$, giving us uniform (but improper) hyperpriors (see [10] for the general case). Maximising $p(\boldsymbol{\alpha}, \sigma^2 \,|\, \boldsymbol{y})$ is then equivalent to maximising the marginal likelihood, or equivalently, its logarithm

$$\begin{aligned} \mathcal{L}(\boldsymbol{\alpha}, \sigma^2) &= \log p(\boldsymbol{y} \,|\, \boldsymbol{\alpha}, \sigma^2) = \log \mathcal{N}(\boldsymbol{y} \,|\, \boldsymbol{0}, \boldsymbol{C}) \\ &= -\frac{1}{2} \left[N \log 2\pi + \log |\boldsymbol{C}| + \boldsymbol{y}^T \boldsymbol{C}^{-1} \boldsymbol{y} \right] \end{aligned} \tag{3.16}$$

The procedure of finding $\boldsymbol{\alpha}$ and σ^2 that maximise the (log) marginal likelihood (3.16) is also known as *type-II Maximum likelihood* and *evidence approximation.*

3.2.1 Original Training Algorithm

The original training algorithm in [10] is derived by setting the derivatives of (3.16) to zero. We obtain the following update equations for $\boldsymbol{\alpha}$ and σ^2:

$$\alpha_j^{\text{new}} = \frac{\gamma_j}{\mu_j^2} \tag{3.17}$$

$$(\sigma^2)^{\text{new}} = \frac{||\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\mu}||^2}{N - \sum_j \gamma_j} \tag{3.18}$$

where μ_j is the jth component of the posterior mean $\boldsymbol{\mu}$ (3.15). The quantities γ_j are defined by

$$\gamma_j = 1 - \alpha_j \Sigma_{jj}$$

where Σ_{jj} is the jth diagonal element of the posterior covariance $\boldsymbol{\Sigma}$ (3.14).

To train the model, we can start by giving $\boldsymbol{\alpha}$ and σ^2 some initial values and evaluate the mean and covariance of the weights posterior using equations (3.15) and (3.14), respectively. Next we alternate between re-estimating the hyperparamaters $\boldsymbol{\alpha}$ and σ^2 using (3.17) and (3.18) and updating the posterior mean and covariance parameters using (3.15) and (3.14). We continue until a relevant convergence criterion is met. For example, we may choose to stop if the change in the marginal likelihood - or,

Algorithm 1 Sparse Bayesian Learning: Original Training Algorithm

Choose some initial positive values for σ^2 and α_j for $j = 1, \cdots, M$
repeat
  $\boldsymbol{A} = \text{diag}(\boldsymbol{\alpha})$
  $\boldsymbol{\Sigma} = \left(\sigma^{-2}\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \boldsymbol{A}\right)^{-1}$
  $\boldsymbol{\mu} = \sigma^{-2}\boldsymbol{\Sigma}\boldsymbol{\Phi}^T\boldsymbol{y}$

  for $j = 1, \cdots, M$ **do**
    $\gamma_j = 1 - \alpha_j\Sigma_{jj}$
    $\alpha_j = \gamma_j/\mu_j^2$
  end for
  $\sigma^2 = ||\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\mu}||^2/(N - \sum_j \gamma_j)$
until Convergence

alternatively, the change in the parameter values - between two iterations is below a certain pre-defined threshold.

This procedure is summarised in Algorithm 1.

During training, it is typically observed that many of the hyperparameters α_j tend to infinity. Equations (3.15) and (3.14) imply that the weights w_j corresponding to these hyperparameters have a posterior distribution where the mean and the variance are both zero, meaning their posterior is infinitely peaked at zero. As a consequence, the corresponding basis functions $\phi_j(\cdot)$ are effectively removed from the model and we achieve sparsity.

In the case of the RVM, where $\phi_j(\cdot) \equiv K(\cdot, \boldsymbol{x}^{(j)})$, the input vectors $\boldsymbol{x}^{(j)}$ corresponding to the remaining non-zero weights are known as the *relevance vectors* of the model.

3.2.2 Sequential Sparse Bayesian Learning Algorithm

A central drawback of the training algorithm discussed in the previous section is its speed. The computational complexity scales with the cube of the number of basis functions. During training, as basis functions are pruned from the model, the algorithm accelerates. Nevertheless, if M is very large, the procedure can be very expensive to run.

An alternative strategy of maximising the marginal likelihood (3.16) was developed by [11], resulting in a highly accelerated training algorithm: the *Sequential Sparse Bayesian Learning Algorithm*. It starts with a single basis function and maximises

the marginal likelihood by sequentially adding and deleting candidate basis functions. This significantly reduces the computational complexity of the algorithm.

To derive the algorithm, we follow the analysis in [9] and consider the dependence of the marginal likelihood $\mathcal{L}(\boldsymbol{\alpha}, \sigma^2)$ on a single hyperparameter α_j. First, we decompose the matrix $\boldsymbol{C}$, defined in (3.12), as follows:

$$\begin{aligned}\boldsymbol{C} &= \sigma^2 \boldsymbol{I}_N + \sum_{m \neq j} \alpha_m^{-1} \boldsymbol{\phi}_m \boldsymbol{\phi}_m^T + \alpha_j^{-1} \boldsymbol{\phi}_j \boldsymbol{\phi}_j^T \\ &= \boldsymbol{C}_{-j} + \alpha_j^{-1} \boldsymbol{\phi}_j \boldsymbol{\phi}_j^T\end{aligned}$$

where $\boldsymbol{C}_{-j} \equiv \sigma^2 \boldsymbol{I}_N + \sum_{m \neq j} \alpha_m^{-1} \boldsymbol{\phi}_m \boldsymbol{\phi}_m^T$ is $\boldsymbol{C}$ without the contribution of the jth basis vector $\boldsymbol{\phi}_j$. Making use of standard identities [WHICH ONES?] for matrix inverses and determinants, we can express $|\boldsymbol{C}|$ and $\boldsymbol{C}^{-1}$ as

$$\boldsymbol{C}^{-1} = \boldsymbol{C}_{-j}^{-1} - \frac{\boldsymbol{C}_{-j}^{-1} \boldsymbol{\phi}_j \boldsymbol{\phi}_j^T \boldsymbol{C}_{-j}^{-1}}{\alpha_j + \boldsymbol{\phi}_j^T \boldsymbol{C}_{-j}^{-1} \boldsymbol{\phi}_j}$$

$$|\boldsymbol{C}| = |\boldsymbol{C}_{-j}| \left| 1 + \alpha_j^{-1} \boldsymbol{\phi}_j^T \boldsymbol{C}_{-j}^{-1} \boldsymbol{\phi}_j \right|$$

This allows us to decompose the marginal likelihood:

$$\begin{aligned}\mathcal{L}(\boldsymbol{\alpha}, \sigma^2) &= \mathcal{L}(\boldsymbol{\alpha}_{-j}, \sigma^2) + \frac{1}{2} \left[\log \alpha_j - \log(\alpha_j + s_j) + \frac{q_j^2}{\alpha_j + s_j} \right] \\ &\equiv \mathcal{L}(\boldsymbol{\alpha}_{-j}, \sigma^2) + \ell(\alpha_j, \sigma^2)\end{aligned} \tag{3.19}$$

This conveniently seperates terms in α_j in $\ell(\alpha_j, \sigma^2)$ from the remaining terms in $\mathcal{L}(\boldsymbol{\alpha}_{-j}, \sigma^2)$, which is the (log) marginal likelihood with the basis vector $\boldsymbol{\phi}_j$ excluded.

The quantity s_j is the *sparsity factor*, defined as

$$s_j = \boldsymbol{\phi}_j^T \boldsymbol{C}_{-j}^{-1} \boldsymbol{\phi}_j.$$

It serves as a measure of how much the marginal likelihood would decrease if we added $\boldsymbol{\phi}_j$ to the model. The quantity q_j, on the other hand, is known as the *quality factor*. It is defined as

$$q_j = \boldsymbol{\phi}_j^T \boldsymbol{C}_{-j}^{-1} \boldsymbol{y}$$

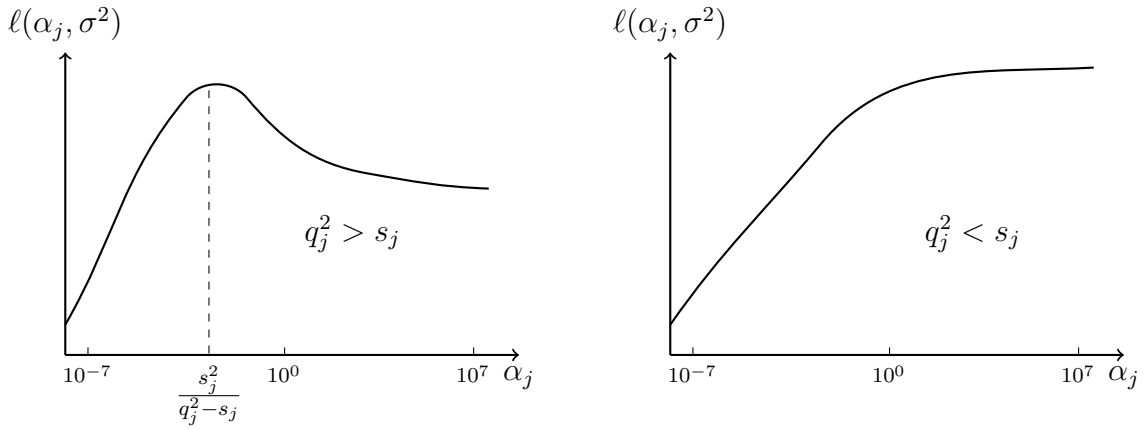


Fig. 3.1 Example plots of $\ell(\alpha_j, \sigma^2)$ against α_j illustrating the stationary points when $q_j^2 > s_j$ (left) and $q_j^2 < s_j$ (based on [9]).

and measures the extent to which ϕ_j increases $\mathcal{L}(\boldsymbol{\alpha}, \sigma^2)$ by helping to explain the data $\boldsymbol{y}$. Thus, a particular basis vector $\boldsymbol{\phi}_j$ should not be included in the model if its sparsity factor s_j is large, unless it is offset by a large quality factor q_j.

We can see this more explicitly if we consider the first derivative of $\ell(\alpha_j, \sigma^2)$ with respect to α_j [9]

$$\frac{\partial \ell(\alpha_j, \sigma^2)}{\partial \alpha_j} = \frac{\alpha_j^{-1} s_j^2 - (q_j^2 - s_j)}{2(\alpha_j + s_j)^2}$$

Equating it to zero (and noting that α_j is an inverse-variance and therefore positive), we obtain the following solution for α_j:

$$\alpha_j = \begin{cases} s_j^2/(q_j^2 - s_j) & \text{if } q_j^2 > s_j \\ +\infty & \text{otherwise} \end{cases} . \tag{3.20}$$

The solution (3.20) is illustrated in Figure 3.1.

It follows that, if, during training, a candidate basis vector $\boldsymbol{\phi}_j$ is currently included in the model (meaning $\alpha_j < \infty$) even though $q_j^2 \leq s_j$, then α_j should be set to ∞ and $\boldsymbol{\phi}_j$ should be pruned from the model. On the other hand, if $\boldsymbol{\phi}_j$ is currently excluded from the model (i.e. $\alpha_j = \infty$), but $q_j^2 > s_j$, then α_j should be set to $s_j^2/(q_j^2 - s_j)$ and $\boldsymbol{\phi}_j$ should be added to the model. Furthermore, if $\boldsymbol{\phi}_j$ is included and $q_j^2 > s_j$, then we may also re-estimate α_j. Each step in the algorithm (weakly) increases the marginal likelihood. Thus we are guaranteed to find a maximum.

During the algorithm, we must maintain and update values of the quality factors and sparsity factors for all basis functions, as well as the posterior mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$

Algorithm 2 Sequential Sparse Bayesian Learning Algorithm [11]

1: Initialise σ^2.
2: Add basis function $\boldsymbol{\phi}_j$ to the model, where $j = \arg\max_m \left\{ ||\boldsymbol{\phi}_m^T \boldsymbol{y}||^2 / ||\boldsymbol{\phi}_m||^2 \right\}$. Set $\alpha_j = \frac{||\boldsymbol{\phi}_j||^2}{||\boldsymbol{\phi}_j^T \boldsymbol{y}||^2 / ||\boldsymbol{\phi}_j||^2 - \sigma^2}$. Set $\alpha_m = \infty$ for $m \neq j$.
3: Compute $\boldsymbol{\Sigma} = \left(\sigma^{-2} \boldsymbol{\Phi}^T \boldsymbol{\Phi} + \boldsymbol{A} \right)^{-1}$ and $\boldsymbol{\mu} = \sigma^{-2} \boldsymbol{\Sigma} \boldsymbol{\Phi}^T \boldsymbol{y}$ which are scalars initially. Compute S_m, Q_m, s_m and q_m for $m = 1, \cdots, M$ using (3.21) − (3.24).
4: **repeat**
5: Select some candidate basis vector $\boldsymbol{\phi}_j$.
6: **if** $q_j^2 > s_j$ and $\alpha_j = \infty$ **then add** $\boldsymbol{\phi}_j$ to the model and update α_j.
7: **if** $q_j^2 > s_j$ and $\alpha_j < \infty$ **then re-estimate** α_j.
8: **if** $q_j^2 < s_j$ and $\alpha_j < \infty$ **then delete** $\boldsymbol{\phi}_j$ from the model and set $\alpha_j = \infty$.
9: Update $\sigma^2 = ||\boldsymbol{y} - \boldsymbol{\Phi} \boldsymbol{w}|| / (N - M + \sum_m \alpha_m \boldsymbol{\Sigma}_{mm})$[10].
10: Update $\boldsymbol{\Sigma}$, $\boldsymbol{\mu}$ and S_m, Q_m, s_m, q_m for $m = 1, \cdots, M$.
11: **until** Convergence

of the weights $\boldsymbol{w}$. In practice, it easier to keep track of the quantities $Q_m = \boldsymbol{\phi}_m^T \boldsymbol{C}^{-1} \boldsymbol{\phi}_m$ and $S_m = \boldsymbol{\phi}_m^T \boldsymbol{C}^{-1} \boldsymbol{y}$ which can also be written as (using the Woodbury Identity)

$$S_m = \sigma^{-2} \boldsymbol{\phi}_m^T \boldsymbol{\phi}_m - \sigma^{-4} \boldsymbol{\phi}_m^T \boldsymbol{\Phi} \boldsymbol{\Sigma} \boldsymbol{\Phi}^T \boldsymbol{\phi}_m \tag{3.21}$$

$$Q_m = \sigma^{-2} \boldsymbol{\phi}_m^T \boldsymbol{y} - \sigma^{-4} \boldsymbol{\phi}_m^T \boldsymbol{\Phi} \boldsymbol{\Sigma} \boldsymbol{\Phi}^T \boldsymbol{y} \tag{3.22}$$

where $\boldsymbol{\Sigma}$ and $\boldsymbol{\Phi}$ contain only the basis functions that are currently included in the model.

The factors s_m and q_m can by obtained from S_m and Q_m as follows:

$$s_m = \frac{\alpha_m S_m}{\alpha_m - S_m} \tag{3.23}$$

$$q_m = \frac{\alpha_m Q_m}{\alpha_m - S_m} \tag{3.24}$$

Note that if $\alpha_m = \infty$, then $q_m = Q_m$ and $s_m = S_m$.

We have summarized the procedure in Algorithm 2. After initializing the standard deviation σ^2 in step 1, we add the first basis function $\boldsymbol{\phi}_j$ to the model. We could initialize with any basis vector, but in step 2, we pick the one with the largest normalized projection on the target vector $\boldsymbol{y}$, i.e. we choose $j = \arg\max_m \left\{ ||\boldsymbol{\phi}_m^T \boldsymbol{y}||^2 / ||\boldsymbol{\phi}_m||^2 \right\}$. In step 3 we compute the model statistics and in step 4 we begin the large loop of the

algorithm. There are two things to note here. First, in step 5, we need to select a candidate basis vector $\boldsymbol{\phi}_j$. We are free to pick one at random. Alternatively, it is possible to compute the change in the marginal likelihood for each ccandidate basis vector and choose the one that would give us the largest increase. Second, we would usually like to estimate the noise variance σ^2 from the data, as is done in step 9. However, in practice, we may decide to set σ^2 in advance in step 1 and keep it fixed throughout the algorithm. If we decide to do so, then we can perform the updates in step 10 using very efficient update formulae that do not require matrix inversions. The formulae can be found in the appendix of [11]. If we do decide to update σ^2 in step 9, then we must use the full equations (3.14), (3.15) and (3.21)-(3.24).

3.3 Making Predictions

Once we have trained the model, we may use it to predict the target y^* for a new input vector $\boldsymbol{x}^*$. To do so, we would like to compute the *predictive distribution*

$$p(y^* \mid \boldsymbol{y}) = \int p(y^* \mid \boldsymbol{w}, \boldsymbol{\alpha}, \sigma^2) p(\boldsymbol{w}, \boldsymbol{\alpha}, \sigma^2 \mid \boldsymbol{y})\, d\boldsymbol{w}\, d\boldsymbol{\alpha}\, d\sigma^2.$$

We cannot compute this integral analytically, nor do we actually know the posterior of all the model parameters. Instead, we use the type-II maximum likelihood solutions for $\boldsymbol{\alpha}$ and σ^2 that we obtained during training and base our predictions on the posterior distribution of the weights conditioned on $\boldsymbol{\alpha}$ and σ^2. The predictive distribution for $\boldsymbol{x}^*$ is then:

$$p(y^* \mid \boldsymbol{y}, \boldsymbol{\alpha}, \sigma^2) = \int p(y^* \mid \boldsymbol{w}, \sigma^2) p(\boldsymbol{w} \mid \boldsymbol{y}, \boldsymbol{\alpha}, \sigma^2)\, d\boldsymbol{w} \tag{3.25}$$

Both factors in the integrand are Gaussians, and we can therefore readily compute the integral to get

$$p(y^* \mid \boldsymbol{y}, \boldsymbol{\alpha}, \sigma^2) = \mathcal{N}(y^* \mid \mu^*, (\sigma^2)^*) \tag{3.26}$$

The predictive mean is given by

$$\mu^* = \boldsymbol{\mu}^T \boldsymbol{\phi}(\boldsymbol{x}^*) \tag{3.27}$$

and the predictive variance is given by

$$(\sigma^2)^* = \sigma^2 + \boldsymbol{\phi}(\boldsymbol{x}^*)^T \boldsymbol{\Sigma} \boldsymbol{\phi}(\boldsymbol{x}^*) \tag{3.28}$$

Equation (3.27) implies that, if we want to produce point predictions, we may simply set the weights $\boldsymbol{w}$ equal to posterion mean $\boldsymbol{\mu}$ which is typically very sparse. If we are also interested in error bars for our predictions, we can obtain them using Equation (3.28). The error bars consist of two parts, the noise in the data σ^2 and the uncertainty in the weights.

For more details and derivations on Sparse Bayesian Learning, see [9–11].

Chapter 4

Basis Functions

In Section 2.1.2, we introduced transform coding. We said that any discrete signal $\boldsymbol{v} \in \mathbb{R}^M$ can be expressed in a different basis via a basis transform:

$$\boldsymbol{v} = \boldsymbol{\Psi}\boldsymbol{w}$$

where $\boldsymbol{\Psi}$ is the $M \times M$ basis matrix and $\boldsymbol{w} \in \mathbb{R}^M$ is the representation of $\boldsymbol{v}$ in the $\boldsymbol{\Psi}$ basis.

The particular classes of signals $\boldsymbol{v}$ that we are interested in are digital images and digital video. The aim of this chapter is to construct a basis matrix $\boldsymbol{\Psi}$ that gives us a (near-) sparse representation of a wide range of such signals $\boldsymbol{v}$. Finding a set of basis functions $\boldsymbol{\Psi}$ that achieve such a transformation lies at the heart of many lossy compression techniques.

It is important to note here that the choice of basis functions $\boldsymbol{\Psi}$ typically has a significant effect on the performance of the reconstruction algorithms.

4.1 Discrete Cosine Transform

The first basis transform that we will use is the Discrete Cosine Transform (DCT), one of the most widely used transforms in signal processing. It underlies JPEG image compression and is used in various video compression algorithms such as MJPEG, MPEG, H.261 and H.263 [12]. [1]

[1] A related transform, known as the *Modified DCT* is used in many lossy audio compression formats such as MP3, AAC and Vorbis.

(a) Original signal $\boldsymbol{v}$ (b) DCT of $\boldsymbol{v}$

Fig. 4.1 Panel (a) shows the original signal $\boldsymbol{v}$, a 256×256 grayscale image known as "cameraman". Panel (b) illustrates the 2-D DCT of $\boldsymbol{v}$. The brightness of a an element increases with the absolute value of the corresponding DCT coefficient. (The high-frequency coefficients have been enhanced to show more detail).

A DCT decomposes a signal in terms of cosine functions with different frequencies. Its extensive use in lossy compression algorithms is due to the DCT's *energy compaction* properties. The majority of a signal's energy is contained within relatively few coefficients - typically those corresponding to the lower frequency basis functions.

On a side note, the DCT comes in a various versions that have minor differences between them. In the following, we will describe the most widely used version, known as the *DCT-II*, as well as its inverse transform, the *DCT-III*. We will refer to them simply as "the DCT" and "the Inverse DCT (IDCT)", respectively.

4.1.1 One-Dimensional DCT

Formally, the DCT $\boldsymbol{w}$ of a one-dimensional signal $\boldsymbol{v}$ of length M is given by

$$w_k = c(k) \sum_{m=1}^{M} v_m \cos\left(\frac{\pi(2i-1)(k-1)}{2M}\right) \qquad k = 1, \cdots, M \tag{4.1}$$

where

$$c(k) = \begin{cases} \sqrt{\frac{1}{M}} & \text{if } k = 1 \\ \sqrt{\frac{2}{M}} & \text{otherwise} \end{cases}$$

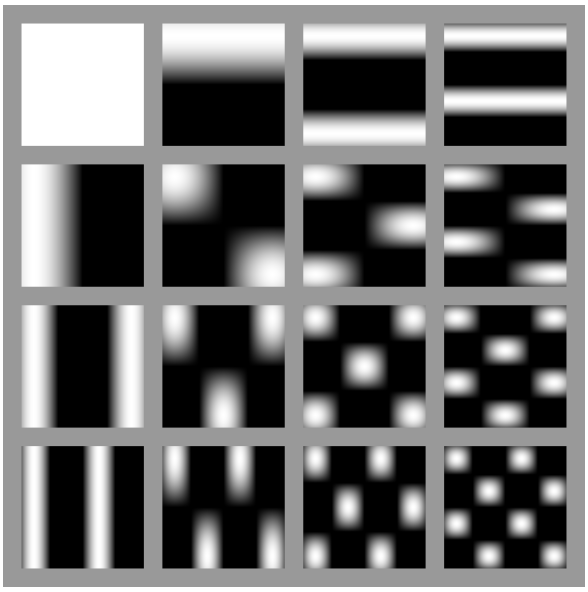


Fig. 4.2 The 2-D DCT basis functions that are used by the DCT to decompose a 4×4 image. The spatial frequency increases towards the bottom right corner.

This transforms a signal $\boldsymbol{v}$ in the original domain (time or space) into its representation $\boldsymbol{w}$ in the DCT domain.

Conversely, given a signal $\boldsymbol{w}$ in the DCT domain, we can transform it back to the orignal (time or space) domain via the IDCT defined by

$$v_n = \sum_{k=1}^{M} c(k) w_k \cos\left(\frac{\pi(2i-1)(k-1)}{2M}\right) \qquad n = 1, \cdots, M \tag{4.2}$$

We can express equation (4.2) in the desired form $\boldsymbol{v} = \boldsymbol{P}\boldsymbol{w}$. The entries of the basis matrix $\boldsymbol{P}$ are given by

$$P_{n,k} = c(k) \cos\left(\frac{\pi(2i-1)(k-1)}{2M}\right). \tag{4.3}$$

Note that the basis matrix $\boldsymbol{P}$ is orthogonal, $\boldsymbol{P}^T\boldsymbol{P} = \boldsymbol{I}_M$.

4.1.2 Multi-Dimensional DCT

Once we know how to perform the DCT on a one-dimensional signal, we can easily extend the transform to multi-dimensional signals (images, video, etc). To do so, we simply perform successive 1-D transforms along each dimension of the signal. This property is known as *seperability*.

Suppose the signal of interest is a digital image. That means that $\boldsymbol{v}$ is a $M_1 \times M_2$ matrix where $M_1 \times M_2$ is the resolution of the image. To transform the signal, we first perform the DCT on every row of the matrix. Following that, we perform the DCT on every column of the resulting matrix to get the final transformed signal.

Figure 4.1 shows an example of a 2-D signal $\boldsymbol{v}$ and its transform $\boldsymbol{w}$. Note that the majority of the energy of the transformed signal is concentrated in the top left corner. Most of the DCT coefficients are zero or very close to zero.

In Figure 4.2, we show the 2-D basis functions that would used by the DCT to decompose a signal of size 4×4. Each basis function is characterised by a horizontal and vertical spatial frequency. Typically, natural images are mostly made up of low-frequency components and the corresponding coefficients are therefore relatively large. The highest-frequency components are usually only needed to describe very fine details.

For a discussion on the properties of the DCT, see [4]

4.2 Discrete Wavelet Transform

Wavelets have become a very popular tool in signal processing. Their energy compaction properties are on par and often superior to those of the DCT for a wide range of signal classes. In 2000, the JPEG committee released a new image coding standard, JPEG2000, that is gradually replacing the original JPEG standard. The new format moved away from the DCT and uses a Discrete Wavelet Transform (DWT) instead.

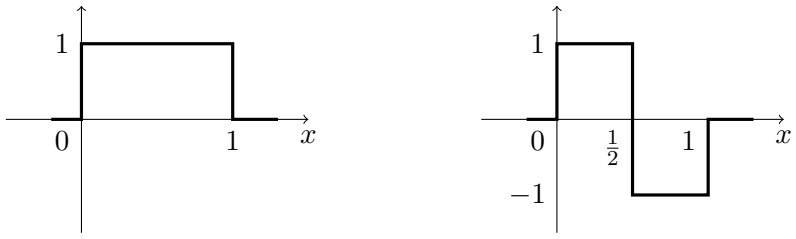
[CAVEAT and INTRO]

4.2.1 Introduction to Wavelets

To motivate wavelets, consider again the one-dimensional signal $\boldsymbol{v}$ of length M. Suppose, for simplicity, that M is a power of 2, $M = 2^q$ say. We can view the $\boldsymbol{v}$ as a piecewise-constant function $v(x)$ on the half-open interval $[0, 1)$, where $v(x) = v_i$ if $x \in [\frac{i-1}{M}, \frac{i}{M})$.

Let V^j denote the vector space containing all piecewise-constant functions f defined on the interval $[0, 1)$ that consist of 2^j pieces, each of which is constant across a sub-interval of size 2^{-j}. Thus, V^0 consists of all functions that are constant on $[0, 1)$, while V^1 consists of all functions that have two constant pieces, one over $[0, 1/2)$ and one over $[1/2, 1)$. In particular, our signal $v(x)$ resides in the space V^q.

Note that if $f \in V^j$, then $f \in V^{j+1}$. Thus, the vector spaces V^j are nested: $V^0 \subset V^1 \subset V^2 \subset \cdots$.



(a) Haar scaling function $\phi(x)$ (b) Haar mother wavelet $\psi(x)$

Fig. 4.3 The scaling function and wavelet function for the Haar wavelets.

Next, we need to choose a basis for each vector space V^j. To do so, we introduce a *scaling function* (also known as *scalet*, or *father wavelet*) that is usually denoted $\phi(x)$. The form of the scaling function depends on the particular choice wavelet decomposition.

For example, for the *Haar wavelets* the scaling function is given by

$$\phi(x) = \begin{cases} 1 & \text{if } 0 \leq x < 1 \\ 0 & \text{otherwise} \end{cases} \tag{4.4}$$

See Figure 4.3a for a plot of $\phi(x)$.

Given the scaling function $\phi(x)$, we can define the following basis for V^j:

$$\phi_k^j(x) := 2^{j/2}\phi(2^j x - k) \qquad k = 0, \cdots, 2^j - 1 \tag{4.5}$$

Using this basis, we can decompose our signal $v(x) \in V^q$ as

$$v(x) = \sum_{k=0}^{2^q-1} c_k^q \phi_k^q(x)$$

For the scaling function defined in equation (4.4), we have that $c_k^q = v_{k+1}$.

To obtain *wavelets*, consider the *orthogonal complement* of V^j in V^{j+1} and denote it W^j. That is, $W^j = \{f \in V^{j+1} : \quad \langle f, g\rangle = 0 \quad \forall g \in V^j\}$ where the inner product $\langle f, g\rangle$ is given by

$$\langle f, g\rangle = \int_0^1 f(x)g(x)dx.$$

By forming a basis for W^j, we obtain a set of *wavelet functions* $\{\psi_k^j,\ k = 0, \cdots, 2^j - 1\}$. Wavelet functions can be constructed by scaling and shifting a so-called *mother wavelet*

$\psi(x)$ as follows:

$$\psi_k^j(x) = 2^{j/2}\psi(2^j x - k) \qquad k = 0, \cdots, 2^j - 1 \tag{4.6}$$

For the Haar wavelets, the mother wavelet is given by:

$$\psi(x) = \begin{cases} 1 & 0 \le x < 1/2 \\ -1 & 1/2 \le x < 1 \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

The Haar mother wavelet is shown in Figure 4.3b.

Note that, since the scaling functions ϕ_k^j form a basis of V^j and the wavelet functions ψ_k^j form a basis of W^k, and since W^j is the orthogonal complement to V^j in V^{j+1}, it follows that the set $\{\phi_k^j, \psi_k^j : k = 0, \cdots, 2^j - 1\}$ forms a basis of the vector space V^{j+1}.

This allows us to express our signal $v \in V^q$ as

$$v(x) = \sum_{k=0}^{2^{q-1}-1} d_k^{q-1}\psi_k^{q-1}(x) + \sum_{k=0}^{2^{q-1}-1} c_k^{q-1}\phi_k^{q-1}(x)$$

This gives us the first level of the discrete wavelet transform of v. The coefficients c_k and d_k are sometimes referred to as "approximation" coefficients and "detail" coefficients, respectively.

We can continue the decomposition by splitting the basis for V^{q-1} into the bases for V^{q-2} and W^{q-2} to get the next level of the transform:

$$v(x) = \sum_{k=0}^{2^{q-1}-1} d_k^{q-1}\psi_k^{q-1}(x) + \sum_{k=0}^{2^{q-2}-1} d_k^{q-2}\psi_k^{q-2}(x) + \sum_{k=0}^{2^{q-2}-1} c_k^{q-2}\phi_k^{q-2}(x)$$

To get the full decomposition, we continue in this fashion up to the qth level:

$$v(x) = \sum_{j=0}^{q-1}\sum_{k=0}^{2^j-1} d_k^j\psi_k^j(x) + c_0^0\phi(x)$$

The full DWT of $\boldsymbol{v}$ consists of the coefficients $\{c_0^0, d_k^j : j = 0, \cdots, q-1,\ k = 0, \cdots, 2^j - 1\}$.

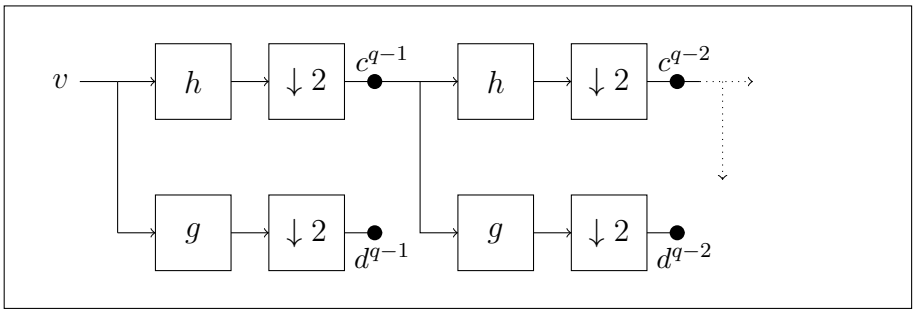


Fig. 4.4 The first two levels of the DWT of the signal $\boldsymbol{v}$ of length 2^q via a filter bank.

4.2.2 Computing the DWT

In practice, we can compute one level of the DWT coefficients by passing the signal v through a *low-pass filter* h and a high-pass filter g, respectively, and then downsampling the results by a factor of two. Passing v through the filters h and g results in the *convolutions*:

$$(v \star h)(x) = \sum_{k=-\infty}^{\infty} v(k)h(x-k)$$
$$(v \star g)(x) = \sum_{k=-\infty}^{\infty} v(k)g(x-k)$$

To downsample by a factor of two, we remove every second sample:

$$(v \downarrow 2)(x) := v(2x)$$

Overall, these computations can be done by multiplying the vector $\boldsymbol{v}$ by a matrix $\boldsymbol{H}$ and a matrix $\boldsymbol{G}$ to get the approximation and detail coefficients, repectively.

To compute the next level, we take the approximation coefficients of the current stage and pass them again through the filter bank. The procedure is depicted in Figure 4.4.

The coefficients of the filters $h(x)$ and $g(x)$ depend on our choice of the scaling function $\phi(x)$ and mother wavelet function $\psi(x)$. They satisfy the following relations:

$$\phi(x) = \sqrt{2} \sum_{k=-\infty}^{\infty} h(k)\phi(2x-k)$$
$$\psi(x) = \sqrt{2} \sum_{k=-\infty}^{\infty} g(k)\phi(2x-k)$$

as well as

$$g(k) = (-1)^k h(1-k).$$

For the Haar wavelets (4.4,4.7), the filter coefficients are

$$\begin{aligned} h(0) = h(1) &= \frac{1}{\sqrt{2}}, \qquad h(k) = 0 \text{ if } k \neq 0, 1 \\ g(0) = -g(1) &= \frac{1}{\sqrt{2}}, \qquad g(k) = 0 \text{ if } k \neq 0, 1 \end{aligned}$$

The first level approximation coefficients of the Haar DWT of signal $\boldsymbol{v} \in \mathbb{R}^M$, with $M = 2^q$, are thus given by $\boldsymbol{H}^q \boldsymbol{v}$ where $\boldsymbol{H}^q$ is the $2^{q-1} \times 2^q$ matrix given by

$$\boldsymbol{H}^q_{haar} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & 1 \end{bmatrix} \tag{4.8}$$

To get the corresponding detail coefficients, we multiply $\boldsymbol{v}$ by the $2^{q-1} \times 2^q$ matrix $\boldsymbol{G}^q$ given by

$$\boldsymbol{G}^q_{haar} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & -1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & -1 \end{bmatrix} \tag{4.9}$$

Overall to perform the discrete Haar wavelet transform at the first scale, we multiply $\boldsymbol{v}$ by

$$\boldsymbol{P}^{(1)}_{haar} = \begin{bmatrix} \boldsymbol{H}^q \\ \boldsymbol{G}^q \end{bmatrix}$$

where we dropped *haar*-subscript on $\boldsymbol{H}$ and $\boldsymbol{G}$ for ease of viewing.

To compute the transform at the second scale, we replace $\boldsymbol{H}^q$ by $\boldsymbol{H}^{q-1}\boldsymbol{H}^q$ and $\boldsymbol{G}^{q-1}\boldsymbol{h}^q$:

$$\boldsymbol{P}^{(2)}_{haar} = \begin{bmatrix} \boldsymbol{H}^{q-1}\boldsymbol{H}^q \\ \boldsymbol{G}^{q-1}\boldsymbol{H}^q \\ \boldsymbol{G}^q \end{bmatrix}$$

We can continue this process until the qth scale to get the full Haar wavelet transform matrix:

$$\boldsymbol{P}_{haar}^{(q)} = \begin{bmatrix} \boldsymbol{H}^1\boldsymbol{H}^2\cdots\boldsymbol{H}^{q-1}\boldsymbol{H}^q \\ \boldsymbol{G}^1\boldsymbol{H}^2\cdots\boldsymbol{H}^{q-1}\boldsymbol{H}^q \\ \vdots \\ \boldsymbol{G}^{q-2}\boldsymbol{H}^{q-1}\boldsymbol{H}^q \\ \boldsymbol{G}^{q-1}\boldsymbol{H}^q \\ \boldsymbol{G}^q \end{bmatrix}$$

To form the basis matrix $\boldsymbol{\Psi}$ such that $\boldsymbol{v} = \boldsymbol{\Psi}\boldsymbol{w}$ corresponding to the stage j DWT, we can invert the matrix $\boldsymbol{P}_{haar}^{(j)}$:

$$\boldsymbol{\Psi} = \left(\boldsymbol{P}_{haar}^{(j)}\right)^{-1} \tag{4.10}$$

In practice, one usually works with wavelet functions that lead to orthogonal transform matrices, so that $\left(\boldsymbol{P}^{(j)}\right)^{-1} = \left(\boldsymbol{P}^{(j)}\right)^{T}$. The Haar wavelets satisfy this orthoganility condition. Therefore, the basis matrix $\boldsymbol{\Psi}$ from equation (4.10) can also be obtained as follows:

$$\boldsymbol{\Psi} = \left(\boldsymbol{P}_{haar}^{(j)}\right)^{T}. \tag{4.11}$$

[CAVEAT boundary conditions]

4.2.3 Two-Dimensional Haar Wavelets

In Figure 4.3, we showed the 1-dimensional Haar scaling function $\phi(x)$ and wavelet function $\psi(x)$. In general, we can define the 2-dimensional scaling wavelet functions by taking the products of their one-dimensional versions:

$$\begin{aligned} \phi(x,y) &= \phi(x)\phi(y) \\ \psi^H(x,y) &= \phi(x)\psi(y) \\ \psi^V(x,y) &= \psi(x)\phi(y) \\ \psi^D(x,y) &= \psi(x)\psi(y) \end{aligned} \tag{4.12}$$

Where the superscripts H, V and D refer to the high-pass filters in the horizontal, vertical and diagonal direction, respectively. These functions are illustrated in Figure 4.5 for the Haar wavelet system. The 2-D Haar wavelet functions act as *edge detectors* in their respective directions. For example, if we pass a 2×2 image patch through the

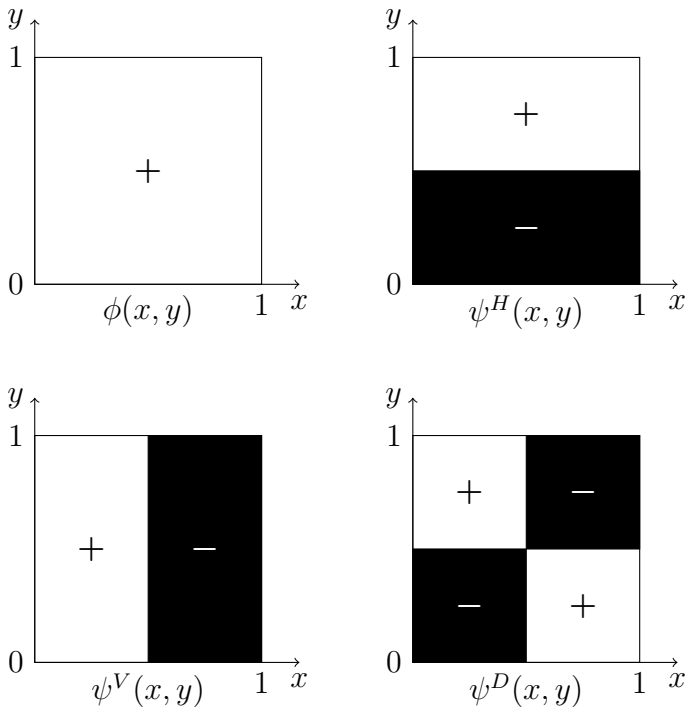
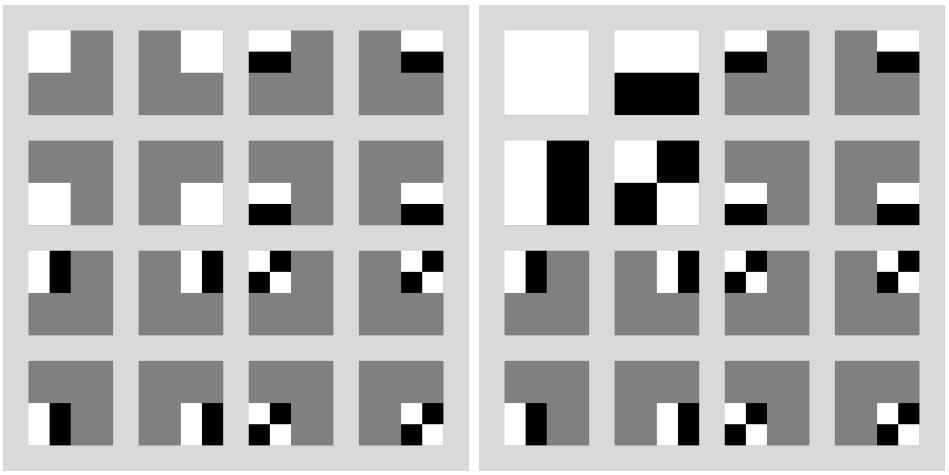


Fig. 4.5 The 2-D Haar scaling function and wavelet functions. Inside the $[0, 1] \times [0, 1]$ square, white corresponds to $+2$ and black corresponds to -1. The functions are zero outside the unit square.



(a) Scale 1 (b) Scale 2

Fig. 4.6 The haar basis functions at the first scale (a) and the second scale (b). The DWT would use these basis functions to decompose a 4×4 image.

ψ^V filter, the corresponding detail coefficient will be small unless there is sharp change in pixel intensities between the left side and right side of the patch, i.e. unless there is a vertical edge.

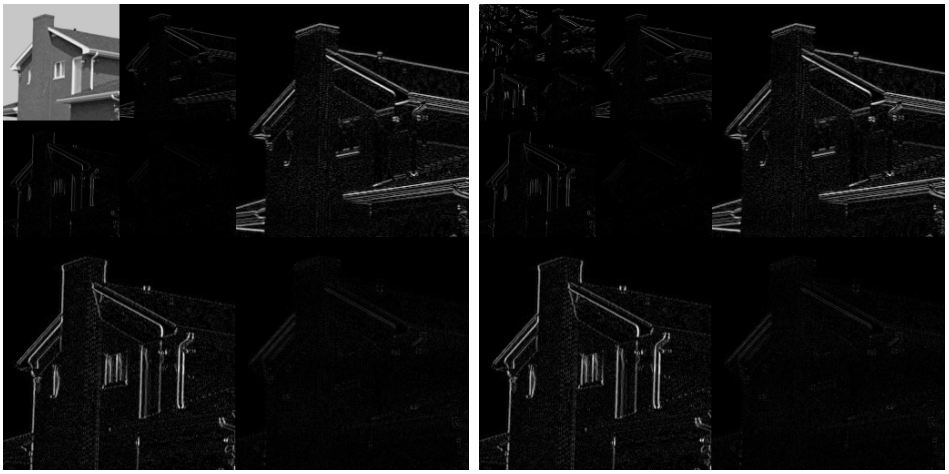
The wavelet basis functions for scale j can be formed in two different ways. In the so-called "standard" construction [7], we first compute the DWT for scale j for each column, followed by computing the scale-J DWT on the intermediate result for each row.

The non-standard construction, on the other hand, builds up the scale-j iteratively transforming the columns and the rows. First, we perform the standard construction, but only for the first scale. Next, we do another level-1 DWT, but only on the approximation coefficients of the previous stage. We repeat this process j times to get the level-j DWT.

In this document, we will only consider the non-standard construction of the DWT basis. The reason is that the basis functions resulting from the non-standard construction always have square support and to keep in line with [5].

In Figure 4.6, we illustrate the basis functions resulting from the non-standard construction. We show two sets of basis functions that can be used by the Haar DWT to decompose a signal of size 4×4, one corresponding to the first scale and another set corresponding to the level 2 decomposition. The DWT of a digital image is illustrated at various levels in Figure 4.7. One can clearly see that the high-pass filters corresponding to the Haar wavelets act as edge detectors.

(a) Original 512×512 image (b) Scale 1 DWT



(c) Scale 2 DWT (d) Scale 9 DWT

Fig. 4.7 The 2-D DWT using Haar wavelets of a digital image (a) of size 512×512. We compute the DWT at the first and the second scale (b,c) as well as the full DWT (d). In (d), we have a single approximation coefficient at the top left corner. In (b,c,d), the brightness of a pixel increases monotonically with the size of the corresponding DWT coefficient. We have enhanced the detail coefficients (relative to the approximation coefficients) for better visibility.

4.3 Basis Matrix for 2-D and 3-D Signals

So far, we have displayed two-dimensional signals as matrices and three-dimensional signals as 3-D arrays (also known as "3-D matrices", third-order tensors, or "cubes"). To reconcile this with the notation used in Chapter 2, we need to *vectorize* the multi-dimensional signal $\boldsymbol{v}$. More importantly, the RVM in Chapter 3 is agnostic of the dimensionality of the signal. It takes as input a target vector $\boldsymbol{y}$ and a design matrix $\boldsymbol{\Phi}$ and outputs a weights vector $\boldsymbol{w}$. Any information about the structur of a signal (such as its dimensionality) is encoded in the design matrix. Therefore, it is important that we vectorize our multi-dimensional signals and also form the basis matrix $\boldsymbol{Psi}$ from the transform matrices (corresponding to the DCT/DWT/etc) in a way that is consistent with the vectorization.

We will define the particular vectorization scheme that was used in our algorithm and describe how we built the basis matrices from the specific (inverse) transform matrices.

4.3.1 Basis Matrix for One-Dimensional Signals

We start with 1-D signals. This case is trivial, since a one-dimensional signal $\boldsymbol{v}$ can directly be treated as a vector in $\mathbb{R}^M$, where M is the length of the signal.

The basis matrix matrix, in this case, is simply the inverse transform matrix. So, for the DCT, the basis matrix is $\boldsymbol{\Psi} = \boldsymbol{P}$, where $\boldsymbol{P}$ is defined in equation (4.3). We have defined the basis matrix corresponding to the discrete Haar wavelet transform in equation (4.11).

4.3.2 Basis Matrix for Two-Dimensional Signals

Suppose $\boldsymbol{v}$ is a two-dimensional signal of size $r \times c$ (i.e. r rows and c columns). Suppose further that r and c are powers of 2, $r = 2^{q_1}$ and $c = 2^{q_2}$, say.

We vectorize $\boldsymbol{v}$ in a row-major fashion. To be explicit, let $\boldsymbol{v}_j^T$ be the jth row of $\boldsymbol{v} \in \mathbb{R}^{r \times c}$. The vectorized form of $\boldsymbol{v}$ is then given by

$$\boldsymbol{v}^V = \begin{bmatrix} \boldsymbol{v}_1 \\ \boldsymbol{v}_2 \\ \vdots \\ \boldsymbol{v}_r \end{bmatrix}$$

where we use the V superscript to indicate the vectorized signal. Thus, we end up with a long vector $\boldsymbol{v}^V \in \mathbb{R}^{rc}$.

To form the basis matrix corresponding to the two-dimensional DCT, we use the DCT's seperability property. Let $\boldsymbol{\Psi}_r$ and $\boldsymbol{\Psi}_c$ be the DCT basis matrices corresponding to one-dimensional signals of length r and c, respectively. The basis matrix corresponding to $\boldsymbol{v}^V$ is then given by forming the *kronecker product* of $\boldsymbol{\Psi}_c$ and $\boldsymbol{\Psi}_r$:

$$\boldsymbol{\Psi} = \boldsymbol{\Psi}_c \otimes \boldsymbol{\Psi}_r$$

The kronecker product $P \otimes Q$ between matrices P and Q with dimensions $m_P \times n_P$ and $m_Q \times n_Q$, respectively, is defined to be the block matrix

$$\begin{bmatrix} P_{1,1}Q & P_{1,2}Q & \cdots & P_{1,n_P}Q \\ P_{2,1}Q & P_{2,2}Q & \cdots & P_{2,n_P}Q \\ \vdots & \vdots & \ddots & \vdots \\ P_{m_P,1}Q & P_{m_P,2}Q & \cdots & P_{m_P,n_P}Q \end{bmatrix} \tag{4.13}$$

of size $(m_P m_Q) \times (n_P n_Q)$.

For the 2-D Haar DWT at the first scale, the basis matrix is constructed in a similar fashion. Let $\boldsymbol{H}^{q_1}$ and $\boldsymbol{H}^{q_2}$ be defined according to equation (4.8), and let $\boldsymbol{G}^{q_1}$ and $\boldsymbol{G}^{q_2}$ be formed according to (4.9) (where we dropped the *haar* subscript. The basis matrix is constructed as follows:

$$\boldsymbol{\Psi} = \begin{bmatrix} \boldsymbol{H}^{q_2} \otimes \boldsymbol{H}^{q_1} \\ \boldsymbol{H}^{q_2} \otimes \boldsymbol{G}^{q_1} \\ \boldsymbol{G}^{q_2} \otimes \boldsymbol{H}^{q_1} \\ \boldsymbol{G}^{q_2} \otimes \boldsymbol{G}^{q_1} \end{bmatrix}^T .$$

For a signal of size $2^{q_1} \times 2^{q_2}$, we can build a cascade up to the qth scale, where $q = \min\{q_1, q_2\}$. The resulting basis matrix is given by

$$\boldsymbol{\Psi} = \begin{bmatrix} (\boldsymbol{H}^{q_2-q}\boldsymbol{H}^{q_2-q+1}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{H}^{q_1-q}\boldsymbol{H}^{q_1-q+1}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{H}^{q_2-q}\boldsymbol{H}^{q_2-q+1}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{G}^{q_1-q}\boldsymbol{H}^{q_1-q+1}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{G}^{q_2-q}\boldsymbol{H}^{q_2-q+1}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{H}^{q_1-q}\boldsymbol{H}^{q_1-q+1}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{G}^{q_2-q}\boldsymbol{H}^{q_2-q+1}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{G}^{q_1-q}\boldsymbol{H}^{q_1-q+1}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{H}^{q_2-q+1}\boldsymbol{H}^{q_2-q+2}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{G}^{q_1-q+1}\boldsymbol{H}^{q_1-q+2}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{G}^{q_2-q+1}\boldsymbol{H}^{q_2-q+2}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{H}^{q_1-q+1}\boldsymbol{H}^{q_1-q+2}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{G}^{q_2-q+1}\boldsymbol{H}^{q_2-q+2}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{G}^{q_1-q+1}\boldsymbol{H}^{q_1-q+2}\cdots\boldsymbol{H}^{q_1}) \\ \vdots \\ (\boldsymbol{G}^{q_2-2}\boldsymbol{H}^{q_2-1}\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{G}^{q_1-2}\boldsymbol{H}^{q_1-1}\boldsymbol{H}^{q_1}) \\ (\boldsymbol{H}^{q_2-1}\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{G}^{q_1-1}\boldsymbol{H}^{q_1}) \\ (\boldsymbol{G}^{q_2-1}\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{H}^{q_1-1}\boldsymbol{H}^{q_1}) \\ (\boldsymbol{G}^{q_2-1}\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{G}^{q_1-1}\boldsymbol{H}^{q_1}) \\ \boldsymbol{H}^{q_2} \otimes \boldsymbol{G}^{q_1} \\ \boldsymbol{G}^{q_2} \otimes \boldsymbol{H}^{q_1} \\ \boldsymbol{G}^{q_2} \otimes \boldsymbol{G}^{q_1} \end{bmatrix}^T .$$

Note that the dimensions of the basis matrix are $(rc) \times (rc)$, regardless of the scale that we use.

Chapter 5

Multi-Scale Cascade of Estimations

In this chapter, we combine the Compressive Sensing framework from Chapter 2 with the Bayesian framework from Chapter 3 to form the *Bayesian Compressive Sensing* (BCS) framework.

Following that, we will discuss the *Multi-Scale Cascade of Estimations* (MSCE) algorithm.

5.1 Bayesian Compressive Sensing

In the Bayesian Compressive Sensing (BCS) [3] framework, we attempt to approach the Compressive Sensing problem from the point of view of linear regression.

Recall the setup from Chapter 2. We are intersted in recovering the signal $\boldsymbol{v} \in \mathbb{R}^M$ from the CS measurements $\boldsymbol{\Theta v} = \boldsymbol{y} \in \mathbb{R}^N$, where $N << M$. We also assume that $\boldsymbol{v}$ is compressible in the basis $\boldsymbol{\Psi}$. This means that $\boldsymbol{w} = \boldsymbol{\Psi}^T \boldsymbol{v}$ is well approximated by the K-sparse vector $\boldsymbol{w}_K$ which is identical to $\boldsymbol{w}$ for the K elements in $\boldsymbol{w}$ with the largest magnitude.

Thus, $\boldsymbol{w}_e = \boldsymbol{w} - \boldsymbol{w}_K$ is assumed to be very small in magnitude in we will effectively treat it as noise, Letting $\boldsymbol{\Phi} = \boldsymbol{\Theta\Psi}$, we have

$$\boldsymbol{y} = \boldsymbol{\Theta v} = \boldsymbol{\Theta\Psi w} = \boldsymbol{\Phi w} = \boldsymbol{\Phi w}_K + \boldsymbol{\Phi w}_e = \boldsymbol{\Phi w}_K + \boldsymbol{n}_e$$

where $\boldsymbol{n}_e = \boldsymbol{\Phi w}_e$. Moreover, there could be an additional noise source $\boldsymbol{n}$ in the CS measurements $\boldsymbol{y}$ (e.g. due to finite precision in the measurement device) so that, overall,

$$\boldsymbol{y} = \boldsymbol{\Phi w}_K + \boldsymbol{n}_e + \boldsymbol{n} = \boldsymbol{\Phi w}_K + \boldsymbol{\epsilon} \tag{5.1}$$

with $\boldsymbol{\epsilon} = \boldsymbol{n}_e + \boldsymbol{n}$.

In the BCS literature [3], $\boldsymbol{\epsilon}$ is typically approximated as zero-mean Gaussian noise variable with an unknown variance σ^2:

$$\boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I}_N). \tag{5.2}$$

Although this assumption may not always be completely accurate in practice, it is often used for its desirable analytical properties. We might try to justify it by noting that the sensing matrix $\boldsymbol{\theta}$ is usually constructed randomly (often using Gaussian samples) and that the noise source $\boldsymbol{n}$ in the CS measurements is typically represented by a zero-mean Gaussian distribution.

Note the similarity between equations (5.1) and (5.2) and the geneneral linear regression model in equations (3.4) and (3.3). The only difference is that, in equation (5.1), we have the addtional constraint that $\boldsymbol{w}_K$ is *sparse*.

Since in the BCS framework, we wish to solve the CS problem from a Bayesian standpoint, we are interested in computing a full posterior distribution of the weights $\boldsymbol{w}_K$ and the noise variance σ^2. The sparsity constraint can be entered into the model by placing a sparseness-inducing prior distribution over the weights $\boldsymbol{w}_K$.

There is a range of popular sparseness priors. Here, we choose the hierarchical prior that was specified in Chapter 3, equations (3.6), (3.7) and (3.8). This allows us to use the Sparse Bayesian Learning framework to invert the CS system.

More explicitly, we view the CS measurements $\boldsymbol{y} \in \mathbb{R}^N$ as a target vector and $\boldsymbol{\Phi} = \boldsymbol{\Theta}\boldsymbol{\Psi} \in \mathbb{R}^{N \times M}$ as a design matrix and use the Sequential Sparse Bayesian Learning Algorithm (Algorithm 2) to compute the posterior distribution of the weights $\boldsymbol{w} \in \mathbb{R}^M$ (3.13). The desired signal $\boldsymbol{v} \in \mathbb{R}^M$ can then by recovered by computing

$$\hat{\boldsymbol{v}} = \boldsymbol{\Psi}\boldsymbol{\mu} \tag{5.3}$$

where $\boldsymbol{\mu}$ is the posterior mean of $\boldsymbol{w}$.

[ADVANTAGES OVER DCS?, PERFORMANCE?]

5.2 Image Interpolation

The BCS framework of the previous section was used by [5] to reconstruct images that have missing pixel values, such as in Figure 5.1.

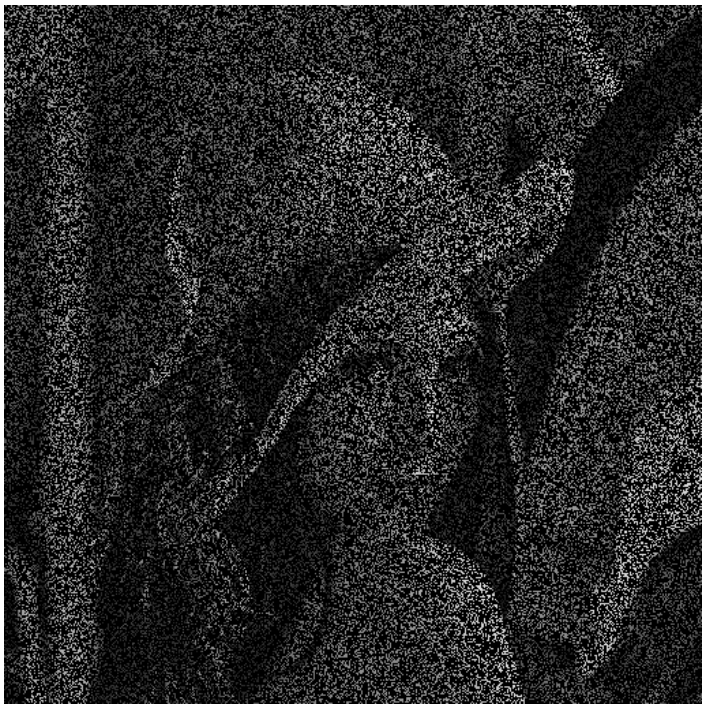


Fig. 5.1 Example of an image with missing pixel values. The missing data points are set to zero and displayed in black

In the context of Compressive Sensing, this problem corresponds to a sensing mechanism that directly measures a down-sampled version of the original signal $\boldsymbol{v}$. In Section 2.3.1 we explained how to construct corresponding sensing matrix $\boldsymbol{\Theta}$. We will refer to such sensing matrices as "masks". The problem of reconstructing such masked signals is sometimes also referred to as "interpolation".

In the BCS framework, we recover the original signal according to (5.3). If $\boldsymbol{\theta}$ is a mask, the design matrix $\boldsymbol{\Phi}$ that is passed to the RVM consists of the basis matrix $\boldsymbol{\Psi}$ but with the rows corresponding to missing pixel values removed. The recovery strategy that we discussed is thus equivalent to using the RVM directly to predict the pixel values at *all* the locations according to equation (3.27).

Since we have knowledge of the true pixel values for a subset of these locations, we can improve the reconstruction by replacing the predicted values with the original data when possible.

5.2.1 Reconstruction using Haar Wavelets

Using the BCS model from above with a Haar basis matrix, we reconstruct the image in Figure 5.1. We display the output in Figure 5.2 for various scales of the Haar basis.

(a) Scale 1: PSNR = 11.75 (b) Scale 2: PSNR = 21.78

(c) Scale 3: PSNR = 24.15 (d) Scale 4: PSNR = 23.72

Fig. 5.2 Reconstruction if the image in Figure 5.1 using the RVM with Haar wavelets at various scales. We use the *Peak Signal-to-Noise Ratio (PSNR)* as our performance metric. The larger the PSNR, the more accurate the reconstruction. See Chapter 8 for a definition of the PSNR.

We see that the scale of the wavelet basis has a strong effect on the quality of the reconstruction. Wavelets at lower scales can fail to reconstruct the entire image, but generally give a more accurate reconstruction at the portion of the image where they succeed. On the other, wavelets at larger scales typically achieve a reconstruction of the entire image at the cost of increased blurring or pixelation. This conflict can be seen in Figure 5.2 by contrasting panel (c) with panel (d): They both get a reconstruction of the entire image, but scale 3 actually outperforms scale 4 in this example. Moreover, if we compare panel (b) with panel (d), we see that, in the part of the image where the recovery succeeds, the scale 2 reconstruction manages to recover finer details than both scale 3 and scale 4.

To explain why this tradeoff exists, recall that two-dimensional Haar wavelets at scale s have a support of size up to $2^s \times 2^s$. Thus, at small scales, the support of the individual basis functions is small. For example, suppose $s = 1$. In this case, each Haar basis functions covers an area of 2×2 pixels (see Figure 4.6). Using these wavelets, we can capture very local relationships between the pixels. This allows us to recover finer details in the reconstruction.

However, it can also lead to the formation of black pixels. At scale 1, each column j of the Haar basis matrix $\boldsymbol{\Psi}$ contains exactly 4 non-zero entries corresponding to the four pixel locations that are covered by the jth basis function. If the masked image happens to be missing data at all four of these locations, the corresponding rows of the basis matrix $\boldsymbol{\Psi}$ will be deleted when forming the design matrix $\boldsymbol{\Phi} = \boldsymbol{\Theta}\boldsymbol{\Psi}$. Therefore, column j of the design matrix will be zero. The same is true for the three columns that correspond to the remaining locations that were covered by basis function j.

The Sequential Sparse Bayesian Learning Algorithm will generally not select any columns that are conpletely zero, since they offer no change to marginal likelyhood. The consequence is that the corresponding entries in the posterior mean $\boldsymbol{\mu}$ of $\boldsymbol{w}$ will remain zero. The resulting reconstruction (5.3) will therefore be unable to recover the 2×2 patch covered by basis function j.

This problem can be mitigated by using larger scales since, for any given number of missing pixels, it is less likely that larger square patches of the masked image are completely void of data. However, the SSBL Algorithm generally prefers to add basis functions with larger support to the model, since they typically cause a larger increase in the Marginal Likelihood. The result is that we get a more smoothed or blurred reconstruction and use the finer details, even in parts of the image that would have been accurately recovered at smaller scales.

Algorithm 3 Multi-Scale Cascade of Estimations [5]

1: Set $s = 1$
2: Let $\boldsymbol{y}_1 = \boldsymbol{y}$
3: **while** $s \leq s_{max}$ **do**
4: Form basis matrix $\boldsymbol{\Psi}$ using Haar wavelets at scale s
5: Call Sequential Sparse Bayesian Learning Algorithm with target vector $\boldsymbol{y}_j$ and design matrix $\boldsymbol{\Phi} = \boldsymbol{\Theta}\boldsymbol{\Psi}$ to get the estimate $\hat{\boldsymbol{v}}_s$
6: Compute $(\sigma^2)^*$ for all newly reconstructed pixel values using equation (5.4)
7: If $(\sigma^2)^* = \sigma^2$ or $(\sigma^2)^* > \tau$, discard the corresponding estimate
8: Set $s = s + 1$
9: Let $\boldsymbol{y}_s$ be $\hat{\boldsymbol{v}}_s$ after the discarded estimates are deleted
10: **end while**
11: **return** $\boldsymbol{v} = \boldsymbol{y}_s$

5.3 Multi-Scale Cascade of Estimations Algorithm

How can we recover the finer details of the image, while also obtaining a full reconstruction? [5] addressed this problem by using the RVM error bars to form a cascade of RVM estimations. The resulting algorithm is called *Multi-Scale Cascade of Estimations* (MSCE).

It begins by using the procedure from the previous section with the Haar basis matrix at the lowest scale to obtain a reconstruction of the masked image. Next the error bars are computed for the predictions at the locations $\boldsymbol{x}^*$ of the missing pixel values:

$$(\sigma^2)^* = \sigma^2 + \boldsymbol{\psi}(\boldsymbol{x}^*)^T \boldsymbol{\Sigma} \boldsymbol{\psi}(\boldsymbol{x}). \tag{5.4}$$

The MSCE algorithm uses the error measure $(\sigma^2)^*$ to decide whether to keep the estimated pixel value for location $\boldsymbol{x}^*$, or whether discard and keep $\boldsymbol{x}^*$ marked as missing. If $(\sigma^2)^*$ is small, but larger than the noise variance σ^2, we trust the estimate and accept it. If $(\sigma^2)^*$ is large, we do not trust the estimate and reject it. Finally, if $(\sigma^2)^*$ is equal to σ^2, then $\boldsymbol{\psi}(\boldsymbol{x}^*)^T \boldsymbol{\Sigma} \boldsymbol{\psi}(\boldsymbol{x}^*) = 0$. This means that $\boldsymbol{\psi}(\boldsymbol{x}^*) = 0$ since $\boldsymbol{\Sigma}$ is positive definite. Thus, the RVM predicted the pixel value at location $\boldsymbol{x}^*$ to be zero. Since the recovery of the pixel was unsuccessful in this case, we reject the estimate.

If the resulting reconstruction still has missing pixel values, we pass it to the next stage of the cascade which uses a Haar basis at the next highest scale (2 in this case).

This process it repeated until we achieve a reconstruction of the entire image or until some pre-defined maximum for the scale of the Haar basis. We have summarized the MCSE in Algorithm 3.

Chapter 6

Video Reconstruction

In this chapter, we explain how to use the MSCE algorithm in [5] to reconstruct masked video signals.

Throughout this chapter, we assume that $\boldsymbol{v}$ is a video signal of size $r \times c \times f$ (r rows, c columns and f frames), where $r = 2^{q_1}$, $c = 2^{q_2}$ and $f = 2^{q_3}$.

First, we explain how to extend the basis functions that were discussed in Chapter 4 to three dimensions.

6.1 Three-Dimensional Basis Functions

6.1.1 Haar Wavelets for Videos

A straightforward way of approaching the Discrete Wavelet Transform of a video would be to simply perform the two-dimensional DWT on each individual frame. This "pseudo 3D" approach is relatively simple, but typically also very inefficient. Usually, we expect there to be continuity between successive frames of a video. The pseudo 3D approach does not allow us to exploit any of the temporal correlations that are common in video signals.

A better approach would be to perform a full 3-D wavelet decomposition using three-dimensional scaling and wavelet functions. In 3-D, we have one scaling function $\phi(x, y, t)$ and a total of $2^3 - 1 = 7$ wavelet functions. These can be obtained by multiplying the 2-D functions in (4.12) with the scaling function $\phi(t)$ and the wavelet

function $\psi(t)$ in the time domain:

$$\begin{aligned}
\phi(x,y,t) &= \phi(t)\phi(x)\phi(y) \\
\psi^H(x,y,t) &= \phi(t)\phi(x)\psi(y) \\
\psi^V(x,y,t) &= \phi(t)\psi(x)\phi(y) \\
\psi^D(x,y,t) &= \phi(t)\psi(x)\psi(y) \\
\psi^T(x,y,t) &= \psi(t)\phi(x)\phi(y) \\
\psi^{HT}(x,y,t) &= \psi(t)\phi(x)\psi(y) \\
\psi^{VT}(x,y,t) &= \psi(t)\psi(x)\phi(y) \\
\psi^{DT}(x,y,t) &= \psi(t)\psi(x)\psi(y)
\end{aligned}$$

We use the T superscript to denote the high-pass filters in the temporal direction.

Figure 6.1 shows the 3-D scaling and wavelet functions for the Haar wavelets. To obtain the basis functions corresponding to the 3-D Haar wavelet domain, we compute scaled and shifted versions of these functions as in:

$$\phi^j_{k_1,k_2,k_3}(x,y,t) = 2^{3j/2}\phi(2^j t - k_3)\phi(2^j x - k_1)\phi(2^j y - k_2)$$

At the first scale, the 3-D wavelet decomposition of a video results in 8 channels: one corresponding to a low-pass filter, and 7 corresponding to high-pass filters in various directions (see Figure 6.2).

For the Haar wavelets, the high-pass filters act as edge detectors, just like in the 2-D case. However, on top of detecting edges along spatial directions, we also get high-pass filters that detect sudden changes in the temporal direction, i.e. motion detectors.

As an example, consider Figure 6.3. In panels (a) and (b), we show two adjacent frames of the "soccer" test video. We perform the 3-D Haar DWT at the first scale and arrange the coefficients as in Figure 6.2. At the first scale, the 3-D Haar basis functions only have support over a $2 \times 2 \times 2$ volume. Thus, all the DWT coefficients that are affected by the data in frames 11 and 12 of the original video, are contained within two "frames", or "slices", of the transformed signal. For frames 11 and 12 of the original 64-frame video, these slices are at positions $12/2 = 6$ and $(6 + 64)/2 = 38$ in the transformed video.

In panel (c), we see the spatial edge detectors (the LLH, LHL and LHH channels) that are similar to those in the 2-D case (Figure 4.7). The top left corner corresponds to the low-pass filter LLL and can be regarded as an "average" of the video.

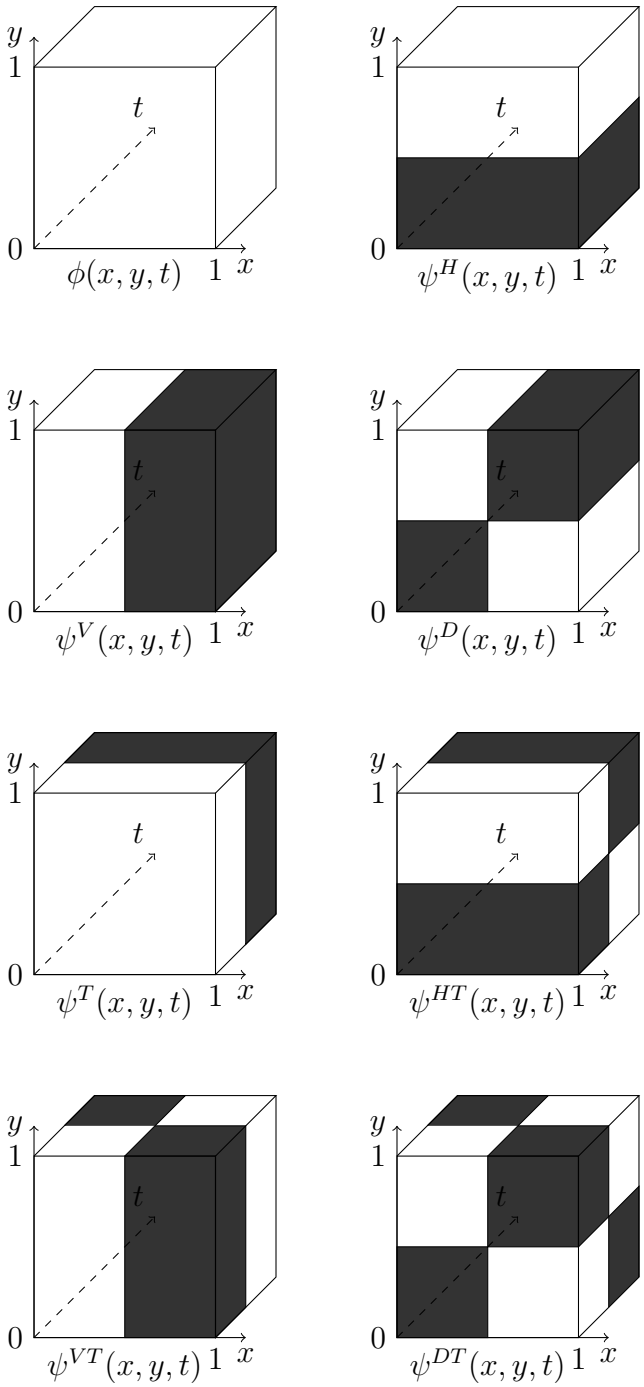


Fig. 6.1 The 3-D Haar scaling function and wavelet functions. Inside the $[0,1] \times [0,1] \times [0,1]$ cube, white corresponds to $+1$ and black corresponds to -1. The functions are zero outside of the cube.

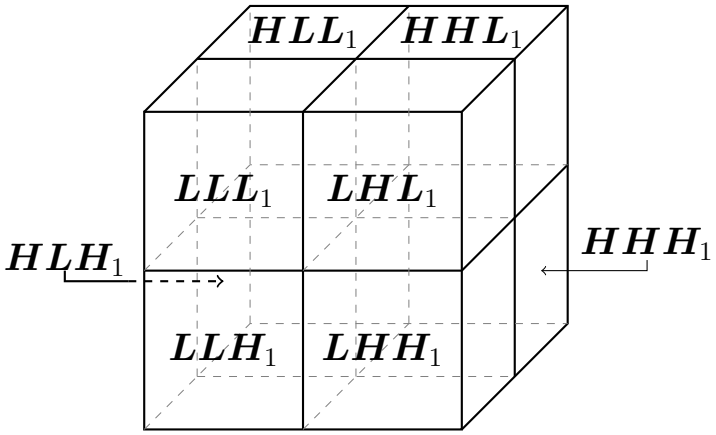
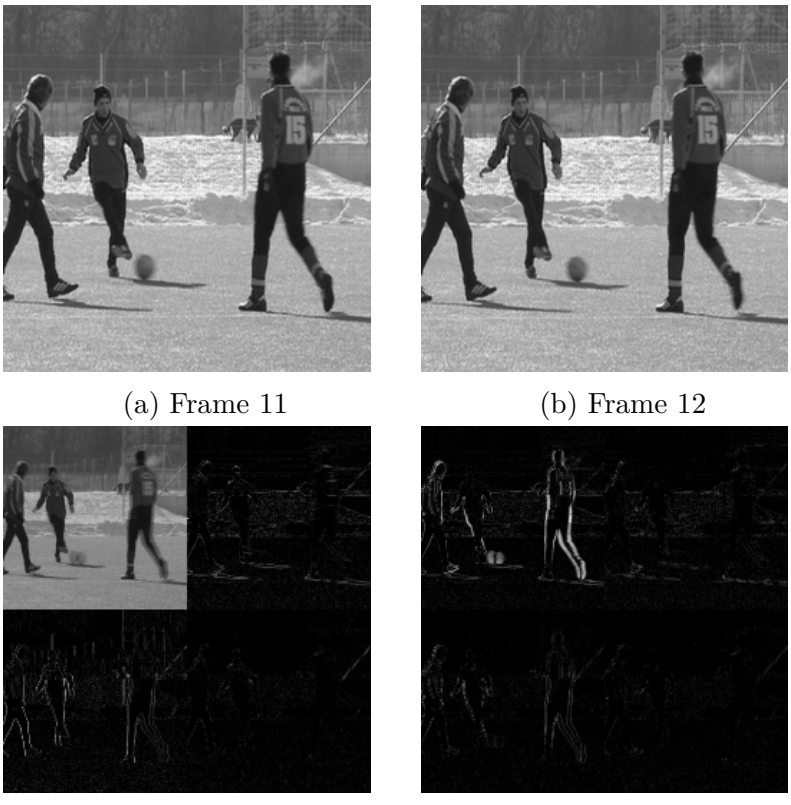


Fig. 6.2 The wavelet decomposition of a three-dimensional signal $v(x, y, t)$ at the first scale. L stands for "low" and H stands for "high". To obtain the decomposition at higher scales, we recursively decompose the low-pass channels $\boldsymbol{LLL}_j$ in a similar way.



(a) Frame 11 (b) Frame 12

(c) Scale 1 DWT: "frame" 6 (d) Scale 1 DWT: "frame" 38

Fig. 6.3 Frames 11 and 12 of a 64-frame video with a resolution of 256×256 (a,b). We perform the three-dimensional DWT at the first scale using Haar wavelets and show two "slices" of the resulting coefficients (c,d). The detail coefficients have been enhanced to improve visibility.

Panel (d), shows the four channels associated with a high-pass filter in the temporal direction. The HLL channel (top left corner of panel (d)) picks up overall differences in adjacent frames. In this example, we note that it detects the motion of the football players and the ball, (as well as a slight motion of the backgrund due to the panning of the camera). The remaining channels in panel (d) combine motion detection with edge detection.

6.1.2 Three-Dimensional DCT

The seperability property of the DCT makes the extension to three dimensions straightforward. To compute the DCT of a video, we first compute the 2-D DCT for each individual frame of the video followed by a 1-D DCT across the temporal axis for each pixel. The DCT basis functions (4.1) in 3-D are characterised by a horizontal and vertical spatial frequency, as well as an additional temporal frequency component.

6.2 Basis Matrix for Three-Dimensional Signals

We vectorize $\boldsymbol{v}$ by first vectorizing each individual frame in a row-major fashion (see Section 4.3.2), followed by stacking the vectorized frames on top of each other. The result $\boldsymbol{v}^V$ is a long vector of length rcf.

The basis matrix corresponding to the DCT is given by

$$\boldsymbol{\Psi} = \boldsymbol{\Psi}_f \otimes \boldsymbol{\Psi}_c \otimes \boldsymbol{\Psi}_r$$

where $\boldsymbol{\Psi}_r$, $\boldsymbol{\Psi}_c$ and $\boldsymbol{\Psi}_f$ are the DCT basis matrices for 1-D signals (4.3) of length r, c and f, respectively. See equation (4.13) for a definition of the Kronecker product $\otimes$.

The Haar basis matrix for video signals at scale 1 is constructed as follows:

$$\boldsymbol{\Psi} = \begin{bmatrix} \boldsymbol{H}^{q_3} \otimes \boldsymbol{H}^{q_2} \otimes \boldsymbol{H}^{q_1} \\ \boldsymbol{H}^{q_3} \otimes \boldsymbol{H}^{q_2} \otimes \boldsymbol{G}^{q_1} \\ \boldsymbol{H}^{q_3} \otimes \boldsymbol{G}^{q_2} \otimes \boldsymbol{H}^{q_1} \\ \boldsymbol{H}^{q_3} \otimes \boldsymbol{G}^{q_2} \otimes \boldsymbol{G}^{q_1} \\ \boldsymbol{G}^{q_3} \otimes \boldsymbol{H}^{q_2} \otimes \boldsymbol{H}^{q_1} \\ \boldsymbol{G}^{q_3} \otimes \boldsymbol{H}^{q_2} \otimes \boldsymbol{G}^{q_1} \\ \boldsymbol{G}^{q_3} \otimes \boldsymbol{G}^{q_2} \otimes \boldsymbol{H}^{q_1} \\ \boldsymbol{G}^{q_3} \otimes \boldsymbol{G}^{q_2} \otimes \boldsymbol{G}^{q_1} \end{bmatrix}^T .$$

where the matrices $\boldsymbol{H}$ and $\boldsymbol{G}$ are defined in equations (4.8) and (4.9).

The extension to higher scales is similar to equation (4.3.2). We can build the matrix up until the qth scale, where $q = \min\{q_1, q_2, q_3\}$. The full basis matrix is

$$\boldsymbol{\Psi} = \begin{bmatrix} (\boldsymbol{H}^{q_3-q}\boldsymbol{H}^{q_3-q+1}\cdots\boldsymbol{H}^{q_3}) \otimes (\boldsymbol{H}^{q_2-q}\boldsymbol{H}^{q_2-q+1}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{H}^{q_1-q}\boldsymbol{H}^{q_1-q+1}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{H}^{q_3-q}\boldsymbol{H}^{q_3-q+1}\cdots\boldsymbol{H}^{q_3}) \otimes (\boldsymbol{H}^{q_2-q}\boldsymbol{H}^{q_2-q+1}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{G}^{q_1-q}\boldsymbol{H}^{q_1-q+1}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{H}^{q_3-q}\boldsymbol{H}^{q_3-q+1}\cdots\boldsymbol{H}^{q_3}) \otimes (\boldsymbol{G}^{q_2-q}\boldsymbol{H}^{q_2-q+1}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{H}^{q_1-q}\boldsymbol{H}^{q_1-q+1}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{H}^{q_3-q}\boldsymbol{H}^{q_3-q+1}\cdots\boldsymbol{H}^{q_3}) \otimes (\boldsymbol{G}^{q_2-q}\boldsymbol{H}^{q_2-q+1}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{G}^{q_1-q}\boldsymbol{H}^{q_1-q+1}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{G}^{q_3-q}\boldsymbol{H}^{q_3-q+1}\cdots\boldsymbol{H}^{q_3}) \otimes (\boldsymbol{H}^{q_2-q}\boldsymbol{H}^{q_2-q+1}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{H}^{q_1-q}\boldsymbol{H}^{q_1-q+1}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{G}^{q_3-q}\boldsymbol{H}^{q_3-q+1}\cdots\boldsymbol{H}^{q_3}) \otimes (\boldsymbol{H}^{q_2-q}\boldsymbol{H}^{q_2-q+1}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{G}^{q_1-q}\boldsymbol{H}^{q_1-q+1}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{G}^{q_3-q}\boldsymbol{H}^{q_3-q+1}\cdots\boldsymbol{H}^{q_3}) \otimes (\boldsymbol{G}^{q_2-q}\boldsymbol{H}^{q_2-q+1}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{H}^{q_1-q}\boldsymbol{H}^{q_1-q+1}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{G}^{q_3-q}\boldsymbol{H}^{q_3-q+1}\cdots\boldsymbol{H}^{q_3}) \otimes (\boldsymbol{G}^{q_2-q}\boldsymbol{H}^{q_2-q+1}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{G}^{q_1-q}\boldsymbol{H}^{q_1-q+1}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{H}^{q_3-q+1}\boldsymbol{H}^{q_3-q+2}\cdots\boldsymbol{H}^{q_3}) \otimes (\boldsymbol{H}^{q_2-q+1}\boldsymbol{H}^{q_2-q+2}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{G}^{q_1-q+1}\boldsymbol{H}^{q_1-q+2}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{H}^{q_3-q+1}\boldsymbol{H}^{q_3-q+2}\cdots\boldsymbol{H}^{q_3}) \otimes (\boldsymbol{G}^{q_2-q+1}\boldsymbol{H}^{q_2-q+2}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{H}^{q_1-q+1}\boldsymbol{H}^{q_1-q+2}\cdots\boldsymbol{H}^{q_1}) \\ \vdots \\ (\boldsymbol{G}^{q_3-1}\boldsymbol{H}^{q_3}) \otimes (\boldsymbol{G}^{q_2-1}\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{G}^{q_1-1}\boldsymbol{H}^{q_1}) \\ \boldsymbol{H}^{q_3} \otimes \boldsymbol{H}^{q_2} \otimes \boldsymbol{G}^{q_1} \\ \boldsymbol{H}^{q_3} \otimes \boldsymbol{G}^{q_2} \otimes \boldsymbol{H}^{q_1} \\ \boldsymbol{H}^{q_3} \otimes \boldsymbol{G}^{q_2} \otimes \boldsymbol{G}^{q_1} \\ \boldsymbol{G}^{q_3} \otimes \boldsymbol{H}^{q_2} \otimes \boldsymbol{H}^{q_1} \\ \boldsymbol{G}^{q_3} \otimes \boldsymbol{H}^{q_2} \otimes \boldsymbol{G}^{q_1} \\ \boldsymbol{G}^{q_3} \otimes \boldsymbol{G}^{q_2} \otimes \boldsymbol{H}^{q_1} \\ \boldsymbol{G}^{q_3} \otimes \boldsymbol{G}^{q_2} \otimes \boldsymbol{G}^{q_1} \end{bmatrix}^T$$

The size of the basis matrices is $(rcf) \times (rcf)$. Even for relatively small signals, these matrices can be prohibitively large. We address this problem in Chapter 7.

6.3 Video Interpolation

Once we have successfully created a vectorization scheme for the video signal $\boldsymbol{v}$ and constructed the Haar basis matrices $\boldsymbol{\Psi}$ for the desired scales, we can apply to MSCE to reconstruct masked video signals.

We pass the measured pixel values $\boldsymbol{y}$ and the design matrix $\boldsymbol{\Phi} = \boldsymbol{\Theta}\boldsymbol{\Psi}$ (where $\boldsymbol{\Theta}$ is the video mask) t o Algorithm 3. The output will be an interpolated video that hopefully recovers very fine details across the entire signal.

Figure 6.4, we show two example frames of the test video "foreman". Panels (a) and (b) show the undersampled signal $\boldsymbol{y}$. Panels (c) and (d) show the output of the

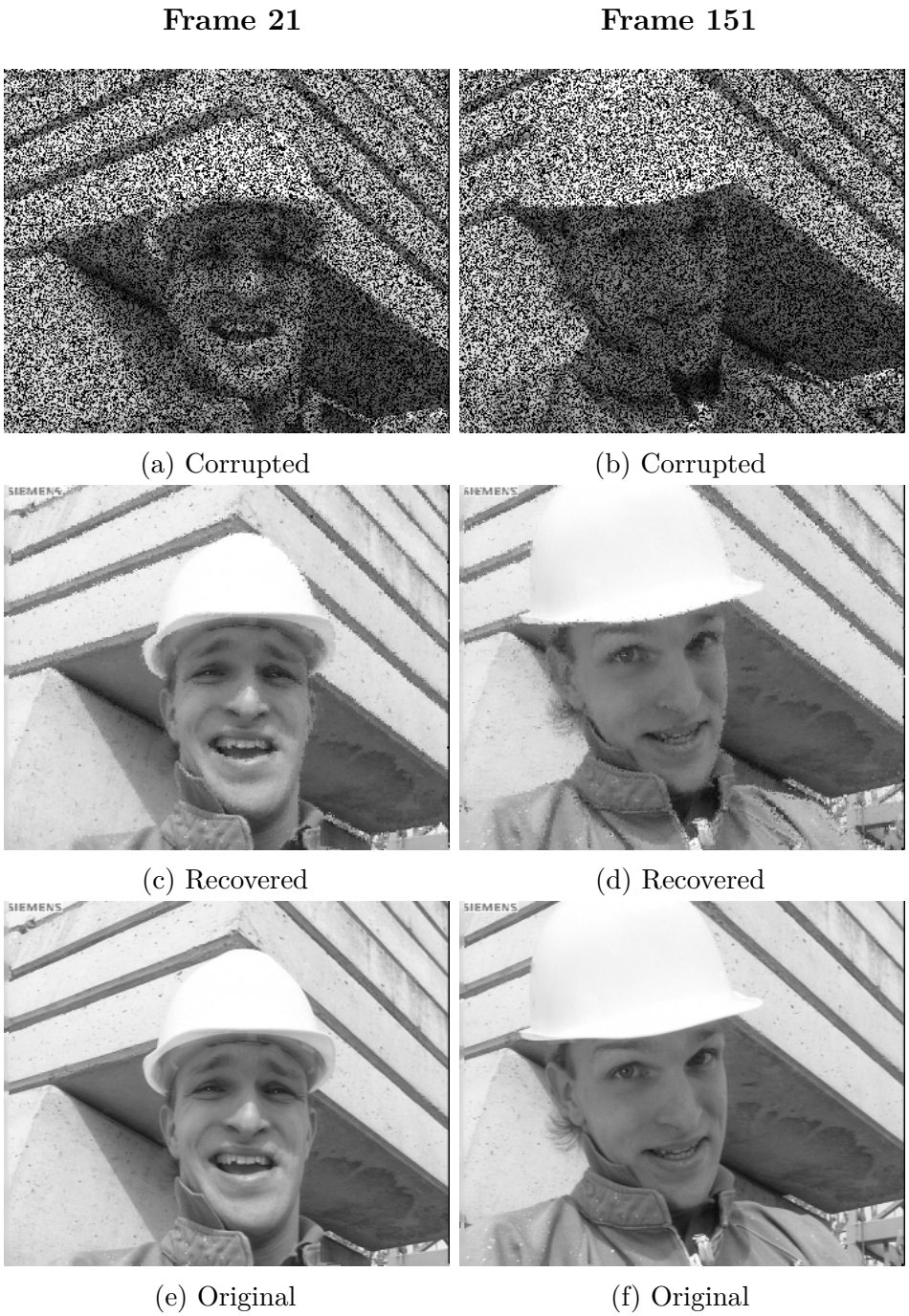


Fig. 6.4 Example of a masked video signal, where only 60% of the pixel values are known (a-b). Reconstruction via the MSCE Algorithm using 2 cascades (PSNR: 28.6) (c-d). Original video (e-f).

MSCE algorithm. For reference, we have also included the frames from the original video in panels (e) and (f).

Further test results of this method are shown in Chapter 8.

Chapter 7

Implementation Details

We have implemented the MSCE Algorithm for video signals in C++.

In this chapter, we will discuss some of the design decisions and optimizations that went into the implementation.

7.1 Blockwise Reconstruction

Let $\boldsymbol{v}$ be a video signal consisting of f frames with a width of w and a height h. Vectorizing $\boldsymbol{v}$ gives us a vector of length hwf. In order to reconstruct such signals in the Bayesian Compressive Sensing framework (Chapter 5), we need to form the basis matrix $\boldsymbol{\Psi}$ which has dimensions $(hwf) \times (hwf)$.

Even for relatively small videos, the memory requirements for such large basis matrices can easily be in the order of terabytes. As an example, consider the commonly used *CIF (Common Intermediate Format)*. CIF videos have a spatial resolution of 352×288. For a CIF video containing 100 frames, the required memory for storing $\boldsymbol{\Psi}$ as *floats* is

$$(288 \times 352 \times 100)^2 \times 4 \text{ bytes} = 4.11 \times 10^{14} \text{ bytes} = 411 \text{ TB}$$

In our code, we address the problem by performing a *blockwise reconstruction*. We split the input signal into small sub-blocks of size $2^{j_1} \times 2^{j_2} \times 2^{j_3}$. A typical size of such a block is $16 \times 16 \times 16$ (a so-called "macroblock").

The blocks are sequentially passed to the MSCE algorithm and after each block has been reconstructed , we put them back together to obtained the recovered video.

Note that the size of the block restricts the number of cascades in the MSCE algorithm. To run the algorithm up to scale s, we require a block size of at least $2^s \times 2^s \times 2^s$.

7.2 Code Optimization

7.2.1 Parallelization

In the blockwise reconstruction, each block is processed independently from the others. Therefore, we have added an option to the code to process the blocks in parallel. Using the *Message Passing Interface* (MPI), we run the program on several processors, splitting the workload evenly between them. This generally leads to a very significant speedup.

However, if the blocksize is too small, the communication between processes - when gathering the results - will dominate over the actual computation time. Initial tests seem to indicate that, in order to achieve a significant increase in the execution time, the blocks should be at least of size $4 \times 4 \times 4$ before switching to parallel mode.

7.2.2 Fixed Noise Variance

At each stage of the MSCE algorithm, we keep the noise variance σ^2 fixed, while training the Sparse Bayesian Learning model. As noted in Section 3.2.2, doing so allows us to use the efficient update formulae in [11] which speed up training.

7.2.3 Modified RVM Training

In the MSCE, the RVM is trained using Algorithm 2. We use a slightly modified version of the training algorithm in which we only consider addition of basis functions to the model.

At each iteration, we add the basis vector $\boldsymbol{\phi}$ that results in the largest increase of the marginal likelihood. We continue to do so until none of the remaining candidate basis functions cause an increase in log marginal likelihood that is above the convergence threshold.

For the problem of image and video reconstruction, this modified algorithm gives qualitatively similar results to the unmodified version [5]. However, it can lead to a significant reduction in the runtime of the algorithm.

[Fill in if size is not 2^N]

Chapter 8

Test Simulations

8.1 Performance Metrics

In this section, we will introduce three performance metrics that are often used to measure the quality of reconstructed images and videos.

Let $\hat{v} \in \mathbb{R}^M$ be a reconstructed signal (in vectorized form) and let $\boldsymbol{v}$ be the corresponding orginal signal. The *mean square error* (MSE) of the reconstruction is defined as

$$\mathrm{MSE}(\hat{\boldsymbol{v}}) = \frac{\sum_{i=1}^{M}(\hat{v}_i - v_i)^2}{M}$$

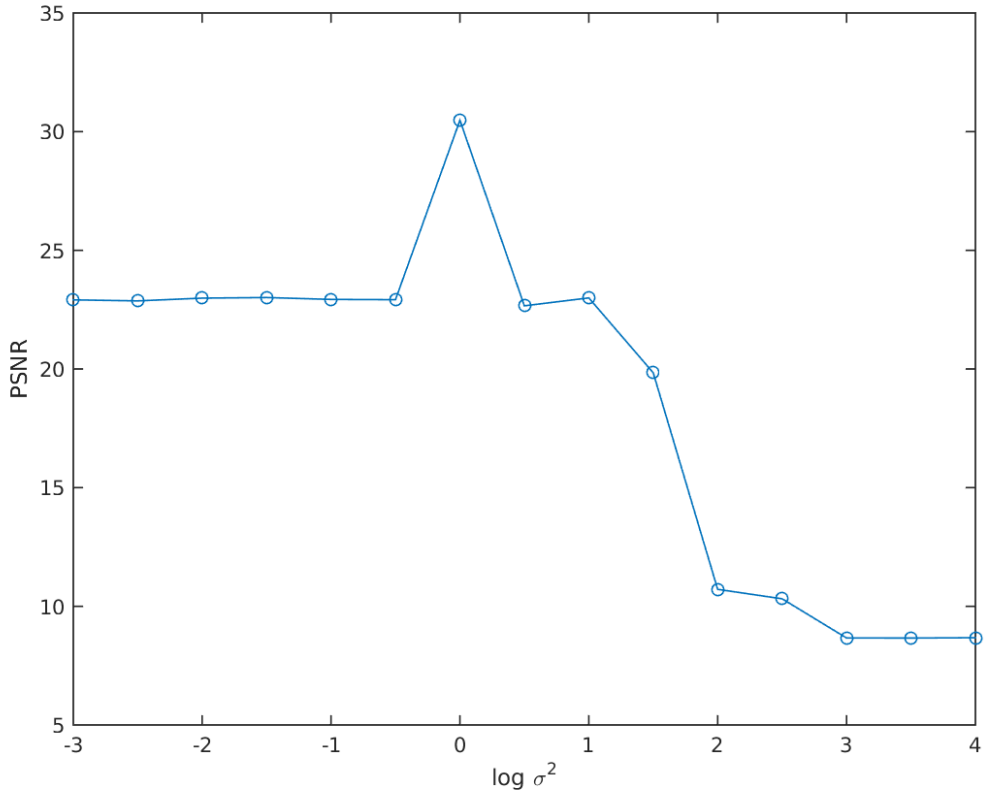
The MSE is zero if and only if we $\hat{\boldsymbol{v}}$ is an exact reconstruction of $\boldsymbol{v}$.

Using the MSE, we can compute the *Peak Signal-to-Noise Ratio* (PSNR) of the reconstruction:

$$\mathrm{PSNR}(\hat{\boldsymbol{v}}) = 10 \cdot \log_{10}\left(\frac{R^2}{\mathrm{MSE}(\hat{\boldsymbol{v}})}\right)$$

where R is the maximum fluctuation in the input signal data type. For grayscale images or videos in which the pixel values are stored as 8-bit unsigned integers, we have that $R = 256$.

The PSNR is usually expressed in term of decibel (dB). Higher values of the PSNR correspond to more accurate reconstructions. The PSNR is widely used in the image and video compression literature to measure the quality of a compressed signal. Generally, when comparing the reconstruction quality, the PSNR should only be used if it was measured on the same signal.



The final performance metric that we compute is the *relative reconstruction error*. It is given by

$$\text{RRE}(\hat{\boldsymbol{v}}) = \frac{||\hat{\boldsymbol{v}} - \boldsymbol{v}||_2}{||\boldsymbol{v}||_2}$$

This is the metric that was used in [3] and [5].

8.2 Model Selection

8.2.1 Noise Variance σ^2

8.2.2 Noise Threshold τ

8.3 Example Results

Different Decimation Patters.

Different Percentages.

Different block size.

8.4 Comparison of MSCE against BCS with DCT for interpolation

Performance curve

What if pre-compressed

Comparison with literature?

8.5 Analysis of Speed

Time vs block dim.

Serial vs MPI.

8.6 Video Reconstruction for General Sensing Matrices

- General sensing Matrix for BCS of videos?
- Masks vs Gaussian vs Rademacher
- MSCE was done to solve the signal interpolation. In particular, to fix the problem with small support of wavelets while still using their power.
- Problem: Limited to signal interpolation. Solution: Use CS sensing matrices (but Cascade don't work yet).

Chapter 9

Conclusion

In this thesis, we investigated the efficacy of the Bayesian Compressive Sensing framework for efficient reconstruction of highly under-sampled video signals. We developed an extension to the Multi-Scale Cascade of Estimations algorithm that achieves near-perfect reconstructions of videos from a very small set of measurements.

To do so, we constructed three-dimensional wavelet basis functions that allow for a highly compressible representation of the video signal. Compressive Sensing inversion is then formulated as a machine learning problem and the Relevance Vector Machine was employed to find highly sparse solutions. To boost performance, a cascade of RVMs it built that exploits the multi-resolution properties of wavelet basis functions.

In order to deal with the large memory requirements of the algorithm, the reconstruction is performed in blockwise fashion. We have also implemented the method as a distributed program, resulting in dramatically reduced execution times.

Future research could improve performance by extending these methods in various ways. Alternative sets of waveles, such as the CDF-9/7 wavelet that is used by the JPEG2000 format, may lead to sparser representations of the video signals. Furthermore, the speed of the algorithm can be increased by using a multi-threaded implementation of the Sequential Sparse Bayesian Learning Algorithm.

Development of Bayesian approaches to Compressive Sensing systems is an active area of research.

References

[1] Baraniuk, R. G. (2007). Compressive sensing. *IEEE signal processing magazine*, 24(4).

[2] Candès, E. J. and Wakin, M. B. (2008). An introduction to compressive sampling. *Signal Processing Magazine, IEEE*, 25(2):21–30.

[3] Ji, S., Xue, Y., and Carin, L. (2008). Bayesian compressive sensing. *IEEE Transactions on Signal Processing*, 56(6):2346–2356.

[4] Khayam, S. A. (2003). The discrete cosine transform (dct): theory and application. *Michigan State University*, 114.

[5] Pilikos, G. (2014). Signal reconstruction using compressive sensing. MPhil thesis, University of Cambridge.

[6] Shannon, C. E. (1949). Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21.

[7] Stollnitz, E. J., DeRose, T. D., and Salesin, D. H. (1995). Wavelets for computer graphics: a primer. 1. *Computer Graphics and Applications, IEEE*, 15(3):76–84.

[8] Taubman, D. and Marcellin, M. (2012). *JPEG2000 Image Compression Fundamentals, Standards and Practice: Image Compression Fundamentals, Standards and Practice*. The Springer International Series in Engineering and Computer Science. Springer US.

[9] Tipping, A. and Faul, A. (2002). Analysis of sparse bayesian learning. *Advances in neural information processing systems*, 14:383–389.

[10] Tipping, M. E. (2001). Sparse bayesian learning and the relevance vector machine. *The journal of machine learning research*, 1:211–244.

[11] Tipping, M. E., Faul, A. C., et al. (2003). Fast marginal likelihood maximisation for sparse bayesian models. In *AISTATS*.

[12] Zeng, J., Au, O. C., Dai, W., Kong, Y., Jia, L., and Zhu, W. (2013). A tutorial on image/video coding standards. In *Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2013 Asia-Pacific*, pages 1–7. IEEE.