MULTI-LAYER DICTIONARY LEARNING USING LOW-RANK UPDATES

A Dissertation
Presented to
The Academic Faculty

By

Lee Richert

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

May 2022

MULTI-LAYER DICTIONARY LEARNING USING LOW-RANK UPDATES

Approved by:

Dr. David V. Anderson
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Mark A. Davenport
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Justin Romberg
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: December 17, 2021

ACKNOWLEDGEMENTS

I would like to acknowledge and thank my advisor Dr. David Anderson for his guidance and mentorship. I would also like to thank my brother Trip for the many hours of invaluable conversations about software design and development.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Acronyms

ADMM	the Alternating Directions Method of Multipliers algorithm
ISTA	the Iterative Shrinkage Thresholding Algorithm
FISTA	the Fast Iterative Shrinkage Tresholding Algorithm
RGB	Describes an image with red, green, and blue channels
JPEG	an image compression standard developed by the Joint Photographic Experts Group
FFT	the discrete Fourier transform
DCT	type-II discrete cosine transform
CPU	central processing unit
RAM	random-access memory

Integer Constants

Capital letters that are not bold are used to denote integer constants.

M	the number of filters
C	the number of channels
$\hat{K}$	the number of elements in a single channel of the signal
K	the number of elements in a single channel of a filter
Я	a small integer that specifies the rank of the dictionary updates
L	the number of layers

Matrices

Bold capital letters are used to denote matrices.

$\boldsymbol{D}$	the dictionary. For most of the dissertation, $\boldsymbol{D}$ has circulant matrix blocks.
$\boldsymbol{G}$	another variable used to represent the dictionary. $\boldsymbol{G}$ is used when it is necessary to distinguish 2 different dictionaries, like when dictionary $\boldsymbol{D}$ is used to approximate dictionary $\boldsymbol{G}$.
$\boldsymbol{S}$	a collection of signal vectors, either gathering multiple channels or multiple samples.
$\boldsymbol{X}$	a collection of dictionary coefficient vectors, corresponding to multple signal vectors.
$\boldsymbol{A}$, $\boldsymbol{B}$	the linear operators appearing the the affine constraints for ADMM. $\boldsymbol{A}$ and $\boldsymbol{B}$ are also used as arbitary matrices when discussing general linear algebra principles.
$\boldsymbol{U}$, $\boldsymbol{V}$	arbitrary matrices used for discussing general linear algebra principles and low-rank updates.
$\boldsymbol{\Phi}$	an arbitrary linear operator
$\boldsymbol{T}$	a diagonal matrix that has diagonal elements of 1 for dictionary elements that are not constrained to be zero, and zeros for the other diagonal elements
$\boldsymbol{W}$	a matrix which converts an image vector from RGB to YUV, downsamples the UV channels, and computes the DCT of 8×8 blocks.
$\boldsymbol{\beta}$	part of the dictionary for the product dictionary model
$\boldsymbol{Q}$	used to denote the matrix $\rho \mathbf{I} + \hat{\boldsymbol{D}}^H \hat{\boldsymbol{D}}$
$\boldsymbol{\Xi}$	used to denote the matrix $\rho \mathbf{I} + \hat{\boldsymbol{D}} \hat{\boldsymbol{D}}^H$
$\boldsymbol{R}$	a rescaling matrix, used to scale a normalized dictionary back to its unnormalized form

Vectors

Bold, lower-case letters are used to denote vectors.

$\boldsymbol{x}$, $\boldsymbol{y}$ the primal variables for ADMM.

$\boldsymbol{u}$ the dual variable for ADMM.

$\boldsymbol{c}$ the constraint vector in ADMM.

$\boldsymbol{x}$, $\boldsymbol{z}$ the coefficients for dictionary model in ADMM algorithm.

$\boldsymbol{z}$ also used as arbitrary complex vector.

$\boldsymbol{v}$ another primal variable (grouped with $\boldsymbol{z}$) used in chapter 4.

γ the corresponding dual variable.

$\boldsymbol{x}$, $\boldsymbol{z}$ also used as vectors in the FISTA algorithm.

$\boldsymbol{x}$ also used as an arbitrary vector throughout document. Context will make clear.

$\boldsymbol{s}$ the signal.

$\boldsymbol{b}$ another arbitrary vector.

$\boldsymbol{u}$, $\boldsymbol{v}$ more arbitrary vectors, usually used in pairs. The may collectively specify a rank-1 update to a matrix: $\boldsymbol{u}\boldsymbol{v}^H$.

$\boldsymbol{f}$ a dictionary filter.

$\boldsymbol{d}$ a column of $\boldsymbol{D}$.

$\boldsymbol{q}$ the quality-factor dependent vector used in quantization in JPEG compression.

$\boldsymbol{\omega}$ an eigenvector.

$\boldsymbol{e}$ the standard Euclidean unit vector.

Non-Integer Scalars

Lower-case letters that are not bold are generally used to denote scalars that are not integer constants. $\mathbb{L}$ and $\mathscr{L}$ are the exceptions.

a, b arbitrary scalars.

ρ a scalar for the ADMM alrogithm that specifies both the weighting of the constraints in the augmented Lagrangian and the stepsize in the dual variable update.

α the over-relaxation or under-relaxation factor for ADMM

λ the factor for the L_1 penalty. λ is also used as the factor for the L_2 penalty on the image gradients for Tikhonov regularization. Context will make clear.

μ the factor for the L_2 penalty for the reconstruction of the signal or coefficients from the previous layer.

$\mathbb{L}$ an estimate of the Lipshitz constant, used to determine stepsize in the ISTA and FISTA algorithms.

τ the eigenvalue

$\eta_{\boldsymbol{u},\boldsymbol{v}}$ used to represent $\boldsymbol{u}^H \hat{\boldsymbol{D}} \boldsymbol{v}$

$\eta_{\boldsymbol{u}}$ used to represent $\|\boldsymbol{D}^H \boldsymbol{u}\|_2^2$

$\eta_{\boldsymbol{v}}$ used to represent $\|\boldsymbol{v}\|_2^2$

r, ω used to specify momentum stepsize in FISTA and a FISTA-like algorithm. Both always appear with superscripts specifying iteration.

$\mathscr{L}$ loss (as in the loss function)

Functions and Operations

$*$	Circular convolution
a^*	the complex conjugate of a.
$\boldsymbol{u}^H, \boldsymbol{U}^H$	the Hermitian transpose of a vector $\boldsymbol{u}$ or matrix $\boldsymbol{U}$
$\mathcal{L}_\rho$	the augmented Lagrangian function
S	the shrinkage operator
$\mathcal{F}$	applies the Fourier tranform to each channel and/or filter
f and g	arbitrary convex functions. f may also be used to specify the objective function of a minimization problem. f and g are also used for arbitrary functions that are part of a composite function. Context should make clear.
$\mathrm{q}(\cdot)$	quantizes a vector
$\mathbb{1}_{\text{condition}}$	takes on a value of 0 when the condition is true and ∞ when the condition is false.
$\arg\min$	the argument minimum of an expression
$\nabla_a b$	the gradient of b in respect to a
L_1,$\|\cdot\|_1$	the L_1 norm
L_2,$\|\cdot\|_2$	the L_2 norm
$\boldsymbol{\Phi}$	n arbitrary linear operator (this operator is also listed under matrices, since matrices are linear operators)
$\boldsymbol{T}$	zeros out all dictionary elements that are constrained to be zero. (This operator is also listed under matrices since matrices are linear operators.)
$\boldsymbol{W}$	converts from RGB to YUV, downsamples the UV channels, and computes the DCT of 8×8 blocks. (This operator also appears under matrices.)
$\mathfrak{R}$	selects the real component of a complex number, vector, or matrix
$\mathfrak{I}$	selects the imaginary component of a complex number, vector, or matrix. (Output is real.)

Superscripts

a^b	An exponent: the b^{th} power of a
a^*	the complex conjugate of a
$\boldsymbol{u}^H$, $\boldsymbol{U}^H$	the Hermitian transpose of vector $\boldsymbol{u}$ or matrix $\boldsymbol{U}$
i^{th}	occuring at position i in a sequence
$\boldsymbol{\gamma}^{(t)}$	the value of vector $\boldsymbol{\gamma}$ at the t^{th} iteration
$\boldsymbol{D}^{(n)}$	the value of dictionary $\boldsymbol{D}$ at the n^{th} dictionary update
$\nabla_a^{(\boldsymbol{x}\to\boldsymbol{y})}\mathscr{L}$	the gradient term of $\mathscr{L}$ in respect to a that corresponds to the computation of $\boldsymbol{y}$ from $\boldsymbol{x}$. The gradient of $\mathscr{L}$ in respect to a is the sum of all the gradient terms.

Subscripts

m	specifies the filter. If there are multiple layers, $[m]$ will be used instead of subscript m.
n	specifies the sample.
c	specifies the channel. If there are multiple layers $[c]$ will be used instead of subscript c.
ℓ	specifies the layer.
$+$ and $-$	used to specify the eigenvalues or eigenvectors of a 2×2 matrix corresponding to the plus or minus in the quadratic formula.
ρ in $\mathcal{L}_\rho$	specifies the scalar weight of the L_2 norm related to the affine constraints, used in the augmented Lagrangian function.
init in $\boldsymbol{D}_{\text{init}}$	short for initial value, appears in algorithms.
sc in $\boldsymbol{\gamma}_{\text{sc}}$	short for "scaled", and indicates that the variable in the algorithm is a scaled form of the variable.
i	used for essentially all other subscript indexing.

Indexing Integers

Lower case letters that are not bold are also used to denote indexing integers.

n selects the dictionary update or corresponding signal

t selects the iteration

m selects the filter

c selects the channel

$\hat{k}$ specifies the frequency

i used for most other indexing.

SUMMARY

This dissertation presents a novel convolutional dictionary learning algorithm for signals with a large number of channels. This algorithm uses low-rank updates for the dictionary, so that a matrix decomposition necessary for pursuit can be updated efficiently. In later chapters, this algorithm is applied to multi-layer dictionary models with multi-channel dictionaries and single-channel coefficients, where the number of filters in one layer is the number of channels in the subsequent dictionary layer. This architecture is demonstrated on the task of JPEG compression artifact removal.

CHAPTER 1
INTRODUCTION

1.1 Dictionaries and Dictionary Learning

Dictionary models are powerful and versatile tools, useful in denoising [1], classification [2], anomaly detection [3], super-resolution [4][5], inpainting [6], trajectory-basis non-rigid structure from motion [7][8], and more. Dictionaries are interpretable, and their models are well-suited for incorporating model assumptions using prior knowledge of the data. In some applications, well-performing dictionaries can be learned from very limited data.

The dictionary model decomposes the signal $\boldsymbol{s}$ into the dictionary $\boldsymbol{D}$ and coefficients $\boldsymbol{x}$.

$$\boldsymbol{s} \approx \boldsymbol{D}\boldsymbol{x} \tag{1.1}$$

When the dictionary is known, solving for $\boldsymbol{x}$ using $\boldsymbol{D}$ and $\boldsymbol{s}$ is a pursuit algorithm. When the dictionary is unknown, learning the dictionary from $\boldsymbol{s}$ is called dictionary learning. In this dissertation, I present a novel dictionary learning algorithm specifically for convolutional dictionaries.

1.1.1 Convolutional Dictionaries

The convolutional dictionary model is a dictionary model which imposes a convolutional structure on the dictionary. This structure is shown in Figure 1.1. Note that while the example uses 1-dimensional convolution, it is straightforward to extend to multi-dimensional convolution through vectorization. The convolutional dictionary model equation can be notated as signals convolved with filters:

$$\boldsymbol{s} \approx \sum_m \boldsymbol{f}_m * \boldsymbol{x}_m \tag{1.2}$$

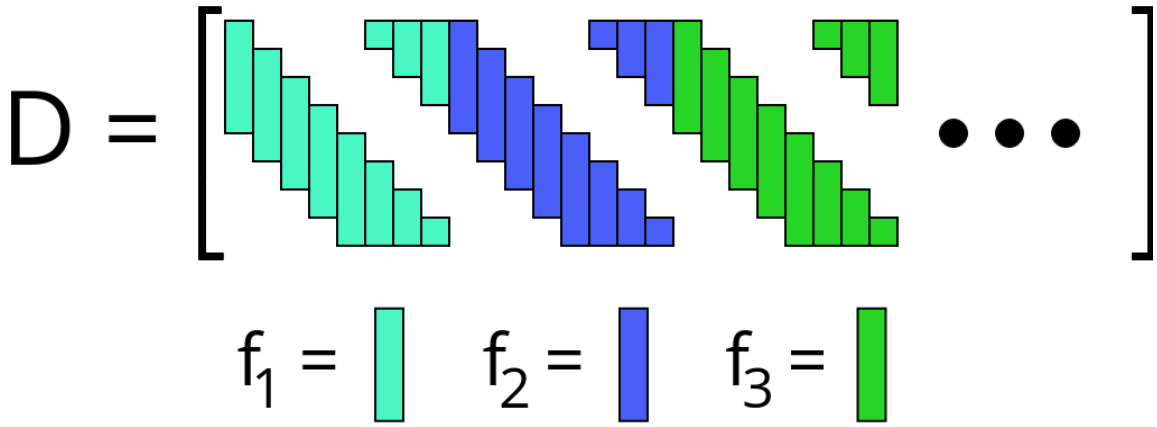


Figure 1.1: This is a 1-dimensional dictionary with convolutional structure. The columns of the dictionary are shifted versions of the dictionary filters.

where $\boldsymbol{f}_m$ is the m^{th} filter of the dictionary, $\boldsymbol{x}_m$ is the coefficients for the m^{th} filter, and $*$ indicates convolution. This structure is useful for signals where shift-invariance is a desirable characteristic and/or when dealing with variable-length data. Convolutional approaches often outperform patch-based methods. For the rest of this dissertation, I will notate $\sum_m \boldsymbol{f}_m * \boldsymbol{x}_m$ as $\boldsymbol{D}\boldsymbol{x}$. The matrix $\boldsymbol{D}$ consists of M circulant blocks $\boldsymbol{D}_m$, each block corresponding to a particular filter $\boldsymbol{f}_m$ of the M dictionary filters.

1.2 Multi-Layer Dictionaries

A multi-layer dictionary treats the coefficients as the signal of a subsequent dictionary model. That is,

$$\begin{aligned} \boldsymbol{s} &\approx \boldsymbol{D}_1\boldsymbol{x}_1 \\ \boldsymbol{x}_1 &\approx \boldsymbol{D}_2\boldsymbol{x}_2 \\ &\vdots \\ \boldsymbol{x}_{L-1} &\approx \boldsymbol{D}_L\boldsymbol{x}_L \end{aligned} \tag{1.3}$$

where L is the number of layers. Pursuit for a multi-layer dictionary model seeks to find the coefficients for all layers $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_L$. Multi-layer dictionary models have been used

in applications such as feature extraction for classification [9][10][11] and trajectory-basis non-rigid structure from motion [8]. Dictionary models can often be easily extended to allow the researcher or practitioner to inject prior knowledge into the model architecture through the use of convex constraints or known linear operator $\boldsymbol{\Phi}$. For convolutional dictionaries, the projection operator $\boldsymbol{\Phi}$ need not have convolutional structure.

$$\begin{aligned} \boldsymbol{s} &\approx \boldsymbol{\Phi} \boldsymbol{D} \boldsymbol{x} \\ \mathrm{f}_i(\boldsymbol{x}) &\leq b_i \; \forall i \end{aligned} \tag{1.4}$$

In the context of multi-layer dictionary models, these types of dictionary model modifications have been shown to be useful for applications such as interpolation of LIDAR measurements using RGB imagery [12][13] and compression-artifact removal [8].

Some researchers have noted that convolutional neural networks resemble multi-layer dictionaries [6]: thresholding is a type of pursuit algorithm, and rectified linear unit activations perform the same operation. A convolutional neural network is a composite function of convolutionals, pooling functions, and activation functions. Over the last decade, convolutional neural networks have reached state-of-the-art performance on a wide variety of tasks. In a way, multi-layer dictionary models can be thought of as a blend of dictionary models and convolutional neural networks, since multi-layer dictionary models mimic the layered structure of convolutional neural networks, while maintaining the interpretability of dictionary models.

1.3 Organization of Dissertation

In chapter 2, I derive a novel dictionary learning method for multi-channel signals. In chapter 3, I show how that approach can be adapted to multi-layer dictionary models. Finally, in chapter 4, I apply the multi-layer dictionary approach to JPEG compression artifact removal. Chapter 5 features some practical considerations when implementing these algorithms.

CHAPTER 2

LEARNING DICTIONARIES FOR MULTI-CHANNEL SIGNALS

2.1 Introduction

Much of the literature on using convolutional dictionaries is tailored to applications with signals that only have a small number of channels (like RGB or grayscale imagery). Many of the methods designed with a small number of channels in mind do not transfer well to applications like hyperspectral imagery [14] with signal measurements containing a large number of channels.

Signal measurements are not the only way to end up with a large number of channels. In a multi-layer dictionary model, the coefficients corresponding to a dictionary from one layer become the signal for the subsequent layer. The number of channels for this signal is the number of dictionary filters from the previous layer, so if the number of filters in a layer is large, the number of channels for the subsequent layer will also be large.

This chapter presents a novel method to learn convolutional dictionaries for multi-channel signals.

2.2 Dictionary Types

There are many ways to construct a convolutional sparse representation of a multi-channel signal. One way models can differ is if and how signal channels share dictionaries and coefficients.

For example, models with single-channel dictionaries represent the channels in the coefficients:[1]

$$\boldsymbol{S} \approx \boldsymbol{D}\boldsymbol{X} \tag{2.1}$$

[1] In this case, both $\boldsymbol{X}$ and $\boldsymbol{S}$ still refer to a single sample. They are capitalized because they are matrices.

$$\boldsymbol{S} = \begin{bmatrix} \boldsymbol{s}_1 & \dots & \boldsymbol{s}_C \end{bmatrix} \tag{2.2}$$

$$\boldsymbol{D} = \begin{bmatrix} \boldsymbol{D}_1 & \dots & \boldsymbol{D}_M \end{bmatrix} \tag{2.3}$$

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_{1,1} & \dots & \boldsymbol{x}_{1,C} \\ \vdots & \ddots & \vdots \\ \boldsymbol{x}_{M,1} & \dots & \boldsymbol{x}_{M,C} \end{bmatrix} \tag{2.4}$$

where $\boldsymbol{s}_c \in \mathbb{R}^{\hat{K}}$, $\boldsymbol{D}_m \in \mathbb{R}^{\hat{K} \times \hat{K}}$, and $\boldsymbol{x}_{m,c} \in \mathbb{R}^{\hat{K}}$, using $\hat{K}$ as the number of elements in a single channel of the signal.

Alternatively, each signal channel could be given its own dictionary model.

$$\boldsymbol{s}_c \approx \boldsymbol{D}_c \boldsymbol{x}_c \tag{2.5}$$

where $\boldsymbol{s}_c \in \mathbb{R}^{\hat{K}}$, $\boldsymbol{D}_c \in \mathbb{R}^{\hat{K} \times M\hat{K}}$, and $\boldsymbol{x}_c \in \mathbb{R}^{M\hat{K}}$.

In both the above-mentioned cases, the models use multi-channel coefficients. For multi-layer models, this poses a problem: with each subsequent layer, the number of coefficients grows by a factor of the number of filters. Thus, the number of coefficients increases exponentially in respect to the number of layers.

This exponential growth could be dampened (though not eliminated) through the use of product dictionaries [15]. In product dictionary models, correlations between channels are captured in a separate matrix ß.

$$\boldsymbol{S} \approx \boldsymbol{D}\boldsymbol{X}ß \tag{2.6}$$

$$\boldsymbol{S} = \begin{bmatrix} \boldsymbol{s}_1 & \dots & \boldsymbol{s}_C \end{bmatrix} \tag{2.7}$$

$$\boldsymbol{D} = \begin{bmatrix} \boldsymbol{D}_1 & \dots & \boldsymbol{D}_M \end{bmatrix} \tag{2.8}$$

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_{1,1} & \dots & \boldsymbol{x}_{1,B} \\ \vdots & \ddots & \vdots \\ \boldsymbol{x}_{M,1} & \dots & \boldsymbol{x}_{M,B} \end{bmatrix} \tag{2.9}$$

where $\boldsymbol{s}_c \in \mathbb{R}^{\hat{K}}$, $\boldsymbol{D}_m \in \mathbb{R}^{\hat{K} \times \hat{K}}$, $\boldsymbol{x}_{m,b} \in \mathbb{R}^{\hat{K}}$, and $\mathfrak{ß} \in \mathbb{R}^{B \times C}$. Pursuit algorithms for product dictionaries still seek to estimate $\boldsymbol{X}$ for fixed $\boldsymbol{S}$, $\boldsymbol{D}$, and $\mathfrak{ß}$. For dictionary learning, $\mathfrak{ß}$ is updated with $\boldsymbol{D}$.

For this work, I will instead focus on multi-channel dictionaries with shared coefficients.

$$\boldsymbol{s} \approx \boldsymbol{D}\boldsymbol{x} \tag{2.10}$$

$$\boldsymbol{s} = \begin{bmatrix} \boldsymbol{s}_1 \\ \dots \\ \boldsymbol{s}_C \end{bmatrix} \tag{2.11}$$

$$\boldsymbol{D} = \begin{bmatrix} \boldsymbol{D}_{1,1} & \dots & \boldsymbol{D}_{1,M} \\ \vdots & \ddots & \vdots \\ \boldsymbol{D}_{C,1} & \dots & \boldsymbol{D}_{C,M} \end{bmatrix} \tag{2.12}$$

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_M \end{bmatrix} \tag{2.13}$$

where $\boldsymbol{s}_c \in \mathbb{R}^{\hat{K}}$, $\boldsymbol{D}_{c,m} \in \mathbb{R}^{\hat{K} \times \hat{K}}$, and $\boldsymbol{x}_m \in \mathbb{R}^{\hat{K}}$, again using $\hat{K}$ to represent the number of elements in one channel of the signal. When extended to multi-layer dictionaries, this structure closely aligns with that of convolutional neural networks, and the number of chan-

nels for a subsequent dictionary is the number of filters for the dictionary from the previous layer. The number of coefficients does not grow exponentially with the number of layers, but the number of channels can be large.

2.3 Pursuit and Sparse Coding

The dictionary model decomposes the signal $\boldsymbol{s}_n$ into a dictionary $\boldsymbol{D}$ (which generalizes to other signals) and the coefficients $\boldsymbol{x}_n$ (which are specific to the signal $\boldsymbol{s}_n$):

$$\boldsymbol{s}_n \approx \boldsymbol{D}\boldsymbol{x}_n \tag{2.14}$$

(Here the subscript n specifies a particular signal and its corresponding coefficients.) A pursuit algorithm finds the coefficients $\boldsymbol{x}_n$ corresponding to a particular signal $\boldsymbol{s}_n$ for known dictionary $\boldsymbol{D}$. If the number of dictionary atoms (columns) is larger than the dimension of the signal, then the number of unknowns is larger than the number of equations, and many solutions for $\boldsymbol{x}_n$ represent $\boldsymbol{s}_n$ equally well (at least in an L_2 sense). Researchers and practitioners commonly either impose a sparsity constraint on the coefficients or add a coefficient L_1 penalty to the objective function, which removes this ambiguity from the problem construction. When such a penalty or constraint is used, pursuit is sometimes called sparse coding. With the added coefficient L_1 penalty, the pursuit optimization problem looks like this:

$$\boldsymbol{x}_n = \arg\min_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{s}_n - \boldsymbol{D}\boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{x}\|_1 \tag{2.15}$$

where λ is a nonnegative hyperparameter controlling how much the L_1 norm of the coefficients is penalized. Researchers have proposed many ways to solve this problem. If the dictionary is convolutional and the number of channels is low, a standard approach is to use the Alternating Direction Method of Multipliers (ADMM) algorithm.

2.4 ADMM

ADMM is a convex-optimization algorithm used to solve the optimization problem:

$$\begin{aligned} &\underset{\boldsymbol{x},\boldsymbol{y}}{\text{minimize}} f(\boldsymbol{x}) + g(\boldsymbol{y}) \\ &\text{subject to } \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{y} + \boldsymbol{c} = \boldsymbol{0} \end{aligned} \tag{2.16}$$

where f and g are convex functions [16]. (I will address how to put the sparse coding problem in this form in the next section.)

The ADMM algorithm makes use of the augmented Lagrangian, a particular expression that has a saddle point at the solution to the constrained optimization problem:

$$\mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}) = f(\boldsymbol{x}) + g(\boldsymbol{y}) + \boldsymbol{u}^H(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{y} + \boldsymbol{c}) + \frac{\rho}{2}\|\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{y} + \boldsymbol{c}\|_2^2 \tag{2.17}$$

where ρ is a hyperparameter greater than zero and $\boldsymbol{u}$ is the dual variable for the constraints.

At the saddle-point solution, the augmented Lagrangian is at a minimum in respect to $\boldsymbol{x}$ and $\boldsymbol{y}$, but at a maximum in respect to $\boldsymbol{u}$.

The ADMM algorithm is an iterative search for the saddle point of the augmented Lagrangian. Each iteration consists of a primal update for $\boldsymbol{x}$, a primal update for $\boldsymbol{y}$, and a dual update for $\boldsymbol{u}$:

$$\boldsymbol{x}^{(t+1)} = \arg\min_{\boldsymbol{x}} \mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{y}^{(t)}, \boldsymbol{u}^{(t)}) \tag{2.18}$$

$$\boldsymbol{y}^{(t+1)} = \arg\min_{\boldsymbol{y}} \mathcal{L}_\rho(\boldsymbol{x}^{(t+1)}, \boldsymbol{y}, \boldsymbol{u}^{(t)}) \tag{2.19}$$

$$\boldsymbol{u}^{(t+1)} = \boldsymbol{u}^{(t)} + \rho(\boldsymbol{A}\boldsymbol{x}^{(t+1)} + \boldsymbol{B}\boldsymbol{y}^{(t+1)} + \boldsymbol{c}) \tag{2.20}$$

Algorithm 1: ADMM

$\boldsymbol{y} =$ initial guess

$\boldsymbol{u} = \mathbf{0}$

while *Not Converged* **do**

$\boldsymbol{x} = \arg\min_{\boldsymbol{x}} \mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u})$

$\boldsymbol{y} = \arg\min_{\boldsymbol{y}} \mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u})$

$\boldsymbol{u} = \boldsymbol{u} + \rho(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{y} + \boldsymbol{c})$

end

The primal updates serve to move towards the minimum of the augmented Lagrangian in respect to $\boldsymbol{x}$ and $\boldsymbol{y}$ with $\boldsymbol{u}$ fixed, and the dual update fixes $\boldsymbol{x}$ and $\boldsymbol{y}$, and performs gradient ascent on $\boldsymbol{u}$ with step size ρ. Under very mild assumptions, this process converges to a saddle point of the augmented Lagrangian, which matches a solution to the constrained optimization problem.[2] The ADMM algorithm is shown in Algorithm 1.

There are two common variations of the ADMM algorithm that this dissertation will make use of. The first is the scaled form, which comes from completing the square for the augmented lagrangian function:

$$\mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}) = f(\boldsymbol{x}) + g(\boldsymbol{y}) + \frac{\rho}{2}\|\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{y} + \boldsymbol{c} + \frac{\boldsymbol{u}}{\rho}\|_2^2 - \frac{\rho}{2}\|\frac{\boldsymbol{u}}{\rho}\|_2^2 \tag{2.21}$$

The term $-\frac{\rho}{2}\|\frac{\boldsymbol{u}}{\rho}\|_2^2$ can be ignored for the primal updates because it has no dependence on the primal variables. It is sometimes more convenient to keep track of $\boldsymbol{u}_{\text{sc}} = \frac{\boldsymbol{u}}{\rho}$ instead of $\boldsymbol{u}$, since that is the form that appears in the augmented Lagrangian after completing the square. The dual update for the scaled form is easily derived from equation 2.20.

$$\frac{\boldsymbol{u}^{(t+1)}}{\rho} = \frac{\boldsymbol{u}^{(t)}}{\rho} + \boldsymbol{A}\boldsymbol{x}^{(t+1)} + \boldsymbol{B}\boldsymbol{y}^{(t+1)} + \boldsymbol{c} \tag{2.22}$$

This form is known as scaled ADMM.

[2]Neither the saddle point nor the corresponding solution to the constrained optimization problem are guaranteed to be unique, however.

Another common variation of ADMM updates the dual variable more frequently [17].

$$\boldsymbol{x}^{(t+1)} = \arg\min_{\boldsymbol{x}} \mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{y}^{(t)}, \boldsymbol{u}^{(t)}) \tag{2.23}$$

$$\boldsymbol{u}^{(t+\frac{1}{2})} = \boldsymbol{u}^{(t)} + (\alpha - 1)\rho(\boldsymbol{A}\boldsymbol{x}^{(t+1)} + \boldsymbol{B}\boldsymbol{y}^{(t)} + \boldsymbol{c}) \tag{2.24}$$

$$\boldsymbol{y}^{(t+1)} = \arg\min_{\boldsymbol{y}} \mathcal{L}_\rho(\boldsymbol{x}^{(t+1)}, \boldsymbol{y}, \boldsymbol{u}^{(t+\frac{1}{2})}) \tag{2.25}$$

$$\boldsymbol{u}^{(t+1)} = \boldsymbol{u}^{(t+\frac{1}{2})} + \rho(\boldsymbol{A}\boldsymbol{x}^{(t+1)} + \boldsymbol{B}\boldsymbol{y}^{(t+1)} + \boldsymbol{c}) \tag{2.26}$$

When $\alpha > 1$, this is known as over-relaxation, and if $\alpha < 1$, this is known as under-relaxation.[3] α is always chosen to be greater than zero. In some applications, researchers have found using over-relaxation converges faster than without over-relaxation [17], but optimal choice of α is problem-dependent [18]. Scaled ADMM with over-relaxation is

[3] I have elected to notate over/under relaxation differently than standard notation, but the α is the same, and the notations are mathematically equivalent. The standard notation does not use the first dual update, and instead includes another variable $\boldsymbol{h}^{(t+1)} = \boldsymbol{A}\boldsymbol{x}^{(t+1)} - (1-\alpha)(\boldsymbol{A}\boldsymbol{x}^{(t+1)} + \boldsymbol{B}\boldsymbol{y}^{(t)} + \boldsymbol{c})$ and substitutes $\boldsymbol{h}^{(t+1)}$ for $\boldsymbol{A}\boldsymbol{x}^{(t+1)}$ in the dual-update equation and the second primal-update equation. While more familiar to readers who have dealt with ADMM before, this standard notation complicates ADMM with an extra variable and obscures how the dual update and second primal update relate to the augmented Lagrangian.

shown in Algorithm 2.

Algorithm 2: Scaled ADMM With Over or Under-Relaxation

$\alpha \in (0, 2]$

$\boldsymbol{y} =$ initial guess

$\boldsymbol{u}_{\text{sc}} = \mathbf{0}$

while *Not Converged* **do**

$\boldsymbol{x} = \arg\min_{\boldsymbol{x}} \mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{y}, \rho\boldsymbol{u}_{\text{sc}})$

$\boldsymbol{u}_{\text{sc}} = \boldsymbol{u}_{\text{sc}} + (\alpha - 1)(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{y} + \boldsymbol{c})$

$\boldsymbol{y} = \arg\min_{\boldsymbol{y}} \mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{y}, \rho\boldsymbol{u}_{\text{sc}})$

$\boldsymbol{u}_{\text{sc}} = \boldsymbol{u}_{\text{sc}} + \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{y} + \boldsymbol{c}$

end

2.5 Applying ADMM to the Sparse Coding Problem

Recall from section 2.3, equation 2.15 for sparse coding.

$$\boldsymbol{x}_i = \arg\min_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{s}_i - \boldsymbol{D}\boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{x}\|_1 \tag{2.27}$$

This can be rewritten to match the ADMM form from equation 2.16:

$$\begin{aligned} &\underset{\boldsymbol{x},\boldsymbol{y}}{\text{minimize}} \frac{1}{2}\|\boldsymbol{s}_i - \boldsymbol{D}\boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{y}\|_1 \\ &\text{subject to } \boldsymbol{y} - \boldsymbol{x} = \mathbf{0} \end{aligned} \tag{2.28}$$

Using his form, the ADMM algorithm can be used to solve the sparse coding problem, as shown in Algorithm 3. Given sufficient iterations, $\boldsymbol{x}$ and $\boldsymbol{y}$ will both be close to the optimal, but they may not be equal. Either can be used as an approximate solution to the sparse coding problem.

Computing the augmented Lagrangian of the convex optimization problem in expres-

sion 2.28 yields the following equation:

$$\mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}) = \frac{1}{2}\|\boldsymbol{s}_i - \boldsymbol{D}\boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{y}\|_1 + \frac{\rho}{2}\|\boldsymbol{y} - \boldsymbol{x} + \frac{\boldsymbol{u}}{\rho}\|_2^2 - \frac{1}{2\rho}\|\boldsymbol{u}\|_2^2 \tag{2.29}$$

Starting with the $\boldsymbol{x}$-update:

$$\boldsymbol{x}^{(t+1)} = \arg\min_{\boldsymbol{x}} \mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{y}^{(t)}, \boldsymbol{u}^{(t)}) \tag{2.30}$$

Since the desired result is the minimizer, setting the gradient to zero and solving for $\boldsymbol{x}$ will produce the solution.

$$\nabla_{\boldsymbol{x}^{(t+1)}} \mathcal{L}_\rho(\boldsymbol{x}^{(t+1)}, \boldsymbol{y}(t), \boldsymbol{u}(t)) = \mathbf{0} \tag{2.31}$$

$$\mathbf{0} = \boldsymbol{D}^T\boldsymbol{D}\boldsymbol{x}^{(t+1)} - \boldsymbol{D}^T\boldsymbol{s}_i + \rho\boldsymbol{x}^{(t+1)} - \rho\left(\boldsymbol{y}^{(t)} + \frac{\boldsymbol{u}^{(t)}}{\rho}\right) \tag{2.32}$$

$$(\rho\mathbf{I} + \boldsymbol{D}^T\boldsymbol{D})\boldsymbol{x}^{(t+1)} = \boldsymbol{D}^T\boldsymbol{s}_i + \rho\left(\boldsymbol{y}^{(t)} + \frac{\boldsymbol{u}^{(t)}}{\rho}\right) \tag{2.33}$$

$$\boldsymbol{x}^{(t+1)} = (\rho\mathbf{I} + \boldsymbol{D}^T\boldsymbol{D})^{-1}\left(\boldsymbol{D}^T\boldsymbol{s}_i + \rho\left(\boldsymbol{y}^{(t)} + \frac{\boldsymbol{u}^{(t)}}{\rho}\right)\right) \tag{2.34}$$

In subsection 2.5.1, there is a discussion of the implications of this update equation, how to compute it for cases in which the signal has a low number of channels, and the challenges it poses for signals with many channels.

If using over-relaxation[4], there is a dual update:

$$\frac{\boldsymbol{u}^{(t+\frac{1}{2})}}{\rho} = \frac{\boldsymbol{u}^{(t)}}{\rho} + (\alpha - 1)(\boldsymbol{y}^{(t)} - \boldsymbol{x}^{(t+1)}) \tag{2.35}$$

[4] or under-relaxation

Moving on to the $\boldsymbol{y}$-update:

$$\boldsymbol{y}^{(t+1)} = \arg\min_{\boldsymbol{y}} \mathrm{L}_\rho(\boldsymbol{x}^{(t+1)}, \boldsymbol{y}, \boldsymbol{u}^{(t+\frac{1}{2})}) \tag{2.36}$$

Excluding the terms that don't include $\boldsymbol{y}$ yields

$$\boldsymbol{y}^{(t+1)} = \arg\min_{\boldsymbol{y}} \lambda\|\boldsymbol{y}\|_1 + \frac{\rho}{2}\|\boldsymbol{y} - \boldsymbol{x}^{(t+1)} + \frac{\boldsymbol{u}^{(t+\frac{1}{2})}}{\rho}\|_2^2 \tag{2.37}$$

This is a well-known problem, whose solution is

$$\boldsymbol{y}^{(t+1)} = \mathrm{S}_{\frac{\lambda}{\rho}}(\boldsymbol{x}^{(t+1)} - \frac{\boldsymbol{u}^{(t+\frac{1}{2})}}{\rho}) \tag{2.38}$$

where S is the shrinkage operator:

$$\mathrm{S}_b(x) = \begin{cases} x - b & x > b \\ 0 & -b < x < b \\ x + b & x < -b \end{cases} \tag{2.39}$$

In the case of a vector, matrix, or tensor input, the shrinkage operator is applied element by element.

Finally, the last update equation for the dual variable:

$$\frac{\boldsymbol{u}^{(t+1)}}{\rho} = \frac{\boldsymbol{u}^{(t+\frac{1}{2})}}{\rho} + \boldsymbol{y}^{(t+1)} - \boldsymbol{x}^{(t+1)} \tag{2.40}$$

Algorithm 3: ADMM for Sparse Coding

$\alpha \in (0, 2]$

$\boldsymbol{y} = \boldsymbol{D}^T \boldsymbol{s}$

$\boldsymbol{u}_{\text{sc}} = \mathbf{0}$

while *Not Converged* **do**

$\boldsymbol{x} = \left(\rho \mathbf{I} + \boldsymbol{D}^T \boldsymbol{D}\right)^{-1} \left(\boldsymbol{D}^T \boldsymbol{s} + \rho(\boldsymbol{y} + \boldsymbol{u}_{\text{sc}})\right)$

$\boldsymbol{u}_{\text{sc}} = \boldsymbol{u}_{\text{sc}} + (\alpha - 1)(\boldsymbol{y} - \boldsymbol{x})$

$\boldsymbol{y} = \mathrm{S}_{\frac{\lambda}{\rho}}(\boldsymbol{x} - \boldsymbol{u}_{\text{sc}})$

$\boldsymbol{u}_{\text{sc}} = \boldsymbol{u}_{\text{sc}} + \boldsymbol{y} - \boldsymbol{x}$

end

2.5.1 Exploiting Dictionary Structure for the Inverse Problem

Returning to the $\boldsymbol{x}$ update:

$$\boldsymbol{x}^{(t+1)} = \left(\rho \mathbf{I} + \boldsymbol{D}^T \boldsymbol{D}\right)^{-1} \left(\boldsymbol{D}^T \boldsymbol{s}_i + \rho \left(\boldsymbol{y}^{(t)} + \frac{\boldsymbol{u}^{(t)}}{\rho}\right)\right) \tag{2.41}$$

For problems using a dictionary with convolutional structure, this inverse for the convolutional sparse coding problem is very structured. Exploiting this structure is important for efficient computation, because the matrix $\rho \mathbf{I} + \boldsymbol{D}^T \boldsymbol{D}$ is a large matrix.

Writing $\boldsymbol{D}$ in a block structure, I have

$$\boldsymbol{D} = \begin{bmatrix} \boldsymbol{D}_{1,1} & \dots & \boldsymbol{D}_{1,M} \\ \vdots & \ddots & \vdots \\ \boldsymbol{D}_{C,1} & \dots & \boldsymbol{D}_{C,M} \end{bmatrix} \tag{2.42}$$

where $\boldsymbol{D}_{c,m}$ is a circulant matrix capturing channel c of the m^{th} filter of the dictionary.

Circulant matrices are diagonalizable with Fourier eigenvectors:

$$\boldsymbol{D} = \begin{bmatrix} \mathcal{F}^{-1}\hat{\boldsymbol{D}}_{1,1}\mathcal{F} & \dots & \mathcal{F}^{-1}\hat{\boldsymbol{D}}_{1,M}\mathcal{F} \\ \vdots & \ddots & \vdots \\ \mathcal{F}^{-1}\hat{\boldsymbol{D}}_{C,1}\mathcal{F} & \dots & \mathcal{F}^{-1}\hat{\boldsymbol{D}}_{C,M}\mathcal{F} \end{bmatrix} \tag{2.43}$$

where $\hat{\boldsymbol{D}}_{c,m}$ is a diagonal matrix whose elements are the discrete Fourier transform (FFT) of channel c of the m^{th} dictionary filter.

This sparsely banded structure is a useful form in analyzing the structure of the inverse problem:

$$(\rho\mathbf{I} + \boldsymbol{D}^T\boldsymbol{D})^{-1} = \mathcal{F}^{-1}(\rho\mathbf{I} + \hat{\boldsymbol{D}}^H\hat{\boldsymbol{D}})^{-1}\mathcal{F} \tag{2.44}$$

where

$$\hat{\boldsymbol{D}} = \begin{bmatrix} \hat{\boldsymbol{D}}_{1,1}, & \dots, & \hat{\boldsymbol{D}}_{1,M} \\ \vdots & \ddots & \vdots \\ \hat{\boldsymbol{D}}_{C,1}, & \dots, & \hat{\boldsymbol{D}}_{C,M} \end{bmatrix} \tag{2.45}$$

and in a slight abuse of notation, $\mathcal{F}$ computes the FFT separately on the coefficients for each filter. In [19], Bristow et al. observe the matrix $\rho\mathbf{I} + \hat{\boldsymbol{D}}^H\hat{\boldsymbol{D}}$ is sparsely banded, so the inverse can be broken down into much smaller inverse problems, and one only needs to compute the inverse of an $M \times M$ matrix for every element in the signal. ($\rho\mathbf{I} + \hat{\boldsymbol{D}}^H\hat{\boldsymbol{D}}$) is an $M \times M$ block matrix, whose blocks are diagonal. Each submatrix collects one element from the diagonal of each of the blocks.) This inverse can be calculated in $\mathcal{O}(\hat{K}M^3)$ time.

Furthermore, the maximum rank of these submatrices is C, so if C is small, these inverses can be computed even more efficiently using the Woodbury matrix identity or Sherman-Morrison equations [20] [21] [22].

According to the Woodbury matrix identity [23], for any invertible matrix $\boldsymbol{U}$ and any matrix $\boldsymbol{V}$:

$$(\boldsymbol{U} + \boldsymbol{V}^H\boldsymbol{V})^{-1} = \boldsymbol{U}^{-1} - \boldsymbol{U}^{-1}\boldsymbol{V}^H(\mathbf{I} + \boldsymbol{V}\boldsymbol{U}^{-1}\boldsymbol{V}^H)^{-1}\boldsymbol{V}\boldsymbol{U}^{-1} \tag{2.46}$$

So,

$$(\rho\mathbf{I} + \hat{\boldsymbol{D}}^H\hat{\boldsymbol{D}})^{-1} = \frac{1}{\rho}\mathbf{I} - \frac{1}{\rho}\hat{\boldsymbol{D}}^H(\rho\mathbf{I} + \hat{\boldsymbol{D}}\hat{\boldsymbol{D}}^H)^{-1}\hat{\boldsymbol{D}} \tag{2.47}$$

This means that instead of computing inverse of $\hat{K}$ $M \times M$ matrices (where K is the number of elements in a single channel of the signal), one could instead choose to compute the inverse of $\hat{K}$ $C \times C$ matrices. Rather than computing an inverse explicitly, it is generally preferable to calculate a Cholesky or LDLT decomposition instead, and the efficiency gains due to the Woodbury matrix identity are relevant regardless of the chosen representation. The inverse or matrix decomposition can be computed in $\mathcal{O}(\hat{K}C^3)$ time.

2.6 Sparse Coding for Multi-Channel Signals: Alternatives to My Novel Approach

In applying ADMM to the convolutional sparse coding problem, [20] [21] [22] exploit the low-rank structure of the inverse problem in the $\boldsymbol{x}$ update for efficient computation. Unfortunately, this relies on the number of channels being small, as the rank corresponds to the minimum of the number of channels C and the number of filters M. Broadly, there are two main approaches to avoid or simplify this challenging inverse problem: either construct a variant of the ADMM algorithm that simplifies the inverse problem, or use a proximal gradient approach that avoids it altogether.

In [13][12], the authors use the ADMM algorithm for sparse coding. They observe that if the dictionary is a tight frame, that is, $\boldsymbol{D}\boldsymbol{D}^T = \mathbf{I}$, then the inverse can be simplified without using the frequency representation.

$$(\rho\mathbf{I} + \boldsymbol{D}^T\boldsymbol{D})^{-1} = \frac{1}{\rho}\mathbf{I} - \frac{1}{\rho(\rho+1)}\boldsymbol{D}^T\boldsymbol{D} \tag{2.48}$$

This produces the $\boldsymbol{x}$ update equation:

$$\boldsymbol{x}^{(t+1)} = \frac{1}{\rho+1}\boldsymbol{D}^T\boldsymbol{s} + \left(\mathbf{I} - \frac{1}{\rho+1}\boldsymbol{D}^T\boldsymbol{D}\right)\left(\boldsymbol{z}^{(t)} - \frac{\boldsymbol{\gamma}^{(t)}}{\rho}\right) \tag{2.49}$$

The above equations can be derived using the Woodbury matrix identity. In their work, they use the equations built on the assumption that the dictionary is a tight frame, but develop no mechanism to ensure that their assumption is accurate. Thus, ultimately $\frac{1}{\rho}\mathbf{I} - \frac{1}{\rho(\rho+1)}\boldsymbol{D}^T\boldsymbol{D}$ merely serves as an approximation to $(\rho\mathbf{I} + \boldsymbol{D}^T\boldsymbol{D})^{-1}$. Empirically, they observe that the algorithm converges, but the dictionaries they learn are not tight frames, so the solution they converge to is not optimal[5].

Other works avoid the ADMM algorithm entirely.

The iterative shrinkage thresholding algorithm (ISTA) is an iterative algorithm that minimizes the sum of two convex functions f and g. f is required to be smooth. It is helpful for f to be easily differentiable and g to have a simple proximal operator.

$$\text{prox}_g(\boldsymbol{\mu}) = \arg\min_{\boldsymbol{\nu}} \frac{1}{2}\|\boldsymbol{\nu} - \boldsymbol{\mu}\|_2^2 + g(\boldsymbol{\nu}) \tag{2.50}$$

Then, ISTA has the following update equation, where the constant $\mathbb{L}$ controls step size.

$$\boldsymbol{x}^{(t+1)} = \text{prox}_{\frac{g}{\mathbb{L}}}\left(\boldsymbol{x}^{(t)} - \frac{1}{\mathbb{L}}\nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(t)})\right) \tag{2.51}$$

FISTA is similar to ISTA, but adds momentum [24].

$$\boldsymbol{z}^{(t+1)} = \text{prox}_{\frac{g}{\mathbb{L}}}\left(\boldsymbol{x}^{(t)} - \frac{1}{\mathbb{L}}\nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(t)})\right) \tag{2.52}$$

$$r^{(t+1)} = \frac{1}{2}\left(1 + \sqrt{1 + 4(r^{(t)})^2}\right) \tag{2.53}$$

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{z}^{(t+1)} + \frac{r^{(t)} - 1}{r^{(t+1)}}(\boldsymbol{z}^{(t+1)} - \boldsymbol{x}^{(t)}) \tag{2.54}$$

Applying FISTA to the sparse coding problem, $\frac{1}{2}\|\boldsymbol{s} - \boldsymbol{D}\boldsymbol{x}\|_2^2$ is straightforward to differen-

[5]The solution does not minimize the sparse coding objective function.

tiate and $\lambda\|\boldsymbol{x}\|_1$ has a simple proximal operator.

$$\nabla_{\boldsymbol{x}} \left(\frac{1}{2} \|\boldsymbol{s} - \boldsymbol{D}\boldsymbol{x}\|_2^2 \right) = \boldsymbol{D}^T \boldsymbol{D}\boldsymbol{x} - \boldsymbol{D}^T \boldsymbol{s} \tag{2.55}$$

$$\text{prox}_{\frac{\lambda}{\mathbb{L}}\|\cdot\|_1}(\cdot) = \text{S}_{\frac{\lambda}{\mathbb{L}}} \tag{2.56}$$

So, the FISTA equations for convolutional basis pursuit are the following:

$$\boldsymbol{z}^{(t+1)} = \text{S}_{\frac{\lambda}{\mathbb{L}}} \left(\boldsymbol{x}^{(t)} - \frac{1}{\mathbb{L}} \boldsymbol{D}^T (\boldsymbol{D}\boldsymbol{x}^{(t)} - \boldsymbol{s}) \right) \tag{2.57}$$

$$r^{(t+1)} = \frac{1}{2} \left(1 + \sqrt{1 + 4(r^{(t)})^2} \right) \tag{2.58}$$

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{z}^{(t+1)} + \frac{r^{(t)} - 1}{r^{(t+1)}} (\boldsymbol{z}^{(t+1)} - \boldsymbol{x}^{(t)}) \tag{2.59}$$

In [22], Wohlberg compares FISTA to ADMM on a sparse coding task and finds FISTA converges much slower than ADMM. However, the comparison is made on signals with few channels, so ADMM is able to exploit the structure of $\boldsymbol{D}$ for efficient $\boldsymbol{x}$ updates.

In a recent work [8], Chodosh and Lucey derive prox-linear updates using convex solver methods detailed in [25]. The updates come from the formula:

$$\boldsymbol{z}^{(t+1)} = \arg\min_{\boldsymbol{z}} \left(\nabla f(\boldsymbol{x}^{(t)}) \right)^T (\boldsymbol{z} - \boldsymbol{x}^{(t)}) + \frac{\mathbb{L}}{2} \|\boldsymbol{z} - \boldsymbol{x}^{(t)}\|_2^2 + \lambda \|\boldsymbol{z}\|_1 \tag{2.60}$$

where $\boldsymbol{x}^{(t)} = \boldsymbol{z}^{(t)} + \omega^{(t)}(\boldsymbol{z}^{(t)} - \boldsymbol{z}^{(t-1)})$ and $\omega^{(t)}$ is a momentum factor. This yields the

update equation[6]:

$$\boldsymbol{z}^{(t+1)} = \mathrm{S}_{\frac{\lambda}{\mathbb{L}}}\left(\boldsymbol{x}^{(t)} - \frac{1}{\mathbb{L}}\boldsymbol{D}^T(\boldsymbol{D}\boldsymbol{x}^{(t)} - \boldsymbol{s})\right) \tag{2.61}$$

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{z}^{(t+1)} + \omega^{(t)}(\boldsymbol{z}^{(t+1)} - \boldsymbol{z}^{(t)}) \tag{2.62}$$

While neither Chodosh and Lucey, nor the work they cite, mention FISTA, the resemblance is very close. The only distinction is in the momentum step. Given these similarities, it is likely the performance between the two methods is similar.

2.7 Dictionary Learning

The last few sections have focused on sparse coding. For sparse coding, the dictionary is fixed or known. Most dictionary learning algorithms alternate between pursuing coefficients and updating dictionary filters, as shown in Algorithms 4 and 5.[7] The CoefficientUpdate in Algorithm 5 may consist of one or more sparse coding update steps. Not all dictionary learning algorithms perfectly adhere to this structure [26], but the template covers most approaches.

Algorithm 4: Online Dictionary Learning Algorithm

$\boldsymbol{D} = \boldsymbol{D}_{\text{init}}$
$i = 0$ **while** *Stopping Criteria Not Met* **do**
  $\boldsymbol{s} = \mathrm{GetData}(i)$
  $\boldsymbol{x} = \mathrm{Pursuit}(\boldsymbol{D}, \boldsymbol{s})$
  $\boldsymbol{D} = \mathrm{DictionaryUpdate}(\boldsymbol{D}, \boldsymbol{s}, \boldsymbol{x})$
  $i = i + 1$
end

[6] In their paper, they add a non-negativity constraint and allow different λ for the coefficients of each filter (and possibly spatially varied as well). They also are constructing the equations specifically for a multi-layer network. Those modifications relate to their objective function and constraints, not their minimization algorithm, so I rewrote their equations without those modifications to illustrate how their approach relates to the FISTA algorithm.

[7] The capital $\boldsymbol{S}$ in Algorithm 5 represents the entire collection of signal samples, as opposed to a single signal $\boldsymbol{s}$.

Algorithm 5: Batch Dictionary Learning Algorithm

$\boldsymbol{D} = \boldsymbol{D}_{\text{init}}$
$\boldsymbol{X} = \boldsymbol{X}_{\text{init}}$
while *Stopping Criteria Not Met* **do**
  $\boldsymbol{X} = \text{CoefficientUpdate}(\boldsymbol{D}, \boldsymbol{X}, \boldsymbol{S})$
  $\boldsymbol{D} = \text{DictionaryUpdate}(\boldsymbol{D}, \boldsymbol{S}, \boldsymbol{X})$
end

There are many methods to updating dictionaries: FISTA, ADMM, projected stochastic gradient descent, et cetera. Good dictionaries tend to have the following characteristics:

1. Good dictionaries are able to represent the data well. That is

$$\boldsymbol{s} \approx \boldsymbol{D}\boldsymbol{x} \tag{2.63}$$

   for some sparse coefficients $\boldsymbol{x}$.

2. Furthermore, it is desirable for the dictionary to be normalized.

$$\sum_{c=1}^{C} \|\boldsymbol{d}_{c,m}\|_2^2 = 1 \tag{2.64}$$

   where $\boldsymbol{d}_{c,m}$ is the first column of $\boldsymbol{D}_{c,m}$, that is, the c^{th} channel of zero-padded dictionary filter $\boldsymbol{f}_m$.

3. Specifically, convolutional dictionary filters are typically spatially or temporally constrained.

$$(\mathbf{I} - \boldsymbol{T})\boldsymbol{d}_{c,m} = \mathbf{0} \tag{2.65}$$

   (Here, $\mathbf{I} - \boldsymbol{T}$ selects the elements of $\boldsymbol{d}_{c,m}$ that must be zero.) Generally, dictionary updates do not increase the size of the filters.

Dictionary updates tend to improve data representation while maintaining normalization and spatial or temporal constraints. If using ADMM for pursuit, every time the dictionary is updated, the inverse representation must be updated as well. The inverse representa-

tion can be updated more efficiently if the dictionary update is low rank, but approximating a dictionary update using a truncated singular value decomposition would forfeit characteristic 2. The next section describes a novel means to handle these challenges, with an explanation of how to efficiently update the inverse representation and a novel sparse coding method designed to handle dictionary updates that produce unnormalized dictionaries.

2.8 A Novel Approach to Sparse Coding: ADMM with Low-Rank Dictionary Updates

In this section, I present a novel approach to sparse coding for signals with a large number of channels. The approach uses the ADMM algorithm described in section 2.4 and will share many similarities to the standard ADMM sparse coding approach described in section 2.5 for signals with few channels.

2.8.1 Updating the Inverse Representation

Under many circumstances, inverse representations can be updated efficiently, provided the update adheres to a low-rank structure. Recall the frequency representation of the convolutional dictionary:

$$\hat{\boldsymbol{D}} = \begin{bmatrix} \hat{\boldsymbol{D}}_{1,1} & \dots & \hat{\boldsymbol{D}}_{1,M} \\ \vdots & \ddots & \vdots \\ \hat{\boldsymbol{D}}_{C,1} & \dots & \hat{\boldsymbol{D}}_{C,M} \end{bmatrix} \tag{2.66}$$

where $\hat{\boldsymbol{D}}_{c,m}$ is diagonal for all c and m. Let $\hat{\boldsymbol{D}}_{c,m}[\hat{k}]$ be the $\hat{k}^{\text{th}}$ element of the diagonal and let

$$\hat{\boldsymbol{D}}[\hat{k}] = \begin{bmatrix} \hat{\boldsymbol{D}}_{1,1}[\hat{k}] & \dots & \hat{\boldsymbol{D}}_{1,M}[\hat{k}] \\ \vdots & \ddots & \vdots \\ \hat{\boldsymbol{D}}_{C,1}[\hat{k}] & \dots & \hat{\boldsymbol{D}}_{C,M}[\hat{k}] \end{bmatrix} \tag{2.67}$$

Then $\hat{\boldsymbol{D}}[\hat{k}]$ is a $C \times M$ matrix collecting the $\hat{k}^{\text{th}}$ frequency of all channels and filters of $\boldsymbol{D}$. Thus, $(\rho\mathbf{I}+\hat{\boldsymbol{D}}^H\hat{\boldsymbol{D}})^{-1}$ really consists of $\hat{K}$ separate inverse problems: $(\rho\mathbf{I}+\hat{\boldsymbol{D}}^H[\hat{k}]\hat{\boldsymbol{D}}[\hat{k}])^{-1}$.

Consider the update equation.

$$\hat{\boldsymbol{D}}[\hat{k}]^{(n+1)} = \hat{\boldsymbol{D}}[\hat{k}]^{(n)} + \hat{\boldsymbol{U}}\hat{\boldsymbol{V}}[\hat{k}]^H \tag{2.68}$$

where $\hat{\boldsymbol{U}}$ is an orthogonal matrix of size $C \times \text{Я}$ and $\hat{\boldsymbol{V}}[\hat{k}]$ is an matrix of size $M \times \text{Я}$.[8]

Then,

$$\rho\mathbf{I} + (\hat{\boldsymbol{D}}^{(n+1)})^H[\hat{k}]\hat{\boldsymbol{D}}^{(n+1)}[\hat{k}] = \rho\mathbf{I} + (\boldsymbol{D}^{(n)}[\hat{k}] + \hat{\boldsymbol{U}}\hat{\boldsymbol{V}}^H[\hat{k}])^H(\hat{\boldsymbol{D}}^{(n)}[\hat{k}] + \hat{\boldsymbol{U}}\hat{\boldsymbol{V}}^H[\hat{k}]) \tag{2.69}$$

For brevity and simplicity, I will drop the notation indexing the frequency $\hat{k}$ and selecting the iteration n for matrix $\hat{\boldsymbol{D}}^{(n)}[\hat{k}]$ and simply use $\hat{\boldsymbol{D}}$ instead. However, the reader should keep in mind the $\hat{\boldsymbol{D}}$ here is a dense $C \times M$ matrix capturing the component of the dictionary $\boldsymbol{D}$ for implicit frequency $\hat{k}$, only a submatrix of the sparsely banded $\hat{\boldsymbol{D}}$ of size $\hat{K}C \times M$ from earlier in this section.

$$\rho\mathbf{I} + (\hat{\boldsymbol{D}}^{(n+1)})^H\hat{\boldsymbol{D}}^{(n+1)} = \rho\mathbf{I} + (\hat{\boldsymbol{D}}^{(n)})^H\hat{\boldsymbol{D}}^{(n)} + \hat{\boldsymbol{V}}\hat{\boldsymbol{U}}^H\hat{\boldsymbol{U}}\hat{\boldsymbol{V}}^H + \hat{\boldsymbol{V}}\hat{\boldsymbol{U}}^H\hat{\boldsymbol{D}}^{(n)} + (\hat{\boldsymbol{D}}^{(n)})^H\hat{\boldsymbol{U}}\hat{\boldsymbol{V}}^H \tag{2.70}$$

Given that $\hat{\boldsymbol{U}}$ is an orthogonal matrix, $\hat{\boldsymbol{V}}\hat{\boldsymbol{U}}^H\hat{\boldsymbol{U}}\hat{\boldsymbol{V}}^H$ can easily be broken into Я rank-one Hermitian updates.

$$\hat{\boldsymbol{V}}\hat{\boldsymbol{U}}^H\hat{\boldsymbol{U}}\hat{\boldsymbol{V}}^H = \sum_{i=1}^{\text{Я}} \boldsymbol{u}_i^H\boldsymbol{u}_i\boldsymbol{v}_i\boldsymbol{v}_i^H \tag{2.71}$$

Similarly, $\hat{\boldsymbol{V}}\hat{\boldsymbol{U}}^H\hat{\boldsymbol{D}} + \hat{\boldsymbol{D}}^H\hat{\boldsymbol{U}}\hat{\boldsymbol{V}}^H$ can be broken into Я Hermitian, rank-two updates:

$$\hat{\boldsymbol{V}}\hat{\boldsymbol{U}}^H\hat{\boldsymbol{D}} + \hat{\boldsymbol{D}}^H\hat{\boldsymbol{U}}\hat{\boldsymbol{V}}^H = \sum_{\ell=1}^{\text{Я}} \left(\boldsymbol{v}_i\boldsymbol{u}_i^H\hat{\boldsymbol{D}} + \hat{\boldsymbol{D}}^H\boldsymbol{u}_i\boldsymbol{v}_i^H\right) \tag{2.72}$$

[8] $\hat{\boldsymbol{U}}$ and $\hat{\boldsymbol{V}}$ are also iteration specific, but to notate that would over-clutter the equations. For efficient, low-rank updates to the inverse representation, it is possible for both $\hat{\boldsymbol{U}}$ and $\hat{\boldsymbol{V}}$ to vary in respect to frequency $\hat{k}$ (instead of just $\boldsymbol{V}$). However, most convolutional models limit spatial or temporal support of the dictionary (so that the filter size is smaller than the signal), and preventing $\boldsymbol{U}$ from varying across frequency is part of a means to satisfy that constraint.

Inverse representations can be efficiently updated if the update is Hermitian and rank one. The details of such updates are discussed in Appendix A.

The Hermitian rank-two update consists of two rank-one terms, but the terms are not Hermitian, complicating the update process. However, this can be resolved through eigen-decomposition.

$$\boldsymbol{v}_i \boldsymbol{u}_i^H \hat{\boldsymbol{D}} + \hat{\boldsymbol{D}} \boldsymbol{u}_i \boldsymbol{v}_i^H = \begin{bmatrix} \boldsymbol{v}_i & \hat{\boldsymbol{D}}^H \boldsymbol{u}_i \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{D}}^H \boldsymbol{u}_i & \boldsymbol{v}_i \end{bmatrix}^H \tag{2.73}$$

While matrix products are not commutative, some of the eigenvalues of matrix products are commutative. Furthermore, for general matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ the eigenvectors of $\boldsymbol{AB}$ and $\boldsymbol{BA}$ are related:

$$\boldsymbol{BA}\boldsymbol{\omega} = \tau \boldsymbol{\omega} \implies \boldsymbol{ABA}\boldsymbol{\omega} = \tau \boldsymbol{A}\boldsymbol{\omega} \tag{2.74}$$

where τ is the eigenvalue and $\boldsymbol{\omega}$ is a vector.[9] So, if $\boldsymbol{\omega}$ is an eigenvector of $\boldsymbol{BA}$, $\boldsymbol{A\omega}$ is an eigenvector of $\boldsymbol{AB}$.

$$\begin{bmatrix} \hat{\boldsymbol{D}}^H \boldsymbol{u}_i & \boldsymbol{v}_i \end{bmatrix}^H \begin{bmatrix} \boldsymbol{v}_i & \hat{\boldsymbol{D}}^H \boldsymbol{u}_i \end{bmatrix} = \begin{bmatrix} \boldsymbol{u}_i^H \hat{\boldsymbol{D}} \boldsymbol{v}_i & \boldsymbol{u}_i^H \hat{\boldsymbol{D}} \hat{\boldsymbol{D}}^H \boldsymbol{u}_i \\ \boldsymbol{v}_i^H \boldsymbol{v}_i & \boldsymbol{v}_i^H \hat{\boldsymbol{D}}^H \boldsymbol{u}_i \end{bmatrix} \tag{2.75}$$

The eigenvalues and corresponding eigenvectors of a 2×2 matrix can be computed using the quadratic formula. Assuming that the 2×2 matrix has 2 distinct eigenvalues, the expressions for these are below.[10]

$$\text{eigval}\left(\begin{bmatrix} a & b \\ c & a^* \end{bmatrix} \right) = \text{real}(a) \pm \sqrt{bc - (\text{imag}(a))^2} \tag{2.76}$$

[9] In equation 2.74, some variables are repurposed to explain some general eigenvector relationships for matrix products. The reader should be careful not confuse $\boldsymbol{A}$ and $\boldsymbol{B}$ for the matrices in the ADMM constraints.

[10] It is not guaranteed the 2×2 matrix will have 2 distinct eigenvalues. In Appendix B, I consider those cases.

$$\text{eigvec}\left(\begin{bmatrix} a & b \\ c & a^* \end{bmatrix}\right) = \begin{bmatrix} b \\ -j\,\text{imag}(a) \pm \sqrt{bc - (\text{imag}(a))^2} \end{bmatrix} \tag{2.77}$$

For the sake of brevity, I will drop the subscripts for $\boldsymbol{u}$ and $\boldsymbol{v}$.

Letting $\eta_{\boldsymbol{u}} = \|\hat{\boldsymbol{D}}^H\boldsymbol{u}\|_2^2$, $\eta_{\boldsymbol{v}} = \|\boldsymbol{v}\|_2^2$, and $\eta_{\boldsymbol{u},\boldsymbol{v}} = \boldsymbol{u}^H\hat{\boldsymbol{D}}\boldsymbol{v}$:

$$\text{eigval}\left(\begin{bmatrix} \eta_{\boldsymbol{u},\boldsymbol{v}} & \eta_{\boldsymbol{u}} \\ \eta_{\boldsymbol{v}} & \eta_{\boldsymbol{u},\boldsymbol{v}}^* \end{bmatrix}\right) = \text{real}(\eta_{\boldsymbol{u},\boldsymbol{v}}) \pm \sqrt{\eta_{\boldsymbol{v}}\eta_{\boldsymbol{u}} - (\text{imag}(\eta_{\boldsymbol{u},\boldsymbol{v}}))^2} \tag{2.78}$$

$$\text{eigvec}\left(\begin{bmatrix} \eta_{\boldsymbol{u},\boldsymbol{v}} & \eta_{\boldsymbol{u}} \\ \eta_{\boldsymbol{v}} & \eta_{\boldsymbol{u},\boldsymbol{v}}^* \end{bmatrix}\right) = \begin{bmatrix} \eta_{\boldsymbol{u}} \\ -j\,\text{imag}(\eta_{\boldsymbol{u},\boldsymbol{v}}) \pm \sqrt{\eta_{\boldsymbol{v}}\eta_{\boldsymbol{u}} - (\text{imag}(\eta_{\boldsymbol{u},\boldsymbol{v}}))^2} \end{bmatrix} \tag{2.79}$$

Therefore,

$$\text{eigvec}(\boldsymbol{v}\boldsymbol{u}^H\hat{\boldsymbol{D}} + \hat{\boldsymbol{D}}^H\boldsymbol{u}\boldsymbol{v}^H) = \eta_{\boldsymbol{u}}\boldsymbol{v} + \left(-j\,\text{imag}(\eta_{\boldsymbol{u},\boldsymbol{v}}) \pm \sqrt{\eta_{\boldsymbol{v}}\eta_{\boldsymbol{u}} - (\text{imag}(\eta_{\boldsymbol{u},\boldsymbol{v}}))^2}\right)\hat{\boldsymbol{D}}^H\boldsymbol{u} \tag{2.80}$$

$$\text{eigval}(\boldsymbol{v}\boldsymbol{u}^H\hat{\boldsymbol{D}} + \hat{\boldsymbol{D}}^H\boldsymbol{u}\boldsymbol{v}^H) = \text{real}(\eta_{\boldsymbol{u},\boldsymbol{v}}) \pm \sqrt{\eta_{\boldsymbol{v}}\eta_{\boldsymbol{u}} - (\text{imag}(\eta_{\boldsymbol{u},\boldsymbol{v}}))^2} \tag{2.81}$$

Assuming that the matrix is diagonalizable and that the eigenvectors are orthonormal,[11] if τ_+ and τ_- are the eigenvalues of $\boldsymbol{v}\boldsymbol{u}^H\hat{\boldsymbol{D}} + \hat{\boldsymbol{D}}^H\boldsymbol{u}\boldsymbol{v}^H$ and $\boldsymbol{\omega}_+$ and $\boldsymbol{\omega}_-$ are the corresponding eigenvectors, then

$$\boldsymbol{v}\boldsymbol{u}^H\hat{\boldsymbol{D}} + \hat{\boldsymbol{D}}^H\boldsymbol{u}\boldsymbol{v}^H = \tau_+\boldsymbol{\omega}_+\boldsymbol{\omega}_+^H + \tau_-\boldsymbol{\omega}_-\boldsymbol{\omega}_-^H \tag{2.82}$$

So, the rank-2 updates can be split into 2 rank-1 components that can be used to update the inverse representation for $\rho\mathbf{I} + \hat{\boldsymbol{D}}^H[\hat{k}]\hat{\boldsymbol{D}}[\hat{k}]$. Therefore, the update from equation 2.69

$$\rho\mathbf{I} + (\hat{\boldsymbol{D}}^{(n+1)})^H[\hat{k}]\hat{\boldsymbol{D}}^{(n+1)}[\hat{k}] = \rho\mathbf{I} + (\boldsymbol{D}^{(n)}[\hat{k}] + \hat{\boldsymbol{U}}\hat{\boldsymbol{V}}^H[\hat{k}])^H(\hat{\boldsymbol{D}}^{(n)}[\hat{k}] + \hat{\boldsymbol{U}}\hat{\boldsymbol{V}}^H[\hat{k}]) \tag{2.83}$$

[11] This is not guaranteed. See appendix B for details.

consists of $3\mathcal{R}$ rank-1 Hermitian updates, and the updates must be computed for each frequency $\hat{k}$. Appendix A explains how to update a Cholesky decomposition with rank-1 Hermitian updates.

Recall once again, the dictionary update under consideration:

$$\hat{\boldsymbol{D}}^{(n+1)}[\hat{k}] = \hat{\boldsymbol{D}}^{(n)}[\hat{k}] + \hat{\boldsymbol{U}}\hat{\boldsymbol{V}}^H[\hat{k}] \tag{2.84}$$

This update must be of rank $\mathcal{R}$ at every frequency. Furthermore, the dictionary filter is spatially or temporally limited to its filter size. This second constraint is met if $\hat{\boldsymbol{V}}^H[\hat{k}]$ is similarly spatially limited.

2.8.2 Computational Cost

Given an updated dictionary $\boldsymbol{G}^{(n+1)}$ from some standard dictionary learning update, directly computing the Cholesky decomposition of $\rho\mathbf{I} + \left(\hat{\boldsymbol{G}}^{(n+1)}\right)^H \hat{\boldsymbol{G}}^{(n+1)}$ is of computational cost $\mathcal{O}(\hat{K}M^3)$.[12] The novel approach for the inverse update presented in the previous section replaces the direct computation with low-rank updates to the Cholesky decomposition. This involves two steps:

1. Approximate a standard dictionary update with a low-rank approximation: $\hat{\boldsymbol{U}}\hat{\boldsymbol{V}}[\hat{k}]^H \approx \hat{\boldsymbol{G}}^{(n+1)}[\hat{k}] - \boldsymbol{R}\hat{\boldsymbol{D}}^{(n)}[\hat{k}]$

2. Use the low-rank approximation to update the Cholesky decomposition of $\rho\mathbf{I} + (\boldsymbol{R}\hat{\boldsymbol{D}})^H\boldsymbol{R}\hat{\boldsymbol{D}}$.

Since $\hat{\boldsymbol{U}}$ does not vary with frequency, the first step can be computed in spatial domain. Just as in the frequency domain, the bracket indexing indicates $\boldsymbol{G}[k]$ is a $C \times M$ matrix,

[12] With the use of the Woodbury matrix lemma, this can be instead computed in $\mathcal{O}(\hat{K}C^3)$.

corresponding to k^{th} element for each filter and channel.

$$\boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V}_{\text{normalized}} = \text{TruncSVD}\left(\begin{bmatrix} \boldsymbol{G}^{(n+1)}[0] - \boldsymbol{R}\boldsymbol{D}^{(n)}[0] & \cdots & \boldsymbol{G}^{(n+1)}[K] - \boldsymbol{R}\boldsymbol{D}^{(n)}[K] \end{bmatrix}\right) \tag{2.85}$$

$$\boldsymbol{V} = \boldsymbol{V}_{\text{normalized}}\boldsymbol{\Sigma} \tag{2.86}$$

In many applications, the filter size K will be much smaller than the signal size $\hat{K}$. The computational cost of this operation is $\mathcal{O}(C^2KM) + \mathcal{O}(CM\hat{K}\log(\hat{K}))$. Estimating the truncated SVD computation using randomized SVD reduces the computational cost even further $\mathcal{O}(\text{Я}CKM) + \mathcal{O}(CM\hat{K}\log(\hat{K}))$.

The second step applies 3Я rank-1 Hermitian updates to the Cholesky decomposition. Each update can be computed in $\mathcal{O}(\hat{K}M^2)$ time.[13]

So, the dictionary update computational cost for this novel approach requires $\mathcal{O}(\text{Я}CKM)+$ $\mathcal{O}(CM\hat{K}\log(\hat{K})) + \mathcal{O}(\text{Я}\hat{K}M^2)$.[14]

Using filter size $K = 5$ and signal size $\hat{K} = 64$, I compared the update times of the full Cholesky decomposition from scratch to the Cholesky update described in the previous section. For this comparison, I used a rank-1 approximation $\text{Я} = 1$ and kept the number of filters equal to the number of channels $M = C$. Computations were computed on a CPU with an Intel i7-4710HQ CPU @ 2.50GHz processor with 16GB of RAM. The results are shown in Figure 2.1. For a large number of channels and filters, the computation time for directly computing the Cholesky decomposition from scratch exceeds the computation time for updating a previous decomposition.

[13]Similar to in the direct computation case, the Woodbury matrix lemma can be used to replace the M^2 with C^2 in the computational complexity. However, it is important to account for the fact that $\hat{\boldsymbol{V}}$ is not an orthogonal matrix, since it loses its orthogonality in the Fourier transform. (A unitary transform applied to the columns of a matrix whose rows are orthogonal does not necessarily preserve that orthogonality.) Fortunately, $\hat{\boldsymbol{V}}^H[\hat{k}]\hat{\boldsymbol{V}}[\hat{k}]$ can be diagonalized in $\mathcal{O}(M\text{Я}^2)$ time using the eigendecomposition, which can subsequently be used to diagonalize $\hat{\boldsymbol{U}}\hat{\boldsymbol{V}}^H\hat{\boldsymbol{V}}\hat{\boldsymbol{U}}$ in $\mathcal{O}(C\text{Я}^2$. Since this must be done for each frequency, these diagonalizations would require $\mathcal{O}(M\text{Я}^2\hat{K}) + \mathcal{O}(C\text{Я}^2\hat{K})$.

[14]If C is smaller than M, this computational complexity can be reduced using the Woodbury matrix lemma.

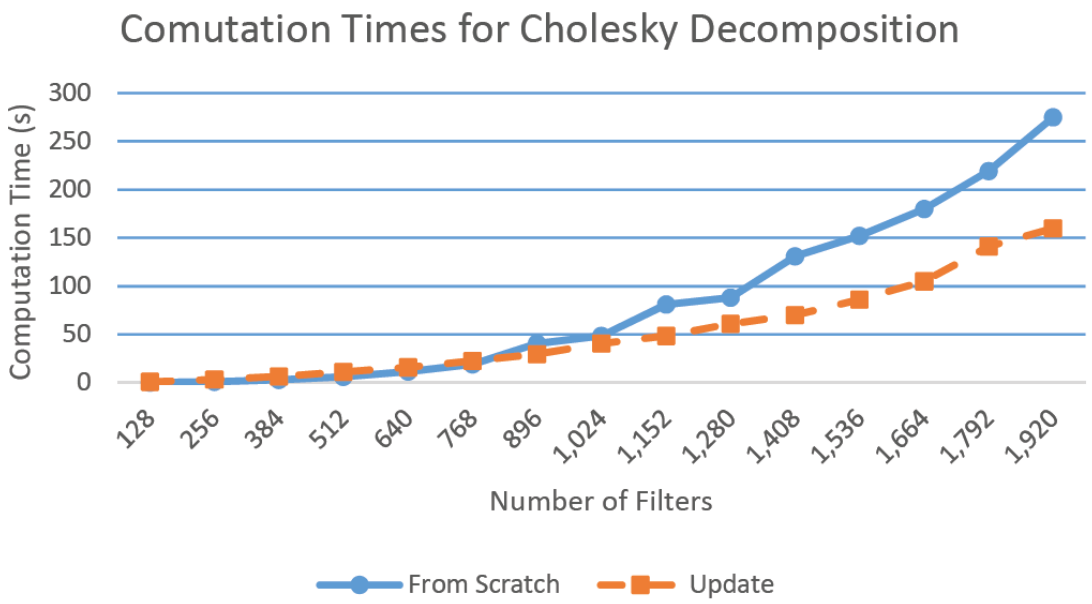


Figure 2.1: This plot shows a comparison of computation times for computing the Cholesky decompostion directly versus updating it instead. In all runs, the number of channels is the same as the number of filters.

2.8.3 Handling Dictionary Normalization

Consider the optimization problem:

$$\begin{aligned} &\min_{\boldsymbol{x},\boldsymbol{y}} \frac{1}{2}\|\boldsymbol{s} - \boldsymbol{D}\boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{y}\|_1 \\ &\text{subject to } \boldsymbol{R}^{-1}\boldsymbol{y} - \boldsymbol{R}^{-1}\boldsymbol{x} = 0 \end{aligned} \tag{2.87}$$

where $\boldsymbol{R}$ is a diagonal matrix with scaled identity blocks:

$$\boldsymbol{R} = \begin{bmatrix} r[1]\mathbf{I} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & r[2]\mathbf{I} & & \vdots \\ \vdots & & \ddots & \\ \mathbf{0} & \dots & & r[M]\mathbf{I} \end{bmatrix} \tag{2.88}$$

and $\boldsymbol{D}$ has normalized dictionary filters.

This optimization problem has the augmented Lagrangian function:

$$\mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}) = \frac{1}{2}\|\boldsymbol{s} - \boldsymbol{D}\boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{y}\|_1 + \boldsymbol{u}^H\boldsymbol{R}^{-1}(\boldsymbol{y} - \boldsymbol{x}) + \frac{\rho}{2}\|\boldsymbol{R}^{-1}(\boldsymbol{y} - \boldsymbol{x})\|_2^2 \quad (2.89)$$

$$\nabla_{\boldsymbol{x}} \mathrm{L}_\rho(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}) = -\boldsymbol{R}^{-1}\boldsymbol{u} - \boldsymbol{D}^H\boldsymbol{s} + \boldsymbol{D}^T\boldsymbol{D}\boldsymbol{x} + \rho\boldsymbol{R}^{-2}\boldsymbol{x} - \rho\boldsymbol{R}^{-2}\boldsymbol{y} \quad (2.90)$$

For $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}$ such that $\nabla_{\boldsymbol{x}}\mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}) = 0$:

$$(\rho\boldsymbol{R}^{-2} + \boldsymbol{D}^T\boldsymbol{D})\boldsymbol{x} = \rho\boldsymbol{R}^{-2}\boldsymbol{y} + \boldsymbol{R}^{-1}\boldsymbol{u} + \boldsymbol{D}^T\boldsymbol{s} \quad (2.91)$$

$$\boldsymbol{R}^{-1}\left(\rho\mathbf{I} + (\boldsymbol{D}\boldsymbol{R})^T(\boldsymbol{D}\boldsymbol{R})\right)\boldsymbol{R}^{-1}\boldsymbol{x} = \rho\boldsymbol{R}^{-2}\boldsymbol{y} + \boldsymbol{R}^{-1}\boldsymbol{u} + \boldsymbol{D}^T\boldsymbol{s} \quad (2.92)$$

$$\left(\rho\mathbf{I} + (\boldsymbol{D}\boldsymbol{R})^T(\boldsymbol{D}\boldsymbol{R})\right)\boldsymbol{R}^{-1}\boldsymbol{x} = \rho\boldsymbol{R}^{-1}\boldsymbol{y} + \boldsymbol{u} + (\boldsymbol{D}\boldsymbol{R})^T\boldsymbol{s} \quad (2.93)$$

$$\boldsymbol{R}^{-1}\boldsymbol{x} = \left(\rho\mathbf{I} + (\boldsymbol{D}\boldsymbol{R})^H(\boldsymbol{D}\boldsymbol{R})\right)^{-1}\left(\rho\boldsymbol{R}^{-1}\boldsymbol{y} + \boldsymbol{u} + (\boldsymbol{D}\boldsymbol{R})^T\boldsymbol{s}\right) \quad (2.94)$$

So,

$$\arg\min_{\boldsymbol{x}} \mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}) = \boldsymbol{R}\left(\rho\mathbf{I} + (\boldsymbol{D}\boldsymbol{R})^T(\boldsymbol{D}\boldsymbol{R})\right)^{-1}\left(\rho\boldsymbol{R}^{-1}\boldsymbol{y} + \boldsymbol{u} + (\boldsymbol{D}\boldsymbol{R})^T\boldsymbol{s}\right) \quad (2.95)$$

However, taking a similar approach to that of scaled ADMM, it will be simpler to track $\boldsymbol{s}_{\text{sc}} = \boldsymbol{R}^{-1}\boldsymbol{x}$ instead of $\boldsymbol{x}$ directly.

$$\boldsymbol{R}^{-1}\boldsymbol{x}^{(t+1)} = \left(\rho\mathbf{I} + (\boldsymbol{D}\boldsymbol{R})^T(\boldsymbol{D}\boldsymbol{R})\right)^{-1}\left((\boldsymbol{D}\boldsymbol{R})^T\boldsymbol{s} + \rho\left(\boldsymbol{R}^{-1}\boldsymbol{y}^{(t)} + \frac{\boldsymbol{u}^{(t)}}{\rho}\right)\right) \quad (2.96)$$

Moving on to the $\boldsymbol{y}$ update,

$$\arg\min_{\boldsymbol{y}} \mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}) = \mathrm{S}_{\frac{\lambda \boldsymbol{R}^2}{\rho}} \left(\boldsymbol{x} - \frac{\boldsymbol{R}\boldsymbol{u}}{\rho} \right) \tag{2.97}$$

Like with $\boldsymbol{x}$, it is simpler to track $\boldsymbol{y}_{\text{sc}} = \boldsymbol{R}^{-1}\boldsymbol{y}$ than $\boldsymbol{y}$ itself.

$$\boldsymbol{R}^{-1}\boldsymbol{y}^{(t+1)} = \mathrm{S}_{\frac{\lambda \boldsymbol{R}}{\rho}} \left(\boldsymbol{R}^{-1}\boldsymbol{x}^{(t+1)} - \frac{\boldsymbol{u}^{\left(t+\frac{1}{2}\right)}}{\rho} \right) \tag{2.98}$$

Finally, tracking $\boldsymbol{u}_{\text{sc}} = \frac{\boldsymbol{u}}{\rho}$ instead of $\boldsymbol{u}$, the dual updates are

$$\frac{\boldsymbol{u}^{\left(t+\frac{1}{2}\right)}}{\rho} = \frac{\boldsymbol{u}^{(t)}}{\rho} + (\alpha - 1)(\boldsymbol{R}^{-1}\boldsymbol{y}^{(t)} - \boldsymbol{R}^{-1}\boldsymbol{x}^{(t+1)}) \tag{2.99}$$

$$\frac{\boldsymbol{u}^{(t+1)}}{\rho} = \frac{\boldsymbol{u}^{\left(t+\frac{1}{2}\right)}}{\rho} + \boldsymbol{R}^{-1}\boldsymbol{y}^{(t+1)} - \boldsymbol{R}^{-1}\boldsymbol{x}^{(t+1)} \tag{2.100}$$

Thus, with this modification to the sparse coding optimization problem, the inverse representation used in the $\boldsymbol{x}$ updates can be updated efficiently (given that the dictionary updates adhere to a particular low-rank structure), and normalization can be handed through a normalization factor $\boldsymbol{R}^{-1}$. The resulting pursuit algorithm is shown in Algorithm 6. The same pursuit algorithm in the larger dictionary learning context is shown in Algorithm 7.

2.9 Conclusion

In this chapter, I have derived a novel sparse coding algorithm for signals with a large number of channels. One of the steps in the iterative algorithm involves solving an inverse problem, but the optimization is constructed such that the representation of the inverse can be updated efficiently $\mathcal{O}(\text{Я}CKM) + \mathcal{O}(CM\hat{K}\log(\hat{K})) + \mathcal{O}(\text{Я}\hat{K}M^2)$ when used within a dictionary-learning algorithm. For certain problems with a large number of channels, this is an improvement to existing methods, whose inverse updates require cubic complexity

Algorithm 6: ADMM for Sparse Coding, Large Number of Channels

$\alpha \in (0, 2]$
$\boldsymbol{D}_{\text{sc}} = \boldsymbol{D}\boldsymbol{R}$
$\boldsymbol{y}_s = \boldsymbol{R}^{-2}\boldsymbol{D}_{\text{sc}}^T\boldsymbol{s}$
$\boldsymbol{u}_s = \mathbf{0}$
while *Not Converged* **do**
  $\boldsymbol{x}_{\text{sc}} = \left(\rho\mathbf{I} + \boldsymbol{D}_{\text{sc}}^T\boldsymbol{D}_{\text{sc}}\right)^{-1}\left(\boldsymbol{D}_{\text{sc}}^T\boldsymbol{s} + \rho\left(\boldsymbol{y}_{\text{sc}} + \boldsymbol{u}_{\text{sc}}\right)\right)$
  $\boldsymbol{u}_{\text{sc}} = \boldsymbol{u}_{\text{sc}} + (\alpha - 1)(\boldsymbol{y}_{\text{sc}} - \boldsymbol{x}_{\text{sc}})$
  $\boldsymbol{y}_{\text{sc}} = \mathrm{S}_{\frac{\lambda \boldsymbol{R}}{\rho}}(\boldsymbol{x}_{\text{sc}} - \boldsymbol{u}_{\text{sc}})$
  $\boldsymbol{u}_{\text{sc}} = \boldsymbol{u}_{\text{sc}} + \boldsymbol{y}_{\text{sc}} - \boldsymbol{x}_{\text{sc}}$
end
$\boldsymbol{x} = \boldsymbol{R}\boldsymbol{x}_{\text{sc}}$
$\boldsymbol{y} = \boldsymbol{R}\boldsymbol{y}_{\text{sc}}$

$\mathcal{O}(\hat{K}M^3)$.

Algorithm 7: Dictionary Learning, Using ADMM with Large Number of Channels

```
// Initialization:
α ∈ (0, 2]
DR = D_init
for m ∈ {1, ... M} do
    R_{m,m} = ‖(DR)_m‖_F
end
(ρI + RD^T DR) = ComputeDecomp (ρ, DR)
i = 0
while Stopping Criteria Not Met do
    s = GetData(i)
    i = i + 1
    // Pursuit:
    D_sc = DR
    y_sc = R^{-2} D_sc^T s
    u_sc = 0
    while Not Converged do
        x_sc = (ρI + RD^T DR)^{-1} (D_sc^T s + ρ (y_sc + u_sc))
        u_sc = u_sc + (α − 1)(y_sc − x_sc)
        y_sc = S_{λR/ρ}(x_sc − u_sc)
        u_sc = u_sc + y_sc − x_sc
    end
    x = R x_sc
    y = R y_sc
    // Updating the Dictionary:
    D̃ = StandardDictionaryUpdate(D, S, X)
    D̃ = Normalize(D̃)
    ΔD = LowRankApprox(D̃ − DR)
    DR = DR + ΔD
    for m ∈ {1, ... M} do
        R_{m,m} = ‖(DR)_m‖_F
    end
    (ρI + RD^T DR) = UpdateDecomp ((ρI + RD^T DR), ρ, D_sc, ΔD)
end
```

CHAPTER 3
LEARNING MULTI-LAYER DICTIONARIES

3.1 Introduction

A multi-layer dictionary model is composed of multiple dictionaries; the model treats the dictionary coefficients of a previous layer as the signal for the subsequent layer. This model dates back to Zeiler's Deconvolutional Neural Networks [10] and can be thought of as a deep autoencoder [27, Chapter 14][28]. Some researchers have interpreted convolutional neural networks as multi-layer dictionary models, the convolution and its corresponding rectified linear units serving as a crude pursuit algorithm [6]. This chapter presents how to apply the novel dictionary learning algorithm from the prior chapter to the multi-layer dictionary learning problem.

3.2 Literature Review

In 2010, Zeiler et al. proposed a multi-layer dictionary model termed a deconvolutional network [10]. The learning process for dictionary filters is entirely unsupervised, and they learn their filters layer-by-layer. Their algorithm is greedy in the sense that there is no feedback from subsequent layers to influence the learning process on the previous layer. This approach was tested both on the task of removing added Gaussian noise to images, and also as a feature extraction method for object recognition on the Caltech-101 dataset [29]. While this research drew a lot of attention at the time, as the success of alternative models like convolutional neural networks grew [30], the popularity of deconvolutional networks decreased.

Multi-layer dictionaries also appear in Bayesian models, going by names such as hierarchical convolutional factor analysis [31][9] and deep deconvolutional learning [32]. These

networks use probabilistic models to prune network architecture and provide interpretable dictionaries. Inference can be slow.

In more recent work, [12] and [13] use ADMM for pursuit on a multi-layer dictionary model. Their pursuit algorithm attempts to solve the minimization problem:

$$\underset{\boldsymbol{x}}{\text{minimize}} = \sum_{\ell=1}^{L} \frac{\mu_\ell}{2} \|\boldsymbol{x}_{\ell-1} - \boldsymbol{D}_\ell \boldsymbol{x}_\ell\|_2^2 + \lambda_\ell \|\boldsymbol{x}_\ell\|_1 \tag{3.1}$$

where $\boldsymbol{x}_0 = \boldsymbol{s}$ is the signal. They convert this to a constrained optimization for the ADMM algorithm.

$$\begin{aligned} &\min_{\boldsymbol{x},\boldsymbol{z}} \sum_{\ell=1}^{L} \frac{\mu_\ell}{2} \|\boldsymbol{z}_{\ell-1} - \boldsymbol{D}_\ell \boldsymbol{x}_\ell\|_2^2 + \lambda_\ell \|\boldsymbol{z}_\ell\|_1 \\ &\text{subject to } \boldsymbol{z}_\ell - \boldsymbol{x}_\ell = 0 \end{aligned} \tag{3.2}$$

where $\boldsymbol{z}_0 = \boldsymbol{s}$ is the signal. The $\boldsymbol{x}$ updates involve solving an inverse problem. They use a tight-frame assumption to approximate the inverse.

Finally, in [8], Chodosh and Lucey use a similar model to [12] and [13], but replace the ADMM approach with FISTA-like linear-proximal iterative steps.

3.3 Multi-Layer ADMM with Low-Rank Updates

This chapter demonstrates how to apply the novel sparse coding method for multi-channel signals to a multi-layer dictionary pursuit problem.

To start off, it is helpful to write a multi-layer dictionary optimization problem. To keep things compact, let $\boldsymbol{x}_0 = \boldsymbol{s}$. I use a similar multi-layer dictionary model to the one used in [12] and [13]:

$$\underset{\boldsymbol{x}}{\text{minimize}} \sum_{\ell=1}^{L} \frac{\mu_\ell}{2} \|\boldsymbol{x}_{\ell-1} - \boldsymbol{D}_\ell \boldsymbol{x}_\ell\|_2^2 + \lambda_\ell \|\boldsymbol{x}_\ell\|_1 \tag{3.3}$$

Applying the ADMM algorithm, I add a secondary primal variable $\boldsymbol{z}_\ell$ for each layer ℓ and constrain it to be equal to $\boldsymbol{x}_\ell$. As in the previous chapter, I scale this constraint so that the inverse representation can be updated efficiently without losing the normalized quality

of the dictionary. This replaces the tight-frame assumption used in [12] and [13]. Again, keeping things compact, let $\boldsymbol{z}_0 = \boldsymbol{s}$.

$$\begin{aligned} &\min_{\boldsymbol{x},\boldsymbol{z}} \sum_{\ell=1}^{L} \frac{\mu_\ell}{2} \|\boldsymbol{z}_{\ell-1} - \boldsymbol{D}_\ell \boldsymbol{x}_\ell\|_2^2 + \lambda_\ell \|\boldsymbol{z}_\ell\|_1 \\ &\text{subject to } \sqrt{\mu_\ell}\boldsymbol{R}_\ell^{-1}\boldsymbol{z}_\ell - \sqrt{\mu_\ell}\boldsymbol{R}_\ell^{-1}\boldsymbol{x}_\ell = 0 \end{aligned} \tag{3.4}$$

where $\boldsymbol{z}_0 = \boldsymbol{s}$ is not a primal variable, but instead the signal itself.

This optimization problem has the augmented Lagrangian function:

$$\mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\gamma}) = f(\boldsymbol{x}, \boldsymbol{z}) + \sum_{\ell=1}^{L} \frac{\rho}{2} \|\sqrt{\mu_\ell}\boldsymbol{R}_\ell^{-1}(\boldsymbol{z}_\ell - \boldsymbol{x}_\ell) + \frac{\boldsymbol{\gamma}_\ell}{\rho}\|_2^2 - \frac{\rho}{2}\|\frac{\boldsymbol{\gamma}_\ell}{\rho}\|_2^2 \tag{3.5}$$

where

$$f(\boldsymbol{x}, \boldsymbol{z}) = \sum_{\ell=1}^{L} \frac{\mu_\ell}{2} \|\boldsymbol{z}_{\ell-1} - \boldsymbol{D}_\ell \boldsymbol{x}_\ell\|_2^2 + \lambda_\ell \|\boldsymbol{z}_\ell\|_1 \tag{3.6}$$

Recall that the ADMM algorithm (with relaxation) iteratively alternates between primal and dual updates, using four update equations:

$$\boldsymbol{x}^{(t+1)} = \arg\min_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{z}^{(t)}, \boldsymbol{\gamma}^{(t)}) \tag{3.7}$$

$$\frac{\boldsymbol{\gamma}^{\left(t+\frac{1}{2}\right)}}{\rho} = \frac{\boldsymbol{\gamma}^{(t)}}{\rho} + (\alpha - 1)(\boldsymbol{A}\boldsymbol{x}^{(t+1)} + \boldsymbol{B}\boldsymbol{z}^{(t)} + \boldsymbol{c}) \tag{3.8}$$

$$\boldsymbol{z}^{(t+1)} = \arg\min_{\boldsymbol{z}} \mathcal{L}\left(\boldsymbol{x}^{(t+1)}, \boldsymbol{z}, \boldsymbol{\gamma}^{\left(t+\frac{1}{2}\right)}\right) \tag{3.9}$$

$$\frac{\boldsymbol{\gamma}^{(t+1)}}{\rho} = \frac{\boldsymbol{\gamma}^{\left(t+\frac{1}{2}\right)}}{\rho} + \boldsymbol{A}\boldsymbol{x}^{(t+1)} + \boldsymbol{B}\boldsymbol{z}^{(t+1)} + \boldsymbol{c} \tag{3.10}$$

where $\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} + \boldsymbol{c} = \boldsymbol{0}$ are the affine constraints.

The first of these updates is the $\boldsymbol{x}$ update, which updates the coefficients.

3.3.1 Coefficients Update Equation

The coefficients update comes from equation 3.7, which can be derived through setting the gradient of the Lagrangian equal to zero and solving for $\boldsymbol{x}$.

$$\nabla_{\boldsymbol{x}_\ell} f(\boldsymbol{x}, \boldsymbol{z}) = \mu_\ell \boldsymbol{D}_\ell^T \boldsymbol{D}_\ell \boldsymbol{x}_\ell - \mu_\ell \boldsymbol{D}_\ell^T \boldsymbol{z}_{\ell-1} \tag{3.11}$$

$$\nabla_{\boldsymbol{x}_\ell} \frac{1}{2} \| \sqrt{\mu_\ell} \boldsymbol{R}_\ell^{-1} (\boldsymbol{z}_\ell - \boldsymbol{x}_\ell) + \frac{\boldsymbol{\gamma}_\ell}{\rho} \|_2^2 = \mu_\ell \boldsymbol{R}_\ell^{-2} \boldsymbol{x} - \mu_\ell \boldsymbol{R}_\ell^{-2} \boldsymbol{z}_\ell - \frac{\sqrt{\mu_\ell} \boldsymbol{R}_\ell^{-1} \boldsymbol{\gamma}_\ell}{\rho} \tag{3.12}$$

Therefore,

$$\nabla_{\boldsymbol{x}_\ell} \mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\gamma}) = \mu_\ell \boldsymbol{D}_\ell^T \boldsymbol{D}_\ell \boldsymbol{x}_\ell - \mu_\ell \boldsymbol{D}_\ell^T \boldsymbol{z}_{\ell-1} + \rho \left(\mu_\ell \boldsymbol{R}_\ell^{-2} \boldsymbol{x} - \mu_\ell \boldsymbol{R}_\ell^{-2} \boldsymbol{z}_\ell - \frac{\sqrt{\mu_\ell} \boldsymbol{R}_\ell^{-1} \boldsymbol{\gamma}_\ell}{\rho} \right) \tag{3.13}$$

For $\boldsymbol{x}$, $\boldsymbol{z}$, $\boldsymbol{\gamma}$, such that $\nabla_{\boldsymbol{x}_\ell} \mathcal{L}_\rho(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_L, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_L, \boldsymbol{\gamma}_1, \ldots, \boldsymbol{\gamma}_L) = 0$:

$$\mu_\ell (\rho \boldsymbol{R}_\ell^{-2} + \boldsymbol{D}_\ell^T \boldsymbol{D}_\ell) \boldsymbol{x}_\ell = \mu_\ell \boldsymbol{D}_\ell^T \boldsymbol{z}_{\ell-1} + \rho \mu_\ell \boldsymbol{R}_\ell^{-2} \boldsymbol{z}_\ell + \sqrt{\mu_\ell} \boldsymbol{R}_\ell^{-1} \boldsymbol{\gamma}_\ell \tag{3.14}$$

$$(\rho \boldsymbol{R}_\ell^{-2} + \boldsymbol{D}_\ell^T \boldsymbol{D}_\ell) \boldsymbol{x}_\ell = \boldsymbol{D}_\ell^T \boldsymbol{z}_{\ell-1} + \rho \boldsymbol{R}_\ell^{-2} \boldsymbol{z}_\ell + \frac{\boldsymbol{R}_\ell^{-1} \boldsymbol{\gamma}_\ell}{\sqrt{\mu_\ell}} \tag{3.15}$$

$$\boldsymbol{x}_\ell = (\rho \boldsymbol{R}_\ell^{-2} + \boldsymbol{D}_\ell^T \boldsymbol{D}_\ell)^{-1} \left(\boldsymbol{D}_\ell^T \boldsymbol{z}_{\ell-1} + \rho \boldsymbol{R}_\ell^{-2} \boldsymbol{z}_\ell + \frac{\boldsymbol{R}_\ell^{-1} \boldsymbol{\gamma}_\ell}{\sqrt{\mu_\ell}} \right) \tag{3.16}$$

This solution is the $\boldsymbol{x}$ update for the ADMM algorithm, but a couple extra steps can put it into a form that is easier to use.

$$\boldsymbol{x}_\ell = \boldsymbol{R}_\ell \left(\rho \mathbf{I} + (\boldsymbol{D}_\ell \boldsymbol{R}_\ell)^T \boldsymbol{D}_\ell \boldsymbol{R}_\ell \right)^{-1} \boldsymbol{R}_\ell \left(\boldsymbol{D}_\ell^T \boldsymbol{z}_{\ell-1} + \rho \boldsymbol{R}_\ell^{-2} \boldsymbol{z}_\ell + \frac{\boldsymbol{R}_\ell^{-1} \boldsymbol{\gamma}_\ell}{\sqrt{\mu_\ell}} \right) \tag{3.17}$$

$$\boldsymbol{R}_\ell^{-1}\boldsymbol{x}_\ell = \left(\rho\mathbf{I} + (\boldsymbol{D}_\ell\boldsymbol{R}_\ell)^T\boldsymbol{D}_\ell\boldsymbol{R}_\ell\right)^{-1}\left((\boldsymbol{D}_\ell\boldsymbol{R}_\ell)^T\boldsymbol{z}_{\ell-1} + \rho\boldsymbol{R}_\ell^{-1}\boldsymbol{z}_\ell + \frac{\boldsymbol{\gamma}_\ell}{\sqrt{\mu_\ell}}\right) \tag{3.18}$$

So, therefore the update equation for $\boldsymbol{R}_\ell^{-1}\boldsymbol{x}_\ell$ is the following:

$$\boldsymbol{R}_\ell^{-1}\boldsymbol{x}_\ell^{(t+1)} = \left(\rho\mathbf{I} + (\boldsymbol{D}_\ell\boldsymbol{R}_\ell)^T\boldsymbol{D}_\ell\boldsymbol{R}_\ell\right)^{-1}\left((\boldsymbol{D}_\ell\boldsymbol{R}_\ell)^T\boldsymbol{z}_{\ell-1}^{(t)} + \rho\left(\boldsymbol{R}_\ell^{-1}\boldsymbol{z}_\ell^{(t)} + \frac{\boldsymbol{\gamma}_\ell^{(t)}}{\rho\sqrt{\mu_\ell}}\right)\right) \tag{3.19}$$

Before moving onto another update equation, there are a few useful things to note here. The form of the inverse matrix identically matches the form from the last chapter, so the inverse representation can be updated efficiently if the updates have a low-rank structure. Furthermore, $\boldsymbol{D}_\ell\boldsymbol{R}_\ell$ is the unnormalized dictionary that is updated through low-rank updates. The normalized dictionary does not need to be explicitly calculated at all. It would be easy to isolate $\boldsymbol{x}_\ell$, but it will be simpler to keep track of $\boldsymbol{R}_\ell^{-1}\boldsymbol{x}_\ell$ instead, similar to how $\frac{\boldsymbol{u}}{\rho}$ is tracked instead of $\boldsymbol{u}$ in the scaled ADMM algorithm.

3.3.2 Proximal Updates

The second set of primal updates comes from equation 3.9, repeated here for convenience:

$$\boldsymbol{z}^{(t+1)} = \arg\min_{\boldsymbol{z}} \mathcal{L}\left(\boldsymbol{x}^{(t+1)}, \boldsymbol{z}, \boldsymbol{\gamma}^{\left(t+\frac{1}{2}\right)}\right) \tag{3.20}$$

where, as before,

$$\mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\gamma}) = f(\boldsymbol{x}, \boldsymbol{z}) + \sum_{\ell=1}^{L}\frac{\rho}{2}\|\sqrt{\mu_\ell}\boldsymbol{R}_\ell^{-1}(\boldsymbol{z}_\ell - \boldsymbol{x}_\ell) + \frac{\boldsymbol{\gamma}_\ell}{\rho}\|_2^2 - \frac{1}{2\rho}\|\boldsymbol{\gamma}_\ell\|_2^2 \tag{3.21}$$

and

$$f(\boldsymbol{x}, \boldsymbol{z}) = \sum_{\ell=1}^{L}\frac{\mu_\ell}{2}\|\boldsymbol{z}_{\ell-1} - \boldsymbol{D}_\ell\boldsymbol{x}_\ell\|_2^2 + \lambda_\ell\|\boldsymbol{z}_\ell\|_1 \tag{3.22}$$

For $\boldsymbol{x}$, $\boldsymbol{z}$, $\boldsymbol{\gamma}$, such that $\nabla_{\boldsymbol{z}_\ell}\mathcal{L}_\rho(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_L,\boldsymbol{z}_0,\ldots,\boldsymbol{z}_L,\boldsymbol{\gamma}_0,\ldots,\boldsymbol{\gamma}_L)=0$:

$$\nabla_{\boldsymbol{z}}\frac{\mu_{\ell+1}}{2}\|\boldsymbol{z}_\ell-\boldsymbol{D}_{\ell+1}\boldsymbol{x}_{\ell+1}\|_2^2+\lambda_\ell\|\boldsymbol{z}_\ell\|_1+\frac{\rho}{2}\|\sqrt{\mu_\ell}\boldsymbol{R}_\ell^{-1}(\boldsymbol{z}_\ell-\boldsymbol{x}_\ell)+\frac{\boldsymbol{\gamma}_\ell}{\rho}\|_2^2=0 \tag{3.23}$$

Something important to note here is that each element of $\boldsymbol{z}_\ell$ can be treated independently, that is:

$$\nabla_{\boldsymbol{z}_\ell[i]}\frac{\mu_{\ell+1}}{2}(\boldsymbol{z}_\ell[i]-(\boldsymbol{D}_{\ell+1}\boldsymbol{x}_{\ell+1})[i])^2+\lambda_\ell|\boldsymbol{z}_\ell[i]|+\frac{\rho}{2}(\frac{\sqrt{\mu_\ell}}{\boldsymbol{R}_\ell[i]}(\boldsymbol{z}_\ell[i]-\boldsymbol{x}_\ell[i])+\frac{\boldsymbol{\gamma}_\ell[i]}{\rho})^2=0 \tag{3.24}$$

where $\boldsymbol{R}_\ell[i]$ is the scalar i^{th} diagonal entry of diagonal matrix $\boldsymbol{R}_\ell$.

$$\nabla_{\boldsymbol{z}_\ell[i]}\frac{\mu_{\ell+1}}{2}(\boldsymbol{z}_\ell[i]-(\boldsymbol{D}_{\ell+1}\boldsymbol{x}_{\ell+1})[i])^2+\lambda_\ell|\boldsymbol{z}_\ell[i]|+\frac{\rho\mu_\ell}{2\boldsymbol{R}_\ell^2[i]}(\boldsymbol{z}_\ell[i]-\boldsymbol{x}_\ell[i]+\frac{\boldsymbol{R}_\ell[i]\boldsymbol{\gamma}_\ell[i]}{\rho\sqrt{\mu_\ell}})^2=0 \tag{3.25}$$

For the sake of brevity, I will now drop the indexing:

$$\nabla_{\boldsymbol{z}_\ell}\frac{\mu_{\ell+1}}{2}(\boldsymbol{z}_\ell^2-2(\boldsymbol{D}_{\ell+1}\boldsymbol{x}_{\ell+1})z_\ell)+\lambda_\ell|\boldsymbol{z}_\ell|+\frac{\rho\mu_\ell}{2\boldsymbol{R}_\ell^2}(\boldsymbol{z}_\ell^2-2\boldsymbol{x}_\ell\boldsymbol{z}_\ell+\frac{2\boldsymbol{R}_\ell\boldsymbol{\gamma}_\ell\boldsymbol{z}_\ell}{\rho\sqrt{\mu_\ell}})=0 \tag{3.26}$$

$$\nabla_{\boldsymbol{z}_\ell}\frac{1}{2}(\mu_{\ell+1}+\rho\mu_\ell\boldsymbol{R}_\ell^{-2})\boldsymbol{z}_\ell^2-\mu_{\ell+1}\boldsymbol{D}_{\ell+1}\boldsymbol{x}_{\ell+1}\boldsymbol{z}_\ell-\rho\mu_\ell\boldsymbol{R}_\ell^{-2}\boldsymbol{x}_\ell\boldsymbol{z}_\ell+\sqrt{\mu_\ell}\boldsymbol{R}_\ell^{-1}\boldsymbol{\gamma}_\ell\boldsymbol{z}_\ell+\lambda_\ell|\boldsymbol{z}_\ell|=0 \tag{3.27}$$

$$\nabla_{\boldsymbol{z}_\ell}\frac{1}{2}\boldsymbol{z}_\ell^2-\frac{\mu_{\ell+1}\boldsymbol{D}_{\ell+1}\boldsymbol{x}_{\ell+1}+\rho\mu_\ell\boldsymbol{R}_\ell^{-2}\boldsymbol{x}_\ell-\sqrt{\mu_\ell}\boldsymbol{R}_\ell^{-1}\boldsymbol{\gamma}_\ell}{\mu_{\ell+1}+\rho\mu_\ell\boldsymbol{R}_\ell^{-2}}\boldsymbol{z}_\ell+\frac{\lambda_\ell}{\mu_{\ell+1}+\rho\mu_\ell\boldsymbol{R}_\ell^{-2}}|\boldsymbol{z}_\ell|=0 \tag{3.28}$$

$$\boldsymbol{z}_\ell=\mathrm{S}_{\frac{\lambda_\ell}{\mu_{\ell+1}+\rho\mu_\ell\boldsymbol{R}_\ell^{-2}}}\left(\frac{\mu_{\ell+1}\boldsymbol{D}_{\ell+1}\boldsymbol{x}_{\ell+1}+\rho\mu_\ell\boldsymbol{R}_\ell^{-2}(\boldsymbol{x}_\ell-\frac{\boldsymbol{R}_\ell\boldsymbol{\gamma}_\ell}{\rho\sqrt{\mu_\ell}})}{\mu_{\ell+1}+\rho\mu_\ell\boldsymbol{R}_\ell^{-2}}\right) \tag{3.29}$$

$$\boldsymbol{z}_\ell = \frac{1}{\mu_{\ell+1} + \rho\mu_\ell \boldsymbol{R}_\ell^{-2}} \mathrm{S}_{\lambda_\ell}\left(\mu_{\ell+1}\boldsymbol{D}_{\ell+1}\boldsymbol{x}_{\ell+1} + \rho\mu_\ell \boldsymbol{R}_\ell^{-1}\left(\boldsymbol{R}_\ell^{-1}\boldsymbol{x}_\ell - \frac{\boldsymbol{\gamma}_\ell}{\rho\sqrt{\mu_\ell}}\right)\right) \tag{3.30}$$

Putting things back in matrix form,[1]

$$\boldsymbol{z}_\ell^{(t+1)} = (\mu_{\ell+1}\mathbf{I} + \rho\mu_\ell \boldsymbol{R}_\ell^{-2})^{-1} \mathrm{S}_{\lambda_\ell}\left(\mu_{\ell+1}\boldsymbol{D}_{\ell+1}\boldsymbol{x}_{\ell+1}^{(t+1)} + \rho\mu_\ell \boldsymbol{R}_\ell^{-1}\left(\boldsymbol{R}_\ell^{-1}\boldsymbol{x}_\ell^{(t+1)} - \frac{\boldsymbol{\gamma}_\ell^{(t+\frac{1}{2})}}{\rho\sqrt{\mu_\ell}}\right)\right) \tag{3.31}$$

Note there is a dependence on $\boldsymbol{R}_{\ell+1}^{-1}\boldsymbol{x}_{\ell+1}$. The last layer will have to be considered separately. Using the same procedure, the update for $\boldsymbol{z}_L$ can be derived. Given how similar the derivations are to those used for the other $\boldsymbol{z}$ layers, I will skip to the result.

$$\boldsymbol{z}_L = \boldsymbol{R}_\ell \mathrm{S}_{\frac{\lambda_L \boldsymbol{R}_L}{\rho\mu_L}}\left(\boldsymbol{R}_L^{-1}\boldsymbol{x}_L - \frac{\boldsymbol{\gamma}_L}{\rho\sqrt{\mu_L}}\right) \tag{3.32}$$

$$\boldsymbol{z}_L^{(t+1)} = \boldsymbol{R}_\ell \mathrm{S}_{\frac{\lambda_L \boldsymbol{R}_L}{\rho\mu_L}}\left(\boldsymbol{R}_L^{-1}\boldsymbol{x}_L^{(t+1)} - \frac{\boldsymbol{\gamma}_L^{(t+\frac{1}{2})}}{\rho\sqrt{\mu_L}}\right) \tag{3.33}$$

3.3.3 Dual Updates

Rather than tracking $\boldsymbol{\gamma}_\ell$ or $\frac{\gamma_\ell}{\rho}$ explicitly, it will be easier to track $\frac{\gamma_\ell}{\rho\sqrt{\mu_\ell}}$. The update equations are very straightforward.

$$\frac{\boldsymbol{\gamma}_\ell^{(t+\frac{1}{2})}}{\rho\sqrt{\mu_\ell}} = \frac{\boldsymbol{\gamma}_\ell^{(t)}}{\rho\sqrt{\mu_\ell}} + (\alpha - 1)(\boldsymbol{R}_\ell^{-1}\boldsymbol{z}_\ell^{(t)} - \boldsymbol{R}^{-1}\boldsymbol{x}_\ell^{(t+1)}) \tag{3.34}$$

$$\frac{\boldsymbol{\gamma}_\ell^{(t+1)}}{\rho\sqrt{\mu_\ell}} = \frac{\boldsymbol{\gamma}_\ell^{(t+\frac{1}{2})}}{\rho\sqrt{\mu_\ell}} + \boldsymbol{R}_\ell^{-1}\boldsymbol{z}_\ell^{(t+1)} - \boldsymbol{R}^{-1}\boldsymbol{x}_\ell^{(t+1)} \tag{3.35}$$

[1] Recall that while indexing was dropped for brevity, equation 3.30 is still implicitly indexed.

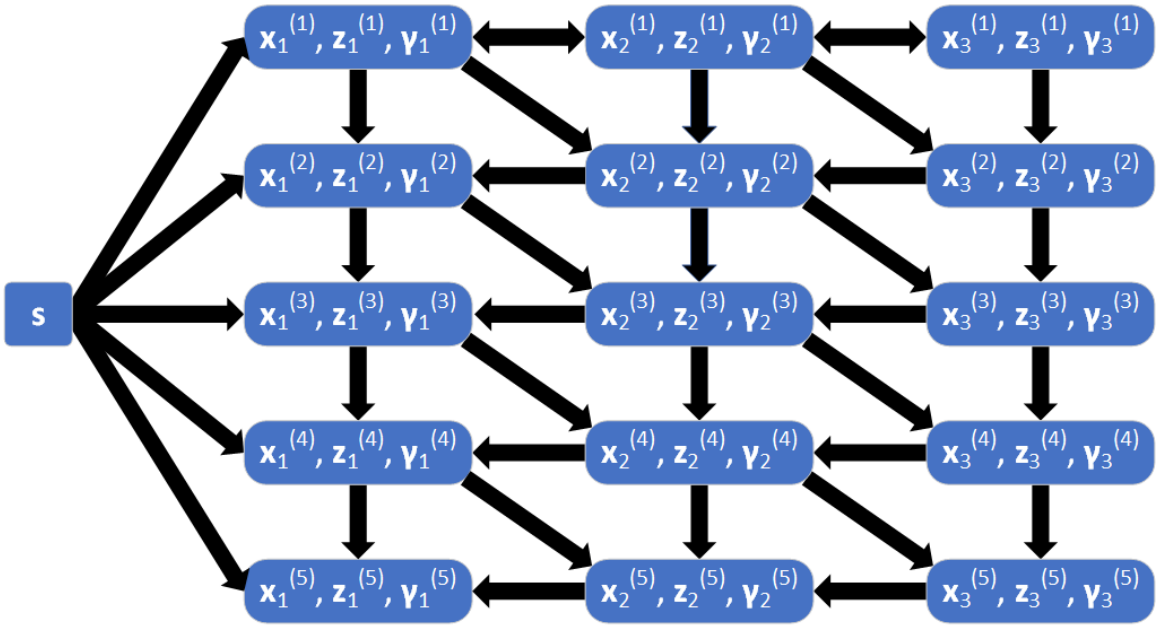


Figure 3.1: This diagram shows interactions between layers across iterations. The double-sided arrows at the top of the diagram go both directions because $\boldsymbol{x}_{\ell+1}$ influences $\boldsymbol{z}_\ell$ during initialization.

3.4 Pursuit Algorithm Summary

Together, the equations from the sections above produce a pursuit algorithm for a multi-layer dictionary model, as shown in Algorithm 8. It is important to note that unlike in [10], there is feedback between layers during pursuit. Figure 3.1 shows the interactions between layers across iterations. This allows for asymptotic convergence to an optimal solution of the objective in equation 3.2.

3.5 Dictionary Learning

There are many possible approaches to compute dictionary updates, but the focus here will be on gradient descent. The pursuit algorithm generates several layers of coefficients from an input signal, and these coefficients can subsequently be used to generate some output, for classification, reconstruction, or some other task. With the appropriate choice of loss function $\mathscr{L}$, backpropagation can be used to update the dictionaries. All of the

computations in pursuit are almost[2] differentiable, so backpropagation is possible all the way back to the signal. Regardless of whether stochastic gradient descent or momentum methods such as Nesterov [33] or ADAM [34] is used, the dictionary updates must be replaced with low-rank substitutes if the number of channels is large, so that the inverse decomposition can be updated efficiently. At least some of the gradients are computed in frequency domain. The convolutional filters are either spatially or temporally bound, and the frequency domain gradients will not respect that constraint, so they will need to be transformed and truncated.

Backpropagating gradients through most of the operations in the pursuit algorithm is very straightforward. Some platforms such as PyTorch [35] or TensorFlow [36] will even do so automatically. However, the $\boldsymbol{x}$-updates rely on a matrix decomposition to solve an inverse problem, and this prevents automatic differentiation in respect to the dictionaries. The equations necessary to compute gradients for the inverse problem in the $\boldsymbol{x}$-update are derived in appendix C. Let $\hat{\boldsymbol{D}}_\ell$ be the frequency domain representation of the unnormalized dictionary corresponding to the ℓth layer, and let $\boldsymbol{Q}_\ell = \rho\mathbf{I} + \hat{\boldsymbol{D}}_\ell^H \hat{\boldsymbol{D}}_\ell$.

$$\nabla_{\hat{\boldsymbol{D}}_\ell}^{(\boldsymbol{b}\to\boldsymbol{Q}_\ell^{-1}\boldsymbol{b})} \mathscr{L} = -(\hat{\boldsymbol{D}}_\ell \boldsymbol{Q}_\ell^{-1}\boldsymbol{b})(\boldsymbol{Q}_\ell^{-1}\nabla_{\boldsymbol{Q}_\ell^{-1}\boldsymbol{b}}\mathscr{L})^H - (\hat{\boldsymbol{D}}_\ell \boldsymbol{Q}_\ell^{-1}\nabla_{\boldsymbol{Q}_\ell^{-1}\boldsymbol{b}}\mathscr{L})(\boldsymbol{Q}_\ell^{-1}\boldsymbol{b})^H \quad (3.36)$$

where $\boldsymbol{b}$ is the input vector to the computational step $\boldsymbol{Q}_\ell^{-1}\boldsymbol{b}$.[3] The superscript of the gradient specifies which gradient term is being computed. This is necessary because same dictionary gets reused across multiple computations, and so these gradient terms must be computed separately and then aggregated to get the actual gradient.

For the Woodbury form[4], the decomposition represents a different matrix for efficiently

[2]Like rectified linear activation functions, there is a discontinuity in the derivative for shrinkage operators. While such functions are technically not differentiable, this does not pose any issue for backpropagation. Operations that rely on a complex conjugate of a complex input also are not differentiable, but since the output is a scalar loss function, gradients can still be computed, which is sufficient for gradient descent. See appendix C for details.

[3]$\boldsymbol{b}$ would take on a value like $\boldsymbol{b} = \mathcal{F}\left((\boldsymbol{D}_\ell \boldsymbol{R}_\ell)^T \boldsymbol{z}_{\ell-1}^{(t)} + \rho\left(\boldsymbol{R}_\ell^{-1}\boldsymbol{z}_\ell^{(t)} + \frac{\boldsymbol{\gamma}_\ell^{(t)}}{\rho\sqrt{\mu_\ell}}\right)\right)$, as seen in the $\boldsymbol{x}$-update equation 3.19.

[4]Useful if the number of filters is larger than the number of channels

solving the inverse problem, and so the equation for the corresponding gradient term is different. Let $\Xi_\ell = \rho \mathbf{I} + \hat{\boldsymbol{D}}_\ell \hat{\boldsymbol{D}}_\ell^H$.

$$\nabla_{\hat{\boldsymbol{D}}_\ell}^{(\boldsymbol{b} \to \Xi_\ell^{-1}\boldsymbol{b})} \mathscr{L} = -(\Xi_\ell^{-1} \nabla_{\boldsymbol{Q}_\ell^{-1}\boldsymbol{b}} \mathscr{L})(\hat{\boldsymbol{D}}_\ell^H \Xi_\ell^{-1} \boldsymbol{b})^H - (\Xi_\ell^{-1}\boldsymbol{b})(\hat{\boldsymbol{D}}_\ell^H \Xi_\ell^{-1} \nabla_{\boldsymbol{Q}_\ell^{-1}\boldsymbol{b}} \mathscr{L})^H \quad (3.37)$$

where $\boldsymbol{b}$ is the input vector to the computational step $\Xi_\ell^{-1}\boldsymbol{b}$.

With these equations it is possible to compute dictionary updates through backpropagation. Putting it all together, a dictionary learning algorithm for multi-layer dictionary models is shown in Algorithm 9.

3.6 Summary

In this chapter, I have applied the novel sparse coding algorithm from the previous chapter to a multi-layer dictionary model. If the dictionaries are updated with low-rank updates, the inverse representation necessary for the $\boldsymbol{x}$ updates in the algorithm can be updated efficiently. This approach offers an alternative to direct proximal methods such as FISTA or mathematically suspect inverse approximations like the tight-frame assumption.

Algorithm 8: ADMM for Multi-Layer Pursuit with Unnormalized Dictionary

input : Over-relaxation parameter: $\alpha \in (0, 2]$, Number of layers: L, Number of filters: M, Signal size: N, Signal: $\boldsymbol{s}$, Unnormalized dictionary for each layer: $\boldsymbol{D}_\ell$, Objective function term coefficients: μ_ℓ and λ_ℓ, Efficient means to compute $(\rho\mathbf{I} + \boldsymbol{D}_\ell^T\boldsymbol{D}_\ell)^{-1}\boldsymbol{b}$

output: Sparse coding coefficients: either $\boldsymbol{x}_\ell$ or $\boldsymbol{z}_\ell$

for $\ell \in \{1, \ldots, L\}$ **do**
  $\boldsymbol{\gamma}_\ell = \mathbf{0}$ **for** $m \in \{1, \ldots, M\}$ **do**
    $r_\ell[m] = \|\boldsymbol{D}_\ell[:, mN]\|_2$
  end
end
$\boldsymbol{x}_1 = \boldsymbol{R}_1^{-2}\boldsymbol{D}_1^T$
for $\ell \in \{2, \ldots, L\}$ **do**
  $\boldsymbol{x}_\ell = \boldsymbol{R}_\ell^{-2}\boldsymbol{D}_\ell^T\boldsymbol{R}_{\ell-1}\boldsymbol{x}_{\ell-1}$
end
while *Not Converged* **do**
  for $\ell \in \{1, \ldots, L-1\}$ **do**
    $\boldsymbol{z}_\ell = (\rho\mu_\ell\mathbf{I} + \mu_{\ell+1}\boldsymbol{R}_\ell^2)^{-1}\boldsymbol{R}_\ell^2\,\mathrm{S}_{\lambda_\ell}\left(\mu_{\ell+1}\boldsymbol{D}_{\ell+1}\boldsymbol{x}_{\ell+1} + \rho\mu_\ell\boldsymbol{R}_\ell^{-1}(\boldsymbol{x}_\ell - \boldsymbol{\gamma}_\ell)\right)$
  end
  $\boldsymbol{z}_L = \boldsymbol{R}_L\,\mathrm{S}_{\frac{\lambda_L\boldsymbol{R}_L}{\rho\mu_L}}(\boldsymbol{x}_L - \boldsymbol{\gamma}_L)$
  for $\ell \in \{1, \ldots, L\}$ **do**
    $\boldsymbol{\gamma}_\ell = \boldsymbol{\gamma}_\ell + \boldsymbol{R}_\ell^{-1}\boldsymbol{z}_\ell - \boldsymbol{x}_\ell$
  end
  $\boldsymbol{x}_1 = (\rho\mathbf{I} + \boldsymbol{D}_1^T\boldsymbol{D}_1)^{-1}(\boldsymbol{D}_1^T\boldsymbol{s} + \rho(\boldsymbol{R}_1^{-1}\boldsymbol{z}_1 + \boldsymbol{\gamma}_1))$
  for $\ell \in \{2, \ldots, L\}$ **do**
    $\boldsymbol{x}_\ell = (\rho\mathbf{I} + \boldsymbol{D}_\ell^T\boldsymbol{D}_\ell)^{-1}(\boldsymbol{D}_\ell^T\boldsymbol{z}_{\ell-1} + \rho(\boldsymbol{R}_\ell^{-1}\boldsymbol{z}_\ell + \boldsymbol{\gamma}_\ell))$
  end
  for $\ell \in \{1, \ldots, L\}$ **do**
    $\boldsymbol{\gamma}_\ell = \boldsymbol{\gamma}_\ell + (\alpha - 1)(\boldsymbol{R}_\ell^{-1}\boldsymbol{z}_\ell - \boldsymbol{x}_\ell)$
  end
end
for $\ell \in \{1, \ldots, L\}$ **do**
  $\boldsymbol{x}_\ell = \boldsymbol{R}_\ell\boldsymbol{x}_\ell$
end

Algorithm 9: Multi-Layer Dictionary Learning

input : Over-relaxation parameter: $\alpha \in (0, 2]$, Signal: $\boldsymbol{s}$, Unnormalized initial dictionary for each layer: $\boldsymbol{D}_\ell$, Objective function term coefficients for each layer: μ_ℓ and λ_ℓ

output: Unnormalized dictionaries for each layer: $\boldsymbol{D}_\ell$

for $\ell \in \{1, \ldots, L\}$ **do**

  $\left(\rho\mathbf{I} + \boldsymbol{D}_\ell^T \boldsymbol{D}_\ell\right) = \text{ComputeDecomp}\left(\rho, \boldsymbol{D}_\ell\right)$

end

$i = 0$

while *Stopping Criteria Not Met* **do**

  $\boldsymbol{s} = \text{GetData}(i)$

  $i = i + 1$

  $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_L =$

  $\text{MultiLayerPursuit}(\boldsymbol{s}, \alpha, \boldsymbol{\mu}, \boldsymbol{\lambda}, \boldsymbol{D}, (\rho\mathbf{I} + \boldsymbol{D}_1^T \boldsymbol{D}_1), \ldots, (\rho\mathbf{I} + \boldsymbol{D}_L^T \boldsymbol{D}_L))$

  $\mathscr{L} = \text{ComputeLoss}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_L)$

  $\nabla_{\hat{\boldsymbol{D}}_1} \mathscr{L}, \ldots, \nabla_{\hat{\boldsymbol{D}}_L} \mathscr{L} = \text{Backpropagation}(\mathscr{L}, \boldsymbol{D}_1, \ldots, \boldsymbol{D}_L)$

  $\boldsymbol{\Delta}\hat{\boldsymbol{D}}_1, \ldots, \boldsymbol{\Delta}\hat{\boldsymbol{D}}_L = \text{CalculateGradientStep}(\nabla_{\hat{\boldsymbol{D}}_1} \mathscr{L}, \ldots, \nabla_{\hat{\boldsymbol{D}}_L} \mathscr{L})$

  for $\ell \in \{1, \ldots, L\}$ **do**

    $\tilde{\boldsymbol{D}} = \text{Normalize}(\hat{\boldsymbol{D}}_\ell + \boldsymbol{\Delta}\hat{\boldsymbol{D}}_\ell)$

    $\boldsymbol{\Delta}\boldsymbol{D}_\ell = \text{Truncate}(\mathcal{F}^{-1}\,\text{LowRankApprox}(\tilde{\boldsymbol{D}} - \hat{\boldsymbol{D}}_\ell))$

    $\left(\rho\mathbf{I} + \boldsymbol{D}_\ell^T \boldsymbol{D}_\ell\right) = \text{UpdateDecomp}\left(\left(\rho\mathbf{I} + \boldsymbol{D}_\ell^T \boldsymbol{D}_\ell\right), \rho, \boldsymbol{D}_s, \boldsymbol{\Delta}\boldsymbol{D}\right)$

    $\boldsymbol{D}_\ell = \boldsymbol{D} + \boldsymbol{\Delta}\boldsymbol{D}$

    $\hat{\boldsymbol{D}}_\ell = \mathcal{F}(\boldsymbol{D}_\ell)$

  end

end

CHAPTER 4

JPEG ARTIFACT REMOVAL

4.1 Introduction

Despite the existence of better compression algorithms, use of the JPEG compression algorithm is ubiquitous: it is the most commonly used image compression algorithm. Overzealous JPEG compression can produce visible distortions, and image restoration from these distortions is a challenging problem. There are two aspects of JPEG compression which make the restoration process more challenging than simpler restoration problems like deblurring or removing salt-and-pepper noise: JPEG's block-based approach is not spatially invariant, and the quantization is nonlinear. This chapter describes a novel approach to address the challenges of JPEG image restoration using the ADMM-based convolutional sparse coding for a multi-layer dictionary model.

4.2 JPEG Algorithm

The JPEG compression process begins with an RGB image input, and consists of five steps. The first is a color transformation, transitioning from RGB to YUV. Then, the U and V color channels are downsampled. The DCT for each 8×8 block is computed (separately for each channel). The DCT coefficients are then quantized using a quantization matrix determined by a user-chosen JPEG quality factor. Finally, these quantized coefficients are reordered and encoded using a lossless variable length coding process.

The standard reconstruction process reverses the lossless encoding, computes the IDCT of the blocks, upsamples the color channels, and reverses the color transform.

4.3 Modeling Compressed JPEG Images

Some researchers have observed that convolutional dictionary models struggle with large smooth components of signals [37], likely due to the fact that shifted versions of smooth filters have high coherence.

For this reason, it is often a good idea to subtract a smoothed version $\boldsymbol{s}_{\text{smth}}$ of the signal, and only apply the dictionary model to the residual $\boldsymbol{s}_{\text{rough}}$.

$$\boldsymbol{s}_{\text{clean}} = \boldsymbol{s}_{\text{smth}} + \boldsymbol{s}_{\text{rough}} \tag{4.1}$$

$$\boldsymbol{s}_{\text{rough}} \approx \boldsymbol{D}_1 \boldsymbol{x}_1 \tag{4.2}$$

When restoring an image after JPEG compression, the original image $\boldsymbol{s}_{\text{clean}}$ is not known. Instead, the compressed image $\boldsymbol{s}$ is observed.

$$\boldsymbol{s} = \mathrm{q}(\boldsymbol{W} \boldsymbol{s}_{\text{clean}}) \tag{4.3}$$

$$\boldsymbol{s} \approx \mathrm{q}(\boldsymbol{W} (\boldsymbol{s}_{\text{smth}} + \boldsymbol{D}_1 \boldsymbol{x}_1)) \tag{4.4}$$

where $\boldsymbol{W}$ maps the signal to 8×8 block frequency coefficients (from the cosine transform), and $\mathrm{q}(\cdot)$ quantizes them.

A means of estimating $\boldsymbol{s}_{\text{smth}}$ from JPEG-compressed image $\boldsymbol{s}$ is discussed in the Practical Considerations chapter. However, for simplicity the experiments later in this chapter use a constant for $\boldsymbol{s}_{\text{smth}}$.

From this idea, I construct the pursuit problem:

$$\underset{\boldsymbol{x}}{\text{minimize}} \frac{\mu_1}{2}\|\boldsymbol{s} - \mathrm{q}(\boldsymbol{W}(\boldsymbol{D}_1\boldsymbol{x}_1 + \boldsymbol{s}_{\text{smth}}))\|_2^2 + \sum_{\ell=2}^{L} \frac{\mu_\ell}{2}\|\boldsymbol{x}_{\ell-1} - \boldsymbol{D}_\ell\boldsymbol{x}_\ell\|_2^2 + \sum_{\ell=1}^{L} \lambda_\ell\|\boldsymbol{x}_\ell\|_1 \tag{4.5}$$

with $\lambda_\ell \geq \mathbf{0}$.

With this multi-layer architecture, the first layer will construct small, simple filters, and subsequent layers superimpose those filters to build more complicated structures.

My approach to solve this problem uses the ADMM algorithm, where $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_L$ are the first set of primal variables, $\boldsymbol{v}, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_L$ are the second set of primal variables, and $\boldsymbol{\gamma}_1, \ldots, \boldsymbol{\gamma}_L$ are the dual variables corresponding to constraints on $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_L$. Here is the corresponding optimization problem:

$$\begin{aligned} &\underset{\boldsymbol{x},\boldsymbol{v},\boldsymbol{z}}{\text{minimize}} \frac{\mu_1}{2}\|\boldsymbol{v} - \boldsymbol{D}_1\boldsymbol{x}_1 - \boldsymbol{s}_{\text{smth}}\|_2^2 + \sum_{\ell=2}^{L} \frac{\mu_\ell}{2}\|\boldsymbol{z}_{\ell-1} - \boldsymbol{D}_\ell\boldsymbol{x}_\ell\|_2^2 + \sum_{\ell=1}^{L} \lambda_\ell\|\boldsymbol{z}_\ell\|_1 \\ &\text{subject to } \sqrt{\mu}\boldsymbol{R}_\ell^{-1}(\boldsymbol{z}_\ell - \boldsymbol{x}_\ell) = \mathbf{0} \\ &\qquad\qquad \mathrm{q}(\boldsymbol{W}\boldsymbol{v}) - \boldsymbol{s} = \mathbf{0} \end{aligned} \tag{4.6}$$

The constraint $\mathrm{q}(\boldsymbol{W}\boldsymbol{v}) - \boldsymbol{s} = \mathbf{0}$ is not an affine constraint because of the quantization. To resolve this, [8] approximate the quantization as a linear operator. However, the constraint is convex, so the constraint can be handled without approximation implicitly using an indicator function [38]. For now, I will focus on the other variable updates.

Setting $\boldsymbol{z}_0 = \boldsymbol{v} - \boldsymbol{s}_{\text{smth}}$, the updates for $\boldsymbol{x}$, $\boldsymbol{z}$, and $\boldsymbol{\gamma}$ are identical to those from the last chapter.

$$\boldsymbol{R}_\ell^{-1}\boldsymbol{x}_\ell^{(t+1)} = \left(\rho\mathbf{I} + (\boldsymbol{D}_\ell\boldsymbol{R}_\ell)^T\boldsymbol{D}_\ell\boldsymbol{R}_\ell\right)^{-1}\left((\boldsymbol{D}_\ell\boldsymbol{R}_\ell)^T\boldsymbol{z}_{\ell-1}^{(t)} + \rho\left(\boldsymbol{R}_\ell^{-1}\boldsymbol{z}_\ell^{(t)} + \frac{\boldsymbol{\gamma}_\ell^{(t)}}{\rho\sqrt{\mu_\ell}}\right)\right) \tag{4.7}$$

$$\boldsymbol{z}_\ell^{(t+1)} = (\mu_{\ell+1}\mathbf{I} + \rho\mu_\ell \boldsymbol{R}_\ell^{-2})^{-1}\,\mathrm{S}_{\lambda_\ell}\left(\mu_{\ell+1}\boldsymbol{D}_{\ell+1}\boldsymbol{x}_{\ell+1}^{(t+1)} + \rho\mu_\ell \boldsymbol{R}_\ell^{-1}\left(\boldsymbol{R}_\ell^{-1}\boldsymbol{x}_\ell^{(t+1)} - \frac{\boldsymbol{\gamma}_\ell^{(t+\frac{1}{2})}}{\rho\sqrt{\mu_\ell}}\right)\right) \tag{4.8}$$

$$\boldsymbol{z}_L^{(t+1)} = \boldsymbol{R}_\ell\,\mathrm{S}_{\frac{\lambda_L \boldsymbol{R}_L}{\rho\mu_L}}\left(\boldsymbol{R}_L^{-1}\boldsymbol{x}_L^{(t+1)} - \frac{\boldsymbol{\gamma}_L^{(t+\frac{1}{2})}}{\rho\sqrt{\mu_L}}\right) \tag{4.9}$$

$$\frac{\boldsymbol{\gamma}_\ell^{\left(t+\frac{1}{2}\right)}}{\rho\sqrt{\mu_\ell}} = \frac{\boldsymbol{\gamma}_\ell^{(t)}}{\rho\sqrt{\mu_\ell}} + (\alpha-1)(\boldsymbol{R}_\ell^{-1}\boldsymbol{z}_\ell^{(t)} - \boldsymbol{R}^{-1}\boldsymbol{z}_\ell^{(t+1)}) \tag{4.10}$$

$$\frac{\boldsymbol{\gamma}_\ell^{(t+1)}}{\rho\sqrt{\mu_\ell}} = \frac{\boldsymbol{\gamma}_\ell^{\left(t+\frac{1}{2}\right)}}{\rho\sqrt{\mu_\ell}} + \boldsymbol{R}_\ell^{-1}\boldsymbol{z}_\ell^{(t+1)} - \boldsymbol{R}^{-1}\boldsymbol{z}_\ell^{(t+1)} \tag{4.11}$$

The only remaining update equation is for $\boldsymbol{v}$. I will present a method for handling the quantization operator in the constraint in the next section.

4.4 Handling Quantization

Recall the optimization problem:

$$\begin{aligned} \underset{\boldsymbol{x},\boldsymbol{v},\boldsymbol{z}}{\text{minimize}} & \frac{\mu_1}{2}\|\boldsymbol{v} - \boldsymbol{D}_1\boldsymbol{x}_1 - \boldsymbol{s}_{\text{smth}}\|_2^2 + \sum_{\ell=2}^{L}\frac{\mu_\ell}{2}\|\boldsymbol{z}_{\ell-1} - \boldsymbol{D}_\ell\boldsymbol{x}_\ell\|_2^2 + \sum_{\ell=1}^{L}\lambda_\ell\|\boldsymbol{z}_\ell\|_1 \\ \text{subject to } & \sqrt{\mu}\boldsymbol{R}_\ell^{-1}(\boldsymbol{z}_\ell - \boldsymbol{x}_\ell) = \mathbf{0} \\ & \mathrm{q}(\boldsymbol{W}\boldsymbol{v}) - \boldsymbol{s} = \mathbf{0} \end{aligned} \tag{4.12}$$

For the $\boldsymbol{v}$ update, it is helpful to introduce a common convex-optimization trick. Consider the following function:

$$\mathbb{1}_{\{\mathrm{q}(\boldsymbol{W}\boldsymbol{v})-\boldsymbol{s}=\mathbf{0}\}} = \begin{cases} 0 & \mathrm{q}(\boldsymbol{W}\boldsymbol{v}) - \boldsymbol{s} = \mathbf{0} \\ +\infty & \text{otherwise} \end{cases} \tag{4.13}$$

The function is convex, and when included in an objective function, it implicitly enforces the constraint $\mathrm{q}(\boldsymbol{W}\boldsymbol{v}) - \boldsymbol{s} = \boldsymbol{0}$. This constraint was used in the same manner in [38] for JPEG artifact removal. This can be rewritten as the following:[1]

$$\mathbb{1}_{\{\mathrm{q}(\boldsymbol{W}\boldsymbol{v})-\boldsymbol{s}=\boldsymbol{0}\}} = \begin{cases} 0 & \boldsymbol{s} - \frac{\boldsymbol{q}}{2} \leq \boldsymbol{W}\boldsymbol{v} \leq \boldsymbol{s} + \frac{\boldsymbol{q}}{2} \\ +\infty & \text{otherwise} \end{cases} \tag{4.14}$$

where $\boldsymbol{q}$ is a vector representing the quantization interval for each element of the signal (in the 8×8 DCT domain). Adding the indicator function to the objective produces the following Lagrangian function:

$$\mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{v}, \boldsymbol{z}, \boldsymbol{\eta}, \boldsymbol{\gamma}) = \frac{\mu_1}{2}\|\boldsymbol{v} - \boldsymbol{D}_1\boldsymbol{x}_1 - \boldsymbol{s}_{\text{smth}}\|_2^2 + \psi(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\gamma}) + \mathbb{1}_{\{q(\boldsymbol{W}\boldsymbol{v})-\boldsymbol{s}=\boldsymbol{0}\}} \tag{4.15}$$

where $\psi(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\gamma})$ is a collection of terms irrelevant to the updates for $\boldsymbol{v}$.

Handling the cases in pointwise fashion, define function h as the following clipping operation:

$$\mathrm{h}(\boldsymbol{x}_1) = \begin{cases} \boldsymbol{s} + \frac{\boldsymbol{q}}{2} - \boldsymbol{W}(\boldsymbol{D}_1\boldsymbol{x}_1 + \boldsymbol{s}_{\text{smth}}) & \boldsymbol{W}(\boldsymbol{D}_1\boldsymbol{x}_1 + \boldsymbol{s}_{\text{smth}}) > \boldsymbol{s} + \frac{\boldsymbol{q}}{2} \\ \boldsymbol{s} - \frac{\boldsymbol{q}}{2} - \boldsymbol{W}(\boldsymbol{D}_1\boldsymbol{x}_1 + \boldsymbol{s}_{\text{smth}}) & \boldsymbol{W}(\boldsymbol{D}_1\boldsymbol{x}_1) + \boldsymbol{s}_{\text{smth}}) < \boldsymbol{s} - \frac{\boldsymbol{q}}{2} \\ \boldsymbol{0} & \text{otherwise} \end{cases} \tag{4.16}$$

Then,

$$\boldsymbol{v}^{(t+1)} = \boldsymbol{D}_1\boldsymbol{x}_1^{(t+1)} + \boldsymbol{s}_{\text{smth}} + \boldsymbol{W}^\dagger\,\mathrm{h}(\boldsymbol{x}_1^{(t+1)}) \tag{4.17}$$

where $\boldsymbol{W}^\dagger$ is the pseudo-inverse of $\boldsymbol{W}$. Figure 4.1 is a diagram showing how variables interact across layers and iterations.

[1]The set $\{\boldsymbol{v} : q(\boldsymbol{W}\boldsymbol{v}) - \boldsymbol{s} = \boldsymbol{0}\}$ does not include all boundary points. Equation 4.14 uses the closure of the set instead. This ensures that there is a minimizer of the augmented Lagrangian in respect to $\boldsymbol{v}$.

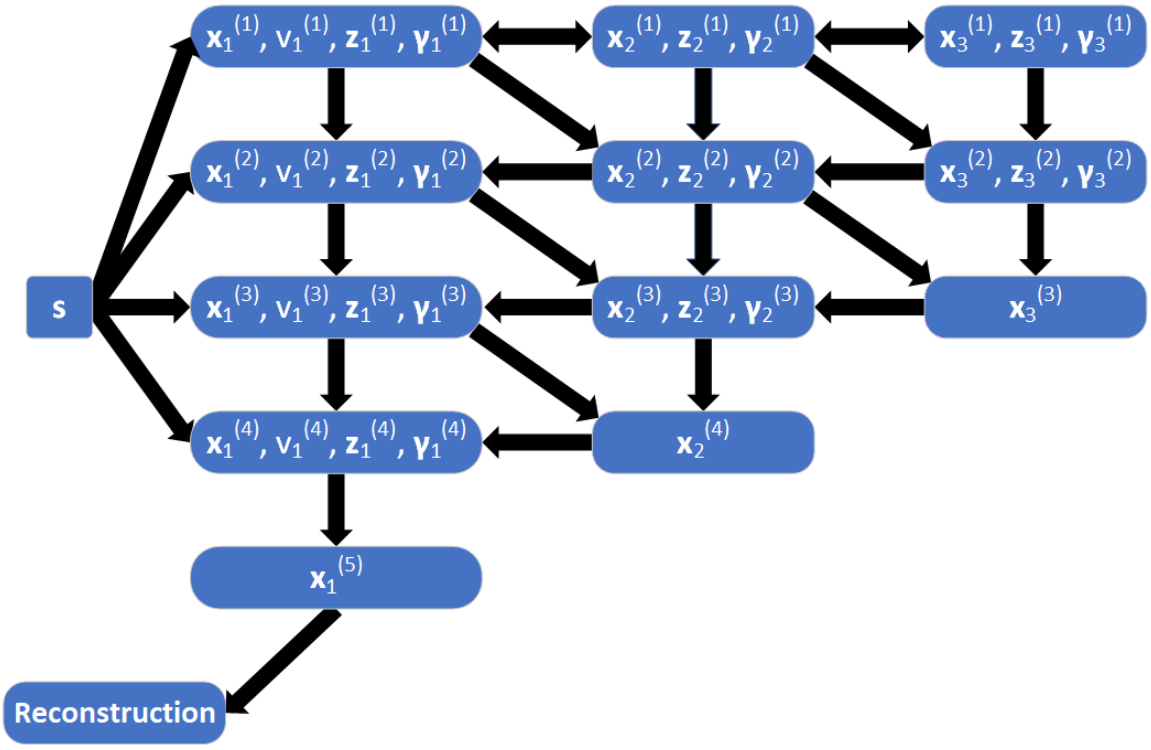


Figure 4.1: This diagram shows interactions between layers across iterations for the ADMM pursuit algorithm applied to JPEG artifact removal. The double-sided arrows at the top of the diagram go both directions because $\boldsymbol{x}_{\ell+1}$ influences $\boldsymbol{z}_{\ell}$ during initialization.

4.5 Backpropagation Approximation

If an iterative sparse coding algorithm converges, after a sufficient number of iterations, the coefficient updates become very small with each iteration, and so the coefficients change very little. Backpropagation can be computationally and memory intensive, and so intuition would suggest backpropagating through steps that are barely even changing the coefficients may not be an optimal use of computational resources. For single-layer dictionary learning, it is very common for practitioners and researchers to iteratively decrease the objective function in respect to the dictionary using gradient methods, treating the coefficients as fixed (rather than a function of the dictionary). Inspired by those methods, I similarly compute approximate gradients without backpropagating through the entire sparse coding algorithm, instead only backpropagating to the last instance of that dictionary's use:[2]

$$\nabla_{\boldsymbol{D}_1}\mathscr{L} \approx \nabla_{\boldsymbol{D}_1}^{\left(\boldsymbol{x}_1^{(T)} \to \boldsymbol{D}_1\boldsymbol{x}_1^{(T)}\right)}\mathscr{L} \tag{4.18}$$

$$\nabla_{\boldsymbol{D}_\ell}\mathscr{L} \approx \nabla_{\boldsymbol{D}_\ell}^{\left(\boldsymbol{x}_\ell^{(T-\ell+1)} \to \boldsymbol{D}_\ell\boldsymbol{x}_\ell^{(T-\ell+1)}\right)}\mathscr{L} \tag{4.19}$$

This approximation[3] of the gradient for dictionary learning is used in the experiment detailed in the subsequent section.

4.6 Experiment

In this section, I apply this novel multi-layer dictionary approach to other multi-layer dictionary algorithms on the task of removing artifacts from JPEG compression.

[2]The last use of $\boldsymbol{D}_{\ell+1}$ occurs in a $\boldsymbol{z}_\ell$ update, and $\boldsymbol{D}_1\boldsymbol{x}_1^{(T)}$ approximately reconstructions the original signal.

[3]It may help to recall from Equations 4.7 and 4.8 that $\boldsymbol{x}_{\ell+1}^{(t)}$ has no influence on $\boldsymbol{x}_\ell^{(t)}$; instead, it is used to update $\boldsymbol{z}_\ell^{(t)}$ which subsequently is used in the $\boldsymbol{x}_\ell^{(t+1)}$ update equation. For this reason, Equation 4.19 depends on $\boldsymbol{x}_\ell^{(T-\ell+1}$, not $\boldsymbol{x}_\ell^{(T)}$. These types of interactions between variables across layers and iterations are shown in Figure 4.1.

Table 4.1: Architecture and Hyper-Parameters for Experiment

Algorithm	Hyperparameter	Num. of Iter.	Num. of Filters		Filter Size	
			Layer 1	Layer 2	Layer 1	Layer 2
ADMM	$\rho = 1$, $\mathcal{R} = 4$	16	32	64	5×5	7×7
FISTA	$\mathbb{L} = 64$	48	32	64	5×5	7×7

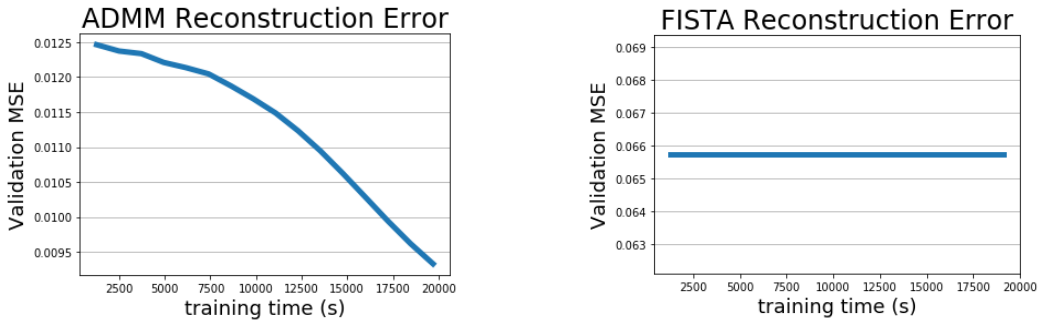
4.6.1 Experiment Setup

The BSDS500 dataset consists of 200 training images, 100 validation images, and 200 test images, and was originally designed to test segmentation algorithms. For this experiment, I compress the images using a quality factor of 25. For training, the algorithms are given both the compressed and raw images. The images vary in size, so I split the images into smaller 32×32 patches. For validation and testing, the algorithms are assessed on how well they reconstruct original image patches from the compressed image patches.

4.6.2 Network Architecture

To compare the FISTA and ADMM for multi-layer dictionary learning, both algorithms are set up with the same basic 2-layer architecture, the details of which are shown in Table 4.1. For both, coefficients of the last layer are constrained to be non-negative.[4] The reason I use $\mathbb{L} = 64$ for FISTA is because $\mathbb{L} = 16$ diverged during sparse coding. For dictionary learning, I use stochastic gradient descent for both, using a step-size of 0.01. Backpropagation is used to learn not just elements of the dictionary, but also the weighting of the penalty terms μ_ℓ and λ_ℓ. The loss minimized is the mean-squared error of the reconstruction in respect to the raw (pre-compression) images. To reduce computation time and memory usage, backprogagation for the dictionary updates is only computed over the last few computations as described in section 4.5.

[4]For ADMM, this is enforced on the $\boldsymbol{z}$ coefficients by adding an indicator function to the objective $\mathbb{1}_{\boldsymbol{z}_L \geq \mathbf{0}}(\boldsymbol{z}_L)$.



(a) ADMM

(b) FISTA

Figure 4.2: This is the reconstruction error for the multi-layer dictionary model measured on the validation set shown as a function of training time. Subfigure 4.2a shows the error for the ADMM algorithm and Subfigure 4.2b shows the error for the FISTA algorithm.

4.6.3 Results

The decrease in reconstruction error across a validation set for ADMM is shown in Figure 4.2a. ADMM is able to reduce the error, improving the dictionary across many updates. A similar curve is shown for FISTA in Figure 4.2b trained on the same data for the same number of dictionary updates. The dictionary updates for FISTA fail to reduce the error. FISTA's failure to successfully learn a better dictionary likely stems from its slower convergence in the sparse coding task. Figures 4.3 shows the sparse coding error across multiple iterations (using ADMM's z coefficients). ADMM's faster convergence allows it to improve the dictionary without backpropagating through the entire network. Even with three times the number of sparse coding iterations, FISTA is still unable to converge fast enough. To succeed, FISTA would either need to backpropagate further through the network, or add more sparse coding iterations, both of which would further increase needed computational resources.

4.7 Conclusion

This chapter showed how to adapt the multi-layer dictionary approach from chapter 3 to the JPEG artifact removal problem. This involved combining the multi-layer model with the

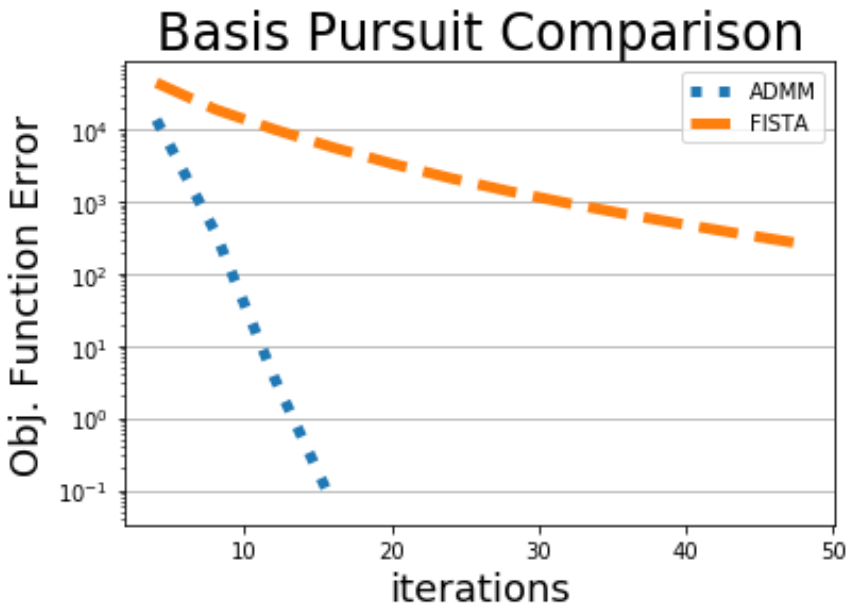


Figure 4.3: This plots the objective function from equation 4.5 in respect to iterations, using z_ℓ for the coefficients of the ADMM approach.

convex indicator function from [38] to impose the quantization constraints, subtracting out the low-frequency components, and approximating the gradients. With my novel approach, ADMM handled the gradient approximations better than FISTA did, and demonstrated faster convergence for sparse coding.

CHAPTER 5

PRACTICAL CONSIDERATIONS

5.1 Boundary Handling

Convolution and circular convolution can produce boundary effects that can impact performance. In the JPEG compression artifact removal chapter, $\boldsymbol{W}$ is defined as the function producing 8×8 block DCT coefficients from an RGB image. However, to mitigate boundary effects this should in practice also include cropping the image. This allows $\boldsymbol{D}_1 \boldsymbol{x}_1$ to reconstruct the signal outside of the measured space (similar to the "boundary masking" in [39], but with the constraint and objective term swapped). Initialization is important however [40]. Reflecting across the boundary is a much better initialization approach than zeros, but if using patches, actual (compressed image) measurements would provide an even better initialization. For the experiments in chapter 4, I use these compressed image measurements.

5.2 Removing Low-Frequency Signal Content

Convolutional dictionary models tend to struggle with low-frequency signal content, so ideally steps should be taken to avoid representing it with the dictionary model. Tikhonov regularization is a straight-forward means to remove low-frequency content [37]:

$$\boldsymbol{s}_{\text{smth}} = \arg\min_{\boldsymbol{x}} \frac{1}{2} \|\boldsymbol{s} - \boldsymbol{x}\|_2^2 + \lambda \| \sum_i \boldsymbol{G}_i \boldsymbol{x} \|_2^2 \tag{5.1}$$

where $\boldsymbol{G}_i$ computes a discrete gradient in the ith direction. This has a closed form solution and can be easily solved in frequency domain.

5.2.1 JPEG Artifact Removal

Tikhonov regularization can struggle with block artifacts from JPEG compression. Rather than smooth the compressed JPEG image itself, it is better to solve for a smoothed version of a signal that compresses to the compressed JPEG image. This problem can be solved using ADMM.

$$\underset{\boldsymbol{x},\boldsymbol{v}}{\text{minimize}}\, \frac{1}{2}\|\boldsymbol{v}-\boldsymbol{x}\|_2^2 + \lambda\|\sum_i \boldsymbol{G}_i\boldsymbol{x}\|_2^2 + \mathbb{1}_{\{\boldsymbol{Q}(\boldsymbol{W}\boldsymbol{v})-\boldsymbol{s}=\boldsymbol{0}\}} \tag{5.2}$$

There are no constraints, so no dual variables are needed. Just alternate between a Tikhonov update for $\boldsymbol{x}$ and an update for $\boldsymbol{v}$ that closely resembles the $\boldsymbol{v}$ update from the last chapter. Handling the cases in pointwise fashion, define function h as the following clipping operation:

$$\mathrm{h}(\boldsymbol{x}) = \boldsymbol{W}(\boldsymbol{v}-\boldsymbol{x}) = \begin{cases} \boldsymbol{s}+\frac{\boldsymbol{q}}{2}-\boldsymbol{W}(\boldsymbol{x}) & \boldsymbol{W}(\boldsymbol{x}) > \boldsymbol{s}+\frac{\boldsymbol{q}}{2} \\ \boldsymbol{s}-\frac{\boldsymbol{q}}{2}-\boldsymbol{W}(\boldsymbol{x}) & \boldsymbol{W}(\boldsymbol{x}) < \boldsymbol{s}-\frac{\boldsymbol{q}}{2} \\ \boldsymbol{0} & \text{otherwise} \end{cases} \tag{5.3}$$

Then,

$$\boldsymbol{v} = \boldsymbol{x} + \boldsymbol{W}^\dagger\, \mathrm{h}(\boldsymbol{x}) \tag{5.4}$$

In the final update for $\boldsymbol{x}$, it may help to increase λ. This will make the resulting smoothed image slightly more smooth, which allows the rough signal component to keep more information content from the image to compute better coefficients.

5.3 Inverse Representation Drift

Rank-1 Hermitian updates to a Cholesky factorization or other inverse representation can be computed in quadratic time. However, these algorithms compute the new inverse represen-

tation from the old one, compounding precision error. Over the course of many iterations, the inverse representation will become less and less accurate. It is necessary to occasionally recompute the inverse representation in cubic time, to mitigate the impact of precision errors. One method to determine when to recompute the inverse representation is to measure the error of the inverse function in reconstructing a random set of vectors $\hat{\boldsymbol{u}}$, and recompute the inverse representation in cubic time when the error ϵ exceeds a threshold.

$$\epsilon = \max_i \|(\rho\mathbf{I} + (\hat{\boldsymbol{D}}\boldsymbol{R})^T\hat{\boldsymbol{D}}\boldsymbol{R})^{-1}(\rho\hat{\boldsymbol{u}}_i + (\hat{\boldsymbol{D}}\boldsymbol{R})^T\hat{\boldsymbol{D}}\boldsymbol{R}\hat{\boldsymbol{u}}_i\|_\infty \tag{5.5}$$

5.4 Tensorflow and Keras

Most of the computations for my research rely on TensorFlow version 2.3.1 and version 2.4.1 [36], a Python library for machine learning specializing in building models with differentiable, parameterizable composite functions and learning model parameters using gradient descent or other gradient-based optimization methods. TensorFlow is a common platform for researchers and developers working on artificial neural networks, and there are many tutorials and examples freely available online, so I will not replicate that work here. This chapter section the reader already has some familiarity with TensorFlow and Keras [41] (a high-level library inside TensorFlow). The goal of this section is to provide the reader with the tools and workarounds to be able to replicate my work without resorting to hacking things together with gradient tape and/or TensorFlow-1-style code.

5.4.1 Why Not Use Gradient Tape and TensorFlow-1-Style Code?

Keras offers a high-level environment. Code written in Keras's framework is easier to integrate with other work. Gradient tape is great for hacking something together or debugging, but promotes styles of coding that are less readable, less maintainable, and less portable. Keras also has a lower learning curve than the broader TensorFlow library.

5.4.2 Shared Weights Between Layers

Trainable TensorFlow variables declared outside of any Keras layer will not be automatically added to a Keras model's list of trainable variables. In most cases, this limitation is not a problem; it is intuitive to declare a layer's weights inside that layer. However, sometimes the same variable is needed in multiple distinct layers. To include a variable in the model's trainable variables, it is sufficient to declare the variable in one layer and pass the variable (or the layer it was initialized in) as an input argument to the __init__ function of the other layers that share that variable. This will work even if the Keras model does not use the layer that declared the variable. Keras users should be careful when combining Keras classes with non-Keras classes, as certain class structures can prevent Keras from detecting some trainable variables.[1]

5.4.3 Custom Partial Gradients

TensorFlow offers a well-documented means of replacing TensorFlow's gradient computations of an operation with specified custom gradient computations. However, if the operation involves multiple tensors that are inputs or trainable variables, the standard approach replaces all the gradients with custom gradients. If TensorFlow's gradient computations are sufficient for some tensors but not others, a workaround is necessary. This workaround is best explained by example.

Suppose the operation is the following:

```
z = f(x,y)
```

for which the standard TensorFlow gradient computations of f are desired in respect to x, but the custom gradient computations desired in respect to y are specified in function $g(\nabla_z \mathcal{L})$. This can be rewritten as the following:

[1] If a Keras layer instantiates a non-Keras class that instantiates a Keras class that creates a trainable variable, Keras will fail to detect the trainable variable, and the variable will not be updated during training or saved when the model weights are saved.

```
@tf.custom_gradient
def h(z,y):
   def grad_fun(grad):
      return (tf.identity(grad),g(grad))
   return z,grad_fun
z = f(x,tf.stop_gradient(y))
z = h(z,y)
```

The function h does nothing on the forward pass, but in the backward pass computes the custom gradient in respect to y as intended.

5.4.4 Updating TensorFlow Variables After Applying Gradients

To update TensorFlow Variables after applying gradients, it is necessary to track which variables are affected and what their corresponding update functions are. To accomplish this, I store the update functions in a Python dictionary using variable names as the dictionary keys. This Python dictionary needs to be widely accessible so that layers can add update functions when they are initialized; a simple way to do this is to make the update function Python dictionary a class attribute. The keys need to be unique, but TensorFlow variable names can conflict. It is easy to avoid this problem by checking for conflicts before adding a new update function.

```
class PostPrc:
   update = {}
   def add_update(varName,update_fun):
      assert varName not in PostPrc.update
      PostPrc.update[varName] = update_fun
```

In the standard Keras training paradigm, models are trained using the fit function, a method in the Keras model object. The fit function calls the function train_step, where gradients are applied. To update TensorFlow Variables after gradients are applied in Ten-

sorflow 2.3.1 on a CPU, train_step is the function to modify. The only change that needs to be made is adding a function call to all update functions that correspond to the model's list of trainable variables.

```
class Model_subclass(tf.keras.Model):
   def train_step(self,data):
      trainStepOutputs =
         tf.keras.Model.train_step(self,data)
      update_ops = []
      for tv in self.trainable_variables:
         if tv.name in PostPrc.update:
            PostPrc.update[tv.name]()
      return trainStepOutputs
```

Changes to Tensorflow variables in the update function must use the assign command (or its variants: assign_add, assign_sub, ect). Otherwise, TensorFlow will detect that computations lie outside of its computational graph and throw an error. Note that using the assign command on Python variables that are not TensorFlow variables will produce some very cryptic error messages, so be sure to use the assign command correctly. If the value change of one TensorFlow variable depends on the value of another TenorFlow variable value pre-update, it may be necessary to use the Tensorflow control_dependencies command to get TensorFlow to track that dependency. TensorFlow has a useful tool called TensorBoard that helps visualize TensorFlow's dependencies, but a workaround is required to use TensorBoard on update functions that are called after applying gradients. To use TensorBoard to visualize dependencies in an update function, temporarily call the update function in the layer's call method, use TensorBoard to verify all necessary dependencies are being tracked, then remove the update function call from the layer's call method.

However, in Tensorflow 2.4.1 on a GPU, the above method fails. The PostPrc class is still fine, but the update functions must be instead be called using Keras callbacks.

```
class
   PostPrcCallback(keras.callbacks.Callback,PostPrc):
   def on_train_begin(self,logs):
      logs = logs or {}
      self.update = []
      for tv in self.model.trainable_variables:
         if tv.name in PostPrc.update:
            self.update.append(PostPrc.update[tv.name])

   def on_batch_end(self, epoch, logs=None):
      logs = logs or {}

      for an_update in self.update:
         an_update()

      for k, v in logs.items():
         self.history.setdefault(k, []).append(v)
```

5.4.5 The Perils of Using Built-In Functions for Complex Tensors and Arrays

Complex numbers are not as frequently used by researchers and practitioners, and many tools and platforms fail on complex cases. Tensorflow's assign_add command can fail on the GPU, so users should use the assign command instead. It is necessary in some cases to split complex variables into their real and imaginary components to avoid GPU errors. The TensorFlow Probability version 0.11.1 [42] is an extension of TensorFlow mostly used for probabilistic models. The library contains a Cholesky update function, but the function does not properly handle complex inputs. To compute Cholesky updates for complex inputs, modifications of this code are necessary. Similarly, the Randomized SVD

algorithm in the Python scikit-learn library does not properly handle complex inputs.

Errors like these are fairly common, so when dealing with complex data, researchers and practitioners should carefully verify that the function libraries they rely on are properly handling complex numbers. It is also important for code to be highly modularized, because if an error does not occur until computations on the computational graph or saving weights, platforms may not be able to identify where in the code the error is coming from, and error messages can be cryptic or even wrong.

CHAPTER 6
CONCLUSION

This dissertation has presented a novel dictionary learning algorithm for signals with a large number of channels and its application to multi-layer dictionary models. This algorithm can be used for JPEG artifact removal, and shows some promise as a competitor to the FISTA algorithm, given its faster convergence in solving the sparse-coding problem, but does have larger memory requirements than FISTA. In addition, appendix A generalizes the work of [43] to handle complex numbers, which to my knowledge has not been done previously. [43] efficiently computes rank-1 updates for a Cholesky factorization.

In addition to seeking other applications, future expansion on this research might look to adapt ρ during training. While ρ must remain fixed for efficient updates of the Cholesky decomposition (or LDLT decomposition), drift over many iterations eventually must be rectified by recomputing the entire decomposition. At such time, an update to ρ would require no additional computational cost. While many methods exist to adapt ρ for ADMM [44][45], these methods are designed to adjust ρ far more frequently. Another potential area of future research lies in tailoring momentum methods for dictionary updates to this novel dictionary learning algorithm. While the dictionary learning algorithm is not incompatible with momentum-based dictionary updates, conventional momentum methods [33][34] will not take into account the low-rank approximation step, which may hurt performance. Finally, since the memory requirements promote the use of frames to decrease the signal size, there may be a need in some applications to combine results across frames. Fortunately, given the popularity of frame-based dictionary models, there is an existing body of work to build off of for this task [46][47][48][49].

Appendices

APPENDIX A

HERMITIAN RANK-1 UPDATES FOR THE CHOLESKY DECOMPOSITION

These derivations are based on the work of [43], but modified to handle complex numbers.

Let $\boldsymbol{A}^{(n)} = \boldsymbol{L}\boldsymbol{L}^H$ be a Hermitian, positive definite matrix of dimension $C \times C$ and $\boldsymbol{L}$ be a lower triangular matrix. Then, the diagonal of $\boldsymbol{L}$ is positive and real.

$$\boldsymbol{A}^{(n)} = \begin{bmatrix} a_{1,1} & \dots & a_{C,1}^* \\ \vdots & \ddots & \vdots \\ a_{C,1} & \dots & a_{C,C} \end{bmatrix} \tag{A.1}$$

$$\boldsymbol{L} = \begin{bmatrix} \ell_{1,1} & 0 & \dots & 0 \\ \ell_{2,1} & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ \ell_{C,1} & \dots & & \ell_{C,C} \end{bmatrix} \tag{A.2}$$

Dividing $\boldsymbol{A}^{(n)}$ into blocks:

$$\boldsymbol{A}^{(n)} = \begin{bmatrix} \ell_{1,1} & \boldsymbol{0}^T \\ \boldsymbol{l}_{2,1} & \boldsymbol{L}_{2,2} \end{bmatrix} \begin{bmatrix} \ell_{1,1}^* & \boldsymbol{l}_{2,1}^H \\ \boldsymbol{0} & \boldsymbol{L}_{2,2}^H \end{bmatrix} \tag{A.3}$$

$$\boldsymbol{A}^{(n)} = \begin{bmatrix} \ell_{1,1}^2 & \ell_{1,1}\boldsymbol{l}_{2,1}^H \\ \ell_{1,1}\boldsymbol{l}_{2,1} & \boldsymbol{L}_{2,2}\boldsymbol{L}_{2,2}^H + \boldsymbol{l}_{2,1}\boldsymbol{l}_{2,1}^H \end{bmatrix} \tag{A.4}$$

Consider the rank-1 update:[1]

$$\boldsymbol{A}^{(n+1)} = \boldsymbol{A}^{(n)} + \lambda \boldsymbol{v}\boldsymbol{v}^H \tag{A.5}$$

[1] If λ is negative, the result is not guaranteed to be positive definite. In general, updates to inverse representations with a negative λ (sometimes called "downdates") are less numerically stable, even if the resulting matrix is still positive definite.

where

$$\boldsymbol{v} = \begin{bmatrix} v_1 \\ \boldsymbol{v}_2 \end{bmatrix} \tag{A.6}$$

$$\boldsymbol{A}^{(n+1)} = \begin{bmatrix} \ell_{1,1}^2 + \lambda v_1 v_1^* & \ell_{1,1}\boldsymbol{l}_{2,1}^H + \lambda v_1 \boldsymbol{v}_2^H \\ \ell_{1,1}\boldsymbol{l}_{2,1} + \lambda v_1^* \boldsymbol{v}_2 & \boldsymbol{L}_{2,2}\boldsymbol{L}_{2,2}^H + \boldsymbol{l}_{2,1}\boldsymbol{l}_{2,1}^H + \lambda \boldsymbol{v}_2 \boldsymbol{v}_2^H \end{bmatrix} \tag{A.7}$$

Let $\boldsymbol{A}^{(n+1)} = \boldsymbol{M}\boldsymbol{M}^H$, where $\boldsymbol{M}$ is a lower triangular matrix. Then, the diagonal of $\boldsymbol{M}$ is positive and real.

$$\boldsymbol{A}^{(n+1)} = \begin{bmatrix} m_{1,1}^2 & m_{1,1}\boldsymbol{m}_{2,1}^H \\ m_{1,1}\boldsymbol{m}_{2,1} & \boldsymbol{M}_{2,2}\boldsymbol{M}_{2,2}^H + \boldsymbol{m}_{2,1}\boldsymbol{m}_{2,1}^H \end{bmatrix} \tag{A.8}$$

Therefore,

$$m_{1,1}^2 = \ell_{1,1}^2 + \lambda v_1 v_1^* \tag{A.9}$$

$$m_{1,1}\boldsymbol{m}_{2,1} = \ell_{1,1}\boldsymbol{l}_{2,1} + \lambda v_1^* \boldsymbol{v}_2 \tag{A.10}$$

$$\boldsymbol{M}_{2,2}\boldsymbol{M}_{2,2}^H + \boldsymbol{m}_{2,1}\boldsymbol{m}_{2,1}^H = \boldsymbol{L}_{2,2}\boldsymbol{L}_{2,2}^H + \boldsymbol{l}_{2,1}\boldsymbol{l}_{2,1}^H + \lambda \boldsymbol{v}_2 \boldsymbol{v}_2^H \tag{A.11}$$

Solving for the first column of $\boldsymbol{M}$:

$$m_{1,1} = \sqrt{\ell_{1,1}^2 + \lambda v_1 v_1^*} \tag{A.12}$$

$$\boldsymbol{m}_{2,1} = \frac{\ell_{1,1}\boldsymbol{l}_{2,1} + \lambda v_1^* \boldsymbol{v}_2}{m_{1,1}} \tag{A.13}$$

Finally,

$$\boldsymbol{M}_{2,2}\boldsymbol{M}_{2,2}^H = \boldsymbol{L}_{2,2}\boldsymbol{L}_{2,2}^H + \boldsymbol{l}_{2,1}\boldsymbol{l}_{2,1}^H + \lambda \boldsymbol{v}_2 \boldsymbol{v}_2^H - \boldsymbol{m}_{2,1}\boldsymbol{m}_{2,1}^H \tag{A.14}$$

$$\boldsymbol{m}_{2,1}\boldsymbol{m}_{2,1}^H = \frac{1}{m_{1,1}^2}\left(\ell_{1,1}^2\boldsymbol{l}_{2,1}\boldsymbol{l}_{2,1}^H + \lambda\ell_{1,1}v_1\boldsymbol{l}_{2,1}\boldsymbol{v}_2^H + \lambda\ell_{1,1}v_1^*\boldsymbol{v}_2\boldsymbol{l}_{2,1}^H + \lambda^2 v_1 v_1^*\boldsymbol{v}_2\boldsymbol{v}_2^H\right) \quad \text{(A.15)}$$

$$\boldsymbol{M}_{2,2}\boldsymbol{M}_{2,2}^H = \boldsymbol{L}_{2,2}\boldsymbol{L}_{2,2}^H + \frac{m_{1,1}^2 - \ell_{1,1}^2}{m_{1,1}^2}\boldsymbol{l}_{2,1}\boldsymbol{l}_{2,1}^H + \frac{\lambda(m_{1,1}^2 - \lambda v_1 v_1^*)}{m_{1,1}^2}\boldsymbol{v}_2\boldsymbol{v}_2^H - \left(\frac{\lambda\ell_{1,1}v_1}{m_{1,1}^2}\boldsymbol{l}_{2,1}\boldsymbol{v}_2^H + \frac{\lambda\ell_{1,1}v_1^*}{m_{1,1}}\boldsymbol{v}_2\boldsymbol{l}_{2,1}^H\right) \quad \text{(A.16)}$$

The expressions $m_{1,1}^2 - \lambda v_1 v_1^*$ and $m_{1,1}^2 - \ell_{1,1}^2$ can be simplified using equation A.12.

$$\boldsymbol{M}_{2,2}\boldsymbol{M}_{2,2}^H = \boldsymbol{L}_{2,2}\boldsymbol{L}_{2,2}^H + \frac{\lambda v_1 v_1^*}{m_{1,1}^2}\boldsymbol{l}_{2,1}\boldsymbol{l}_{2,1}^H + \frac{\lambda\ell_{1,1}^2}{m_{1,1}^2}\boldsymbol{v}_2\boldsymbol{v}_2^H - \frac{\lambda\ell_{1,1}v_1}{m_{1,1}}\boldsymbol{l}_{2,1}\boldsymbol{v}_2^H - \frac{\lambda\ell_{1,1}v_1^*}{m_{1,1}^2}\boldsymbol{v}_2\boldsymbol{l}_{2,1}^H \quad \text{(A.17)}$$

Factoring out $\frac{\lambda}{m_{1,1}^2}$:

$$\boldsymbol{M}_{2,2}\boldsymbol{M}_{2,2}^H = \boldsymbol{L}_{2,2}\boldsymbol{L}_{2,2}^H + \frac{\lambda}{m_{1,1}^2}\left(v_1 v_1^*\boldsymbol{l}_{2,1}\boldsymbol{l}_{2,1}^H + \ell_{1,1}^2\boldsymbol{v}_2\boldsymbol{v}_2^H - \ell_{1,1}v_1\boldsymbol{l}_{2,1}\boldsymbol{v}_2^H - \ell_{1,1}v_1^*\boldsymbol{v}_2\boldsymbol{l}_{2,1}^H\right) \quad \text{(A.18)}$$

Note the factorization:

$$(\ell_{1,1}\boldsymbol{v}_2 - v_1\boldsymbol{l}_{2,1})(\ell_{1,1}\boldsymbol{v}_2 - v_1\boldsymbol{l}_{2,1})^H = \ell_{1,1}^2\boldsymbol{v}_2\boldsymbol{v}_2^H - \ell_{1,1}v_1^*\boldsymbol{l}_{2,1}^H - \ell_{1,1}v_1\boldsymbol{l}_{2,1}\boldsymbol{v}_2 + v_1 v_1^*\boldsymbol{l}_{2,1}\boldsymbol{l}_{2,1}^H \quad \text{(A.19)}$$

Therefore,

$$\boldsymbol{M}_{2,2}\boldsymbol{M}_{2,2}^H = \boldsymbol{L}_{2,2}\boldsymbol{L}_{2,2}^H + \frac{\lambda}{m_{1,1}^2}\left(\ell_{1,1}\boldsymbol{v}_2 - v_1\boldsymbol{l}_{2,1}\right)\left(\ell_{1,1}\boldsymbol{v}_2 - v_1\boldsymbol{l}_{2,1}\right)^H \quad \text{(A.20)}$$

$\boldsymbol{L}_{2,2}\boldsymbol{L}_{2,2}^H$ is a $(C-1)\times(C-1)$ Hermitian, positive definite matrix and $\frac{\lambda}{m_{1,1}^2}(\ell_{1,1}\boldsymbol{v}_2 - v_1\boldsymbol{l}_{2,1})(\ell_{1,1}\boldsymbol{v}_2 - v_1\boldsymbol{l}_{2,1})^H$ is a rank-1 Hermitian update, so the process can be repeated on subsequent columns of $\boldsymbol{L}$ until the entire Cholesky decomposition has been updated. Each

column update is computed in linear time, so the entire update can be computed in quadratic time.

While the order of complexity cannot be further reduced, there are changes that can be made to decrease precision error. Factoring out $\ell_{1,1}$:

$$\boldsymbol{M}_{2,2}\boldsymbol{M}_{2,2}^H = \boldsymbol{L}_{2,2}\boldsymbol{L}_{2,2}^H + \frac{\lambda\ell_{1,1}^2}{m_{1,1}^2}\left(\boldsymbol{v}_2 - \frac{v_1}{\ell_{1,1}}\boldsymbol{l}_{2,1}\right)\left(\boldsymbol{v}_2 - \frac{v_1}{\ell_{1,1}}\boldsymbol{l}_{2,1}\right)^H \tag{A.21}$$

Rather than directly updating λ before moving on to the next column, better precision can be achieved by updating a divisor instead. Looking at the fraction $\frac{\ell_{1,1}^2}{m_{1,1}^2}$:

$$\frac{\ell_{1,1}^2}{m_{1,1}^2} = \frac{\ell_{1,1}^2}{\ell_{1,1}^2 + \lambda v_1 v_1^*} \tag{A.22}$$

$$\frac{\ell_{1,1}^2}{m_{1,1}^2} = \frac{1}{1 + \frac{\lambda v_1 v_1^*}{\ell_{1,1}^2}} \tag{A.23}$$

Therefore,

$$\boldsymbol{M}_{2,2}\boldsymbol{M}_{2,2}^H = \boldsymbol{L}_{2,2}\boldsymbol{L}_{2,2}^H + \frac{\lambda}{1 + \frac{\lambda v_1 v_1^*}{\ell_{1,1}^2}}(\boldsymbol{v}_2 - \frac{v_1}{\ell_{1,1}}\boldsymbol{l}_{2,1})(\boldsymbol{v}_2 - \frac{v_1}{\ell_{1,1}}\boldsymbol{l}_{2,1})^H \tag{A.24}$$

Let

$$\boldsymbol{\omega} = \boldsymbol{v}_2 - \frac{v_1}{\ell_{1,1}}\boldsymbol{l}_{2,1} \tag{A.25}$$

So,

$$\boldsymbol{M}_{2,2}\boldsymbol{M}_{2,2}^H = \boldsymbol{L}_{2,2}\boldsymbol{L}_{2,2}^H + \frac{\lambda}{1 + \frac{\lambda v_1 v_1^*}{\ell_{1,1}^2}}\boldsymbol{\omega}\boldsymbol{\omega}^H \tag{A.26}$$

Previously, $\boldsymbol{m}_{2,1}$ was written in terms of $\boldsymbol{l}_{2,1}$ and $\boldsymbol{v}$. It can instead be written in terms of $\boldsymbol{l}_{2,1}$ and $\boldsymbol{\omega}$. Recall that

$$\boldsymbol{m}_{2,1} = \frac{\ell_{1,1}\boldsymbol{l}_{2,1} + \lambda v_1^* \boldsymbol{v}_2}{m_{1,1}} \tag{A.27}$$

Substituting for $\boldsymbol{v}_2$:

$$\boldsymbol{m}_{2,1} = \frac{\ell_{1,1}\boldsymbol{l}_{2,1} + \lambda v_1^*(\boldsymbol{\omega} + \frac{v_1}{\ell_{1,1}}\boldsymbol{l}_{2,1})}{m_{1,1}} \tag{A.28}$$

Combining the $\boldsymbol{l}_{2,1}$ terms:

$$\boldsymbol{m}_{2,1} = \frac{(\ell_{1,1} + \frac{\lambda v_1 v_1^*}{\ell_{1,1}})\boldsymbol{l}_{2,1} + \lambda v_1^*\boldsymbol{\omega}}{m_{1,1}} \tag{A.29}$$

Factoring out $\frac{1}{\ell_{1,1}}$ produces the expression for $m_{1,1}^2$.

$$\boldsymbol{m}_{2,1} = \frac{\frac{1}{\ell_{1,1}}(\ell_{1,1}^2 + \lambda v_1 v_1^*)\boldsymbol{l}_{2,1} + \lambda v_1^*\boldsymbol{\omega}}{m_{1,1}} \tag{A.30}$$

$$\boldsymbol{m}_{2,1} = \frac{\frac{m_{1,1}^2}{\ell_{1,1}}\boldsymbol{l}_{2,1} + \lambda v_1^*\boldsymbol{\omega}}{m_{1,1}} \tag{A.31}$$

$$\boldsymbol{m}_{2,1} = \frac{m_{1,1}}{\ell_{1,1}}\boldsymbol{l}_{2,1} + \frac{\lambda v_1^*}{m_{1,1}}\boldsymbol{\omega} \tag{A.32}$$

So, to summarize:

$$m_{1,1} = \sqrt{\ell_{1,1}^2 + \lambda|v_1|^2} \tag{A.33}$$

$$\boldsymbol{\omega} = \boldsymbol{v}_2 - \frac{v_1}{\ell_{1,1}}\boldsymbol{l}_{2,1} \tag{A.34}$$

$$\boldsymbol{m}_{2,1} = \frac{m_{1,1}}{\ell_{1,1}}\boldsymbol{l}_{2,1} + \frac{\lambda v_1^*}{m_{1,1}}\boldsymbol{\omega} \tag{A.35}$$

$$\boldsymbol{M}_{2,2}\boldsymbol{M}_{2,2}^H = \boldsymbol{L}_{2,2}\boldsymbol{L}_{2,2}^H + \frac{\lambda}{1 + \frac{\lambda|v_1|^2}{\ell_{1,1}^2}}\boldsymbol{\omega}\boldsymbol{\omega}^H \tag{A.36}$$

The rank-1 Hermitian update to a Cholesky decomposition can be computed in quadratic

time. The algorithm can be rewritten to be computed in place.

Algorithm 10: Cholesky Decomposition Hermitian Rank-One Update

$\boldsymbol{\omega} \leftarrow \boldsymbol{v}$

$\alpha \leftarrow 1$

for $i \in \{1, \ldots, C\}$ **do**

  $\eta \leftarrow \alpha \ell_{i,i}^2 + \lambda |\omega_i|^2$

  $m_{i,i} \leftarrow \sqrt{\ell_{i,i}^2 + \frac{\lambda |\omega_i|^2}{\alpha}}$

  for $j \in \{i+1, \ldots, C\}$ **do**

    $\omega_j \leftarrow \omega_j - \frac{\ell_{j,i} \omega_i}{\ell_{i,i}}$

    $m_{j,i} \leftarrow \frac{m_{i,i} \ell_{j,i}}{\ell_{i,i}} + \frac{\lambda m_{i,i} \omega_j \omega_i^*}{\eta}$

  end

  $\alpha \leftarrow \alpha + \frac{\lambda |\omega_i|^2}{\tau^2}$

end

Algorithm 11: Cholesky Decomposition Hermitian Rank-One Update (In Place)

$\boldsymbol{\omega} \leftarrow \boldsymbol{v}$

$\alpha \leftarrow 1$

for $i \in \{1, \ldots, C\}$ **do**

  $\eta \leftarrow \alpha \ell_{i,i}^2 + \lambda |\omega_i|^2$

  $\tau \leftarrow \ell_{i,i}$

  $\ell_{i,i} \leftarrow \sqrt{\ell_{i,i}^2 + \frac{\lambda |\omega_i|^2}{\alpha}}$

  for $j \in \{i+1, \ldots, C\}$ **do**

    $\omega_j \leftarrow \omega_j - \frac{\ell_{j,i} \omega_i}{\tau}$

    $\ell_{j,i} \leftarrow \frac{\ell_{i,i} \ell_{j,i}}{\tau} + \frac{\lambda \ell_{i,i} \omega_j \omega_i^*}{\eta}$

  end

  $\alpha \leftarrow \alpha + \frac{\lambda |\omega_i|^2}{\tau^2}$

end

APPENDIX B

RANK-2 EIGENDECOMPOSITION EDGE CASES

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{v} & \boldsymbol{D}^H \boldsymbol{u} \end{bmatrix} \tag{B.1}$$

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{D}^H \boldsymbol{u} & \boldsymbol{v} \end{bmatrix}^H \tag{B.2}$$

B.1 Less than 2 Independent Eigenvectors

The matrix $\boldsymbol{AB}$ is a Hermitian $M \times M$ matrix, so it has M real eigenvalues and M independent eigenvectors which can be chosen to be orthogonal. Therefore, $\boldsymbol{BA}$ has 2 independent eigenvectors if $\boldsymbol{B}$ is rank 2. There are three cases that can cause $\boldsymbol{B}$ to have a rank less than 2.

1.
$$\boldsymbol{D}^H \boldsymbol{u} = \boldsymbol{0} \tag{B.3}$$

2.
$$\boldsymbol{v} = \boldsymbol{0} \tag{B.4}$$

3.
$$\boldsymbol{D}^H \boldsymbol{u} = \alpha \boldsymbol{v} \tag{B.5}$$

where α is a scalar.

In the first 2 cases, the matrix $\boldsymbol{BA}$ has one independent eigenvector. The first case implies that only the Hermitian update is nonzero. The second case implies that the entire update is zero. (The eigenvalues are zero, so as long as the normalization of eigenvectors is handled with care, it is not necessary to check for these cases in code.)

In the third case, the diagonalization of $\boldsymbol{AB}$ can be determined directly without using the 2×2 matrix $\boldsymbol{BA}$.

$$\boldsymbol{AB} = 2\Re(\alpha)\boldsymbol{v}\boldsymbol{v}^H \tag{B.6}$$

$$\tau = 2\Re(\alpha)\|\boldsymbol{v}\|_2^2 \tag{B.7}$$

$$\boldsymbol{\omega} = \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|_2} \tag{B.8}$$

The rest of the eigenvalues are zero. (The corresponding 2×2 matrix $\boldsymbol{BA}$ shares the same nonzero eigenvalue. The eigenvector that is lost has an eigenvalue of zero, so like in the other 2 cases, it is not necessary to check for this case in code.)

B.2 Eigenvalues are Not Distinct

If the pair of eigenvalues are the same, then all nonzero vectors are eigenvectors of $\boldsymbol{BA}$. However, it is necessary for a diagonalization expansion with only Hermitian terms that the eigenvectors of $\boldsymbol{AB}$ are chosen to be orthogonal, which can be found using a Gram-Schmidt process. This case is not just a theoretical concern; it is necessary to check for this in code.

APPENDIX C

DIFFERENTIATING THE INVERSE FUNCTION

Appendix C considers the equation

$$f_{\hat{\boldsymbol{D}}}(\boldsymbol{b}) = \left(\rho \mathbf{I} + \hat{\boldsymbol{D}}^H \hat{\boldsymbol{D}}\right)^{-1} \boldsymbol{b} \tag{C.1}$$

and derives the mathematical equations necessary to apply the chain rule to composite functions containing function $f_{\hat{\boldsymbol{D}}}$, treating ρ as a real constant.

C.1 Chain Rule

When differentiating composite functions, the chain rule equation is really useful.

$$\frac{\partial g(f(x))}{\partial x} = \frac{\partial g(f(x))}{\partial f(x)} \frac{\partial f(x)}{\partial x} \tag{C.2}$$

If $f_a(x)$ and $g_a(y)$ both depend on real scalar variable a, then

$$\frac{\partial g_a(f_a(x))}{\partial a} = \frac{\partial g_a(y)}{\partial a} + \frac{\partial g_a(f_a(x))}{\partial f_a(x)} \frac{\partial f_a(x)}{\partial a} \tag{C.3}$$

substituting in $y = f_a(x)$ after differentiation. Applying this inductively, the partial derivatives of a composite function can be derived using expressions for the partial derivatives of each of the component functions. So, if $f_a(x)$ is a component function, the partial derivatives that need to be calculated are $\frac{\partial f_a(x)}{\partial a}$ and $\frac{\partial f_a(x)}{\partial x}$.

C.1.1 Matrix Calculus

There are competing standards for matrix calculus notation. This dissertation will default to the numerator layout:

$$\frac{d\boldsymbol{y}}{d\boldsymbol{x}} = \begin{bmatrix} \frac{dy_1}{dx_1} & \cdots & \frac{dy_1}{dx_n} \\ \vdots & \ddots & \vdots \\ \frac{dy_m}{dx_1} & \cdots & \frac{dy_n}{dx_n} \end{bmatrix} \tag{C.4}$$

The corresponding chain rule equations are

$$\frac{\partial g(f(\boldsymbol{b}))}{\partial \boldsymbol{b}} = \frac{\partial g(\boldsymbol{y})}{\partial \boldsymbol{y}} \frac{\partial f(\boldsymbol{b})}{\partial \boldsymbol{b}} \tag{C.5}$$

$$\frac{\partial g_a(f_a(\boldsymbol{b}))}{\partial a} = \frac{\partial g_a(\boldsymbol{y})}{\partial a} + \frac{\partial g_a(\boldsymbol{y})}{\partial \boldsymbol{y}} \frac{\partial f_a(\boldsymbol{b})}{\partial a} \tag{C.6}$$

So, if $f_a(\boldsymbol{b})$ is a component function, its partial derivatives that need to be calculated are $\frac{\partial f_a(\boldsymbol{b})}{\partial a}$ and $\frac{\partial f_a(\boldsymbol{b})}{\partial \boldsymbol{b}}$.

Even for functions with branching, the necessary partial derivatives are the same. In the branching case, derivatives are aggregated just as they were with the shared dependence on scalar variable a. For example,

$$\frac{\partial g(f_1(\boldsymbol{b}), f_2(\boldsymbol{b}))}{\partial \boldsymbol{b}} = \frac{\partial g(f_1(\boldsymbol{b}), f_2(\boldsymbol{b}))}{\partial f_1(\boldsymbol{b})} \frac{\partial f_1(\boldsymbol{b})}{\partial \boldsymbol{b}} + \frac{\partial g(f_1(\boldsymbol{b}), f_2(\boldsymbol{b}))}{\partial f_2(\boldsymbol{b})} \frac{\partial f_2(\boldsymbol{b})}{\partial \boldsymbol{b}} \tag{C.7}$$

If $\mathscr{L}$ is a real-valued scalar function that depends on $\boldsymbol{b} \in \mathbb{R}^M$, then[1]

$$\nabla_{\boldsymbol{b}} \mathscr{L} = \left(\frac{\partial \mathscr{L}}{\partial \boldsymbol{b}} \right)^T \tag{C.8}$$

If $\boldsymbol{b}$ affects $\mathscr{L}$ through multiple branches, the gradient is the sum of multiple terms. I will

[1] The transpose is necessary because the derivatives adhere to numerator layout.

use a superscript to specify one of the terms by its branch:

$$\nabla_{\boldsymbol{b}}^{(\boldsymbol{b}\to f_i(\boldsymbol{b}))}\mathscr{L} = \left(\frac{\partial \mathscr{L}}{\partial f_i(\boldsymbol{b})}\frac{\partial f_i(\boldsymbol{b})}{\partial \boldsymbol{b}}\right)^T \tag{C.9}$$

$$\nabla_{\boldsymbol{b}}\mathscr{L} = \sum_i \nabla_{\boldsymbol{b}}^{(\boldsymbol{b}\to f_i(\boldsymbol{b}))}\mathscr{L} \tag{C.10}$$

C.1.2 Complex Numbers

The chain rule can be applied for functions of complex variables if the functions are analytic. However, for a non-analytic function f, the limit $\lim_{\Delta z\to 0}\frac{f(z+\Delta z)-f(z)}{\Delta z}$ does not exist. Since all non-constant functions onto reals are nonanalytic, the lack of derivative is a problem. Fortunately, for functions onto reals, a chain-rule like property exists for gradients. For non-analytic function $f:\mathbb{Z}^M\to\mathbb{R}$, the gradient is simply[2]

$$\nabla_{\boldsymbol{z}} f(\boldsymbol{z}) = \left(\frac{\partial f(\boldsymbol{z})}{\partial \Re(\boldsymbol{z})} - j\frac{\partial f(\boldsymbol{z})}{\partial \Im(\boldsymbol{z})}\right)^H \tag{C.11}$$

If a non-analytic, real-valued function $g:\mathbb{Z}^N\to\mathbb{R}$ is paired with analytic function $f:\mathbb{Z}^M\to\mathbb{Z}^N$, then the chain rule can safely be applied for composite function $g\circ f$, treating the Hermitian transpose of the gradient as a derivative.

$$(\nabla_{\boldsymbol{z}} g(f(\boldsymbol{z})))^H = \left(\nabla_{f(\boldsymbol{z})} g(f(\boldsymbol{z}))\right)^H \frac{\partial f(\boldsymbol{z})}{\partial \boldsymbol{z}} \tag{C.12}$$

To see that this is true, the key is to split the complex numbers into real and complex components, and apply the chain rule:

$$\frac{\partial g(f(\boldsymbol{z}))}{\partial \Re(\boldsymbol{z})} = \frac{\partial g(f(\boldsymbol{z}))}{\partial \Re(f(\boldsymbol{z}))}\frac{\partial \Re(f(\boldsymbol{z}))}{\partial \Re(\boldsymbol{z})} + \frac{\partial g(f(\boldsymbol{z}))}{\partial \Im(f(\boldsymbol{z}))}\frac{\partial \Im(f(\boldsymbol{z}))}{\partial \Re(\boldsymbol{z})} \tag{C.13}$$

[2]The partial derivatives in this definition are not guaranteed to exist. For those cases, this derivative cannot be calculated. For the rest of this part of the appendix, assume that the non-analytic functions are chosen such that the partial derivatives exist.

By definition,

$$\frac{\partial g(f(\boldsymbol{z}))}{\partial \Re(f(\boldsymbol{z}))} = \Re\left(\left(\nabla_{f(\boldsymbol{z})} g(f(\boldsymbol{z}))\right)^H\right) \tag{C.14}$$

$$\frac{\partial g(f(\boldsymbol{z}))}{\partial \Im(f(\boldsymbol{z}))} = -\Im\left(\left(\nabla_{f(\boldsymbol{z})} g(f(\boldsymbol{z}))\right)^H\right) \tag{C.15}$$

Using the fact that $\frac{\partial f(\boldsymbol{z})}{\partial \boldsymbol{z}} = \lim_{\Delta \boldsymbol{z} \to \mathbf{0}} \frac{f(\boldsymbol{z}+\Delta \boldsymbol{z}) - f(\boldsymbol{z})}{\Delta \boldsymbol{z}}$,

$$\frac{\partial \Re(f(\boldsymbol{z}))}{\partial \Re(\boldsymbol{z})} = \Re\left(\frac{\partial f(\boldsymbol{z})}{\partial \boldsymbol{z}}\right) \tag{C.16}$$

$$\frac{\partial \Im(f(\boldsymbol{z}))}{\partial \Re(\boldsymbol{z})} = \Im\left(\frac{\partial f(\boldsymbol{z})}{\partial \boldsymbol{z}}\right) \tag{C.17}$$

Therefore,

$$\begin{aligned}\frac{\partial g(f(\boldsymbol{z}))}{\partial \Re(\boldsymbol{z})} = \Re\left(\left(\nabla_{f(\boldsymbol{z})} g(f(\boldsymbol{z}))\right)^H\right) \Re\left(\frac{\partial f(\boldsymbol{z})}{\partial \boldsymbol{z}}\right) \\ - \Im\left(\left(\nabla_{f(\boldsymbol{z})} g(f(\boldsymbol{z}))\right)^H\right) \Im\left(\frac{\partial f(\boldsymbol{z})}{\partial \boldsymbol{z}}\right)\end{aligned} \tag{C.18}$$

$$\frac{\partial g(f(\boldsymbol{z}))}{\partial \Re(\boldsymbol{z})} = \Re\left(\left(\nabla_{f(\boldsymbol{z})} g(f(\boldsymbol{z}))\right)^H \frac{\partial f(\boldsymbol{z})}{\partial \boldsymbol{z}}\right) \tag{C.19}$$

Having computed the partial derivative in respect to the real component, it is necessary to now compute the partial derivative in respect to the imaginary component. Following the same process using the chain rule,

$$\frac{\partial g(f(\boldsymbol{z}))}{\partial \Im(\boldsymbol{z})} = \frac{\partial g(f(\boldsymbol{z}))}{\partial \Re(f(\boldsymbol{z}))} \frac{\partial \Re(f(\boldsymbol{z}))}{\partial \Im(\boldsymbol{z})} + \frac{\partial g(f(\boldsymbol{z}))}{\partial \Im(f(\boldsymbol{z}))} \frac{\partial \Im(f(\boldsymbol{z}))}{\partial \Im(\boldsymbol{z})} \tag{C.20}$$

Using the fact that $\frac{\partial f(\boldsymbol{z})}{\partial \boldsymbol{z}} = \lim_{\Delta \boldsymbol{z} \to \mathbf{0}} \frac{f(\boldsymbol{z}+\Delta \boldsymbol{z}) - f(\boldsymbol{z})}{\Delta \boldsymbol{z}}$,

$$\frac{\partial \Re(f(\boldsymbol{z}))}{\partial \Im(\boldsymbol{z})} = -\Im\left(\frac{\partial f(\boldsymbol{z})}{\partial \boldsymbol{z}}\right) \tag{C.21}$$

$$\frac{\partial \Im(f(\boldsymbol{z}))}{\partial \Im(\boldsymbol{z})} = \Re\left(\frac{\partial f(\boldsymbol{z})}{\partial \boldsymbol{z}}\right) \tag{C.22}$$

Therefore,

$$\frac{\partial g(f(\boldsymbol{z}))}{\partial \Im(\boldsymbol{z})} = -\Re\left(\left(\nabla_{f(\boldsymbol{z})} g(f(\boldsymbol{z}))\right)^H\right) \Im\left(\frac{\partial f(\boldsymbol{z})}{\partial \boldsymbol{z}}\right) - \Im\left(\left(\nabla_{f(\boldsymbol{z})} g(f(\boldsymbol{z}))\right)^H\right) \Re\left(\frac{\partial f(\boldsymbol{z})}{\partial \boldsymbol{z}}\right) \tag{C.23}$$

Finally, combining equations C.19 and C.23 demonstrates that the computations using the gradient-based chain rule satisfy the definition in equation C.11.

$$\nabla_{\boldsymbol{z}} g(f(\boldsymbol{z})) = \left(\frac{\partial g(f(\boldsymbol{z}))}{\partial \Re(\boldsymbol{z})} - j\frac{\partial g(f(\boldsymbol{z}))}{\partial \Im(\boldsymbol{z})}\right)^H \tag{C.24}$$

C.2 Partial Derivatives

C.2.1 Partial Derivatives in Respect to Inputs

The function $f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})$ is analytic in respect to $\boldsymbol{b}$, and its partial derivative in respect to $\boldsymbol{b}$ requires no explanation:

$$\frac{\partial f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}{\partial \boldsymbol{b}} = \left(\rho \mathbf{I} + \hat{\boldsymbol{D}}^H \hat{\boldsymbol{D}}\right)^{-1} \tag{C.25}$$

Using the Woodbury matrix identity, equation C.1 can be rewritten as

$$f_{\hat{\boldsymbol{D}}}(\boldsymbol{b}) = \frac{1}{\rho}\left(\boldsymbol{b} - \hat{\boldsymbol{D}}^H \left(\rho \mathbf{I} + \hat{\boldsymbol{D}}\hat{\boldsymbol{D}}^H\right)^{-1} \hat{\boldsymbol{D}}\boldsymbol{b}\right) \tag{C.26}$$

This form contains the function

$$g_{\hat{\boldsymbol{D}}}(\boldsymbol{y}) = \left(\rho \mathbf{I} + \hat{\boldsymbol{D}}\hat{\boldsymbol{D}}^H\right)^{-1} \boldsymbol{y} \tag{C.27}$$

The partial derivative of $g_{\hat{\boldsymbol{D}}}(\boldsymbol{y})$ in respect to $\boldsymbol{y}$ is also straightforward, given its analytic

nature:

$$\frac{\partial g_{\hat{\boldsymbol{D}}}(\boldsymbol{y})}{\partial \boldsymbol{y}} = \left(\rho \mathbf{I} + \hat{\boldsymbol{D}}\hat{\boldsymbol{D}}^H\right)^{-1} \tag{C.28}$$

The function $g_{\hat{\boldsymbol{D}}}$ from the Woodbury form uses a different inverse: $\left(\rho \mathbf{I} + \hat{\boldsymbol{D}}\hat{\boldsymbol{D}}^H\right)^{-1}$. However, the derivations for its derivatives are very similar to that of the previous inverse $\left(\rho \mathbf{I} + \hat{\boldsymbol{D}}^H\hat{\boldsymbol{D}}\right)^{-1}$, so the focus here will remain on the original formulation without the Woodbury transformation.

C.2.2 Partial Derivatives in Respect to Dictionary

Let

$$\boldsymbol{Q} = \rho \mathbf{I} + \hat{\boldsymbol{D}}^H\hat{\boldsymbol{D}} \tag{C.29}$$

so $f_{\hat{\boldsymbol{D}}}(\boldsymbol{b}) = \boldsymbol{Q}^{-1}\boldsymbol{b}$.

According to [50],

$$\frac{\partial(\rho \mathbf{I} + \hat{\boldsymbol{D}}^H\hat{\boldsymbol{D}})^{-1}}{\partial a} = -\boldsymbol{Q}^{-1}\frac{\partial \boldsymbol{Q}}{\partial a}\boldsymbol{Q}^{-1} \tag{C.30}$$

where a is a real scalar variable.

$$\frac{\partial \boldsymbol{Q}}{\partial a} = \frac{\partial \rho \mathbf{I}}{\partial a} + \frac{\partial \hat{\boldsymbol{D}}^H\hat{\boldsymbol{D}}}{\partial a} \tag{C.31}$$

$$\left(\hat{\boldsymbol{D}}^H\hat{\boldsymbol{D}}\right)_{m_1,m_2} = \sum_i \hat{D}_{i,m_2}\hat{D}^*_{i,m_1} \tag{C.32}$$

where $\hat{D}^*_{i,m_1}$ is the complex conjugate of $\hat{D}_{i,m_1}$. For, $m_1 \neq m_2$:

$$\begin{aligned}
\frac{\partial \left(\hat{\boldsymbol{D}}^H \hat{\boldsymbol{D}}\right)_{m_1,m_2}}{\partial \Re(\hat{D}_{i,m_1})} &= \hat{D}_{i,m_2} \\
\frac{\partial \left(\hat{\boldsymbol{D}}^H \hat{\boldsymbol{D}}\right)_{m_1,m_2}}{\partial \Im(\hat{D}_{i,m_1})} &= -j\hat{D}_{i,m_2} \\
\frac{\partial \left(\hat{\boldsymbol{D}}^H \hat{\boldsymbol{D}}\right)_{m_1,m_2}}{\partial \Re(\hat{D}_{i,m_2})} &= \hat{D}^*_{i,m_1} \\
\frac{\partial \left(\hat{\boldsymbol{D}}^H \hat{\boldsymbol{D}}\right)_{m_1,m_2}}{\partial \Im(\hat{D}_{i,m_2})} &= j\hat{D}^*_{i,m_1}
\end{aligned} \tag{C.33}$$

For the case $m_1 = m_2$:

$$\begin{aligned}
\frac{\partial \left(\hat{\boldsymbol{D}}^H \hat{\boldsymbol{D}}\right)_{m_1,m_1}}{\partial \Re(\hat{D}_{i,m_1})} &= 2\Re(\hat{D}_{i,m_1}) \\
\frac{\partial \left(\hat{\boldsymbol{D}}^H \hat{\boldsymbol{D}}\right)_{m_1,m_1}}{\partial \Im(\hat{D}_{i,m_1})} &= 2\Im(\hat{D}_{i,m_1})
\end{aligned} \tag{C.34}$$

Combining equations C.33 and C.34 yields the following equations:

$$\begin{aligned}
\frac{\partial \left(\hat{\boldsymbol{D}}^H \hat{\boldsymbol{D}}\right)}{\partial \Re(\hat{D}_{c,m})} &= \boldsymbol{e}_m \boldsymbol{e}_c^T \hat{\boldsymbol{D}} + \hat{\boldsymbol{D}}^H \boldsymbol{e}_c \boldsymbol{e}_m^T \\
\frac{\partial \left(\hat{\boldsymbol{D}}^H \hat{\boldsymbol{D}}\right)}{\partial \Im(\hat{D}_{c,m})} &= -j\boldsymbol{e}_m \boldsymbol{e}_c^T \hat{\boldsymbol{D}} + j\hat{\boldsymbol{D}}^H \boldsymbol{e}_c \boldsymbol{e}_m^T
\end{aligned} \tag{C.35}$$

where $\boldsymbol{e}_i$ is the ith Euclidean basis vector.[3]

$$\frac{\partial \rho \mathbf{I}}{\partial \hat{D}_{c,m}} = 0 \tag{C.36}$$

[3]That is, the ith element of vector $\boldsymbol{e}_i$ is equal to one, and all other elements are zero.

So,

$$\begin{aligned}\frac{\partial \boldsymbol{Q}}{\partial \Re(\hat{D}_{c,m})} &= \boldsymbol{e}_m \boldsymbol{e}_c^T \hat{\boldsymbol{D}} + \hat{\boldsymbol{D}}^H \boldsymbol{e}_c \boldsymbol{e}_m^T \\ \frac{\partial \boldsymbol{Q}}{\partial \Im(\hat{D}_{c,m})} &= -j\boldsymbol{e}_m \boldsymbol{e}_c^T \hat{\boldsymbol{D}} + j\hat{\boldsymbol{D}}^H \boldsymbol{e}_c \boldsymbol{e}_m^T\end{aligned} \tag{C.37}$$

Finally, this result can be plugged back into equation C.30

$$\begin{aligned}\frac{\partial (\rho \mathbf{I} + \hat{\boldsymbol{D}}^H \hat{\boldsymbol{D}})^{-1}}{\partial \Re(\hat{D}_{c,m})} &= -\boldsymbol{Q}^{-1}(\boldsymbol{e}_m \boldsymbol{e}_c^T \hat{\boldsymbol{D}} + \hat{\boldsymbol{D}}^H \boldsymbol{e}_c \boldsymbol{e}_m^T)\boldsymbol{Q}^{-1} \\ \frac{\partial (\rho \mathbf{I} + \hat{\boldsymbol{D}}^H \hat{\boldsymbol{D}})^{-1}}{\partial \Im(\hat{D}_{c,m})} &= \boldsymbol{Q}^{-1}(j\boldsymbol{e}_m \boldsymbol{e}_c^T \hat{\boldsymbol{D}} - j\hat{\boldsymbol{D}}^H \boldsymbol{e}_c \boldsymbol{e}_m^T)\boldsymbol{Q}^{-1}\end{aligned} \tag{C.38}$$

$$\begin{aligned}\frac{\partial f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}{\partial \Re(\hat{D}_{c,m})} &= \boldsymbol{Q}^{-1}(-\boldsymbol{e}_m \boldsymbol{e}_c^T \hat{\boldsymbol{D}} - \hat{\boldsymbol{D}}^H \boldsymbol{e}_c \boldsymbol{e}_m^T)\boldsymbol{Q}^{-1}\boldsymbol{b} \\ \frac{\partial f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}{\partial \Im(\hat{D}_{c,m})} &= \boldsymbol{Q}^{-1}(j\boldsymbol{e}_m \boldsymbol{e}_c^T \hat{\boldsymbol{D}} - j\hat{\boldsymbol{D}}^H \boldsymbol{e}_c \boldsymbol{e}_m^T)\boldsymbol{Q}^{-1}\boldsymbol{b}\end{aligned} \tag{C.39}$$

The derivations for the partial derivatives of

$$g_{\hat{\boldsymbol{D}}}(\boldsymbol{y}) = (\rho \mathbf{I} + \boldsymbol{D}\boldsymbol{D}^H)^{-1}\boldsymbol{y} \tag{C.40}$$

are very similar to that of $f_{\hat{\boldsymbol{D}}}$. Let $\boldsymbol{Q}_g = \rho \mathbf{I} + \boldsymbol{D}\boldsymbol{D}^H$. Given the similarity between the derivations, the calculations will not be repeated here, though through the same procedure as before,

$$\begin{aligned}\frac{\partial \boldsymbol{Q}_g}{\partial \Re(\hat{D}_{c,m})} &= \boldsymbol{e}_c \boldsymbol{e}_m^T \hat{\boldsymbol{D}}^H + \hat{\boldsymbol{D}} \boldsymbol{e}_m \boldsymbol{e}_c^T \\ \frac{\partial \boldsymbol{Q}_g}{\partial \Im(\hat{D}_{c,m})} &= j\boldsymbol{e}_c \boldsymbol{e}_m^T \hat{\boldsymbol{D}}^H - j\hat{\boldsymbol{D}} \boldsymbol{e}_m \boldsymbol{e}_c^T\end{aligned} \tag{C.41}$$

$$\begin{aligned}\frac{\partial(\rho\mathbf{I}+\hat{\boldsymbol{D}}\hat{\boldsymbol{D}}^H)^{-1}}{\partial\Re(\hat{D}_{c,m})} &= -\boldsymbol{Q}_g^{-1}(\boldsymbol{e}_c\boldsymbol{e}_m^T\hat{\boldsymbol{D}}^H+\hat{\boldsymbol{D}}\boldsymbol{e}_m\boldsymbol{e}_c^T)\boldsymbol{Q}_g^{-1}\\ \frac{\partial(\rho\mathbf{I}+\hat{\boldsymbol{D}}\hat{\boldsymbol{D}}^H)^{-1}}{\partial\Im(\hat{D}_{c,m})} &= -\boldsymbol{Q}_g^{-1}(j\boldsymbol{e}_c\boldsymbol{e}_m^T\hat{\boldsymbol{D}}^H-j\hat{\boldsymbol{D}}\boldsymbol{e}_m\boldsymbol{e}_c^T)\boldsymbol{Q}_g^{-1}\end{aligned} \tag{C.42}$$

C.3 Backpropagation of Loss Function

To backpropagate the loss function $\mathscr{L}$ through component function $f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})$, the partial derivative in respect to the input of function $f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})$ is necessary to reach earlier layers, as explained in section C.1. If $\boldsymbol{b}$ only influenced $\mathscr{L}$ through $f_{\hat{\boldsymbol{D}}}$, then

$$\nabla_{\boldsymbol{b}}\mathscr{L} = \left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \frac{\partial f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}{\partial \boldsymbol{b}} \tag{C.43}$$

If the dictionary $\hat{\boldsymbol{D}}$ is only used in component function $f_{\hat{\boldsymbol{D}}}$, then[4]

$$\frac{\partial\mathscr{L}}{\partial\Re(\hat{D}_{c,m})} = \Re\left(\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \frac{\partial f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}{\partial\Re(\hat{D}_{c,m})}\right) \tag{C.44}$$

$$\frac{\partial\mathscr{L}}{\partial\Im(\hat{D}_{c,m})} = \Re\left(\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \frac{\partial f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}{\partial\Im(\hat{D}_{c,m})}\right) \tag{C.45}$$

If the dictionary $\hat{\boldsymbol{D}}$ is used in multiple component functions, the expressions $\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \frac{\partial f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}{\partial\Re(\hat{D}_{c,m})}$ and $\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \frac{\partial f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}{\partial\Im(\hat{D}_{c,m})}$ still must be calculated so that the results can be aggregated later. Similarly, if $\boldsymbol{b}$ affects $\mathscr{L}$ through multiple branches,[5] the expression $\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \frac{\partial f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}{\partial \boldsymbol{b}}$ still must be calculated, and $\nabla_{\boldsymbol{b}}\mathscr{L}$ is just an aggregation of

[4] These equations can be derived through chain rule. Note that the function of $\hat{\boldsymbol{D}}_{c,m}$ is non-analytic, so to apply chain rule properly, it is important to work with the real and imaginary components, rather than directly with complex numbers.

[5] That is, not just through function $f_{\hat{\boldsymbol{D}}}$

the results across all branches.

$$\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \frac{\partial f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}{\partial \boldsymbol{b}} = \left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \boldsymbol{Q}^{-1} \tag{C.46}$$

$$\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \frac{\partial f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}{\partial \Re(\hat{D}_{c,m})} = \left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \boldsymbol{Q}^{-1}(-\boldsymbol{e}_m\boldsymbol{e}_c^T\hat{\boldsymbol{D}} - \hat{\boldsymbol{D}}^H\boldsymbol{e}_c\boldsymbol{e}_m^T)\boldsymbol{Q}^{-1}\boldsymbol{b} \tag{C.47}$$

$$\begin{aligned}\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \frac{\partial f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}{\partial \Re(\hat{D}_{c,m})} = &-\left(\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \boldsymbol{Q}^{-1}\boldsymbol{e}_m\right)(\boldsymbol{e}_c^T\hat{\boldsymbol{D}}\boldsymbol{Q}^{-1}\boldsymbol{b}) \\ &-\left(\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \boldsymbol{Q}^{-1}\hat{\boldsymbol{D}}^H\boldsymbol{e}_c\right)(\boldsymbol{e}_m^T\boldsymbol{Q}^{-1}\boldsymbol{b})\end{aligned} \tag{C.48}$$

Note that $\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \boldsymbol{Q}^{-1}\hat{\boldsymbol{D}}^H\boldsymbol{e}_c$ and $\boldsymbol{e}_m^T\boldsymbol{Q}^{-1}\boldsymbol{b}$ are scalars and $\boldsymbol{Q}^{-1}$ is Hermitian, so using the fact that the Hermitian transpose of a scalar is its complex conjugate

$$\begin{aligned}\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \boldsymbol{Q}^{-1}\hat{\boldsymbol{D}}^H\boldsymbol{e}_c &= \left(\boldsymbol{e}_c^T\hat{\boldsymbol{D}}\boldsymbol{Q}^{-1}\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^* \\ \boldsymbol{e}_m^T\boldsymbol{Q}^{-1}\boldsymbol{b} &= (\boldsymbol{b}^H\boldsymbol{Q}^{-1}\boldsymbol{e}_m)^*\end{aligned} \tag{C.49}$$

Therefore,

$$\begin{aligned}\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \frac{\partial f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}{\partial \Re(\hat{D}_{c,m})} = &-\left(\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \boldsymbol{Q}^{-1}\boldsymbol{e}_m\right)(\boldsymbol{e}_c^T\hat{\boldsymbol{D}}\boldsymbol{Q}^{-1}\boldsymbol{b}) \\ &-\left(\boldsymbol{e}_c^T\hat{\boldsymbol{D}}\boldsymbol{Q}^{-1}\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^*(\boldsymbol{b}^H\boldsymbol{Q}^{-1}\boldsymbol{e}_m)^*\end{aligned} \tag{C.50}$$

Scalar multiplication commutes, so

$$\begin{aligned}\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \frac{\partial f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}{\partial \Re(\hat{D}_{c,m})} = &-(\boldsymbol{e}_c^T\hat{\boldsymbol{D}}\boldsymbol{Q}^{-1}\boldsymbol{b})\left(\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \boldsymbol{Q}^{-1}\boldsymbol{e}_m\right) \\ &-\left(\boldsymbol{e}_c^T\hat{\boldsymbol{D}}\boldsymbol{Q}^{-1}\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^*(\boldsymbol{b}^H\boldsymbol{Q}^{-1}\boldsymbol{e}_m)^*\end{aligned} \tag{C.51}$$

$$\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \frac{\partial f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}{\partial \Re(\hat{D}_{c,m})} = \boldsymbol{e}_c^T(-(\hat{\boldsymbol{D}}\boldsymbol{Q}^{-1}\boldsymbol{b})\left(\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \boldsymbol{Q}^{-1}\right) - \left(\hat{\boldsymbol{D}}\boldsymbol{Q}^{-1}\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^* (\boldsymbol{b}^H\boldsymbol{Q}^{-1})^*)\boldsymbol{e}_m \quad \text{(C.52)}$$

$$\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \frac{\partial f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}{\partial \Re(\hat{D}_{c,m})} = -\left((\hat{\boldsymbol{D}}\boldsymbol{Q}^{-1}\boldsymbol{b})\left(\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \boldsymbol{Q}^{-1}\right)\right)_{c,m} - \left(\left(\hat{\boldsymbol{D}}\boldsymbol{Q}^{-1}\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^* (\boldsymbol{b}^H\boldsymbol{Q}^{-1})^*\right)_{c,m} \quad \text{(C.53)}$$

$$\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \frac{\partial f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}{\partial \Im(\hat{D}_{c,m})} = \left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \boldsymbol{Q}^{-1}(j\boldsymbol{e}_m\boldsymbol{e}_c^T\hat{\boldsymbol{D}} - j\hat{\boldsymbol{D}}^H\boldsymbol{e}_c\boldsymbol{e}_m^T)\boldsymbol{Q}^{-1}\boldsymbol{b} \quad \text{(C.54)}$$

$$\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \frac{\partial f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}{\partial \Im(\hat{D}_{c,m})} = j\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \boldsymbol{Q}^{-1}\boldsymbol{e}_m\boldsymbol{e}_c^T\hat{\boldsymbol{D}}\boldsymbol{Q}^{-1}\boldsymbol{b} - j\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \boldsymbol{Q}^{-1}\hat{\boldsymbol{D}}^H\boldsymbol{e}_c\boldsymbol{e}_m^T\boldsymbol{Q}^{-1}\boldsymbol{b} \quad \text{(C.55)}$$

$$\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \frac{\partial f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}{\partial \Im(\hat{D}_{c,m})} = j\left(\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \boldsymbol{Q}^{-1}\boldsymbol{e}_m\right)(\boldsymbol{e}_c^T\hat{\boldsymbol{D}}\boldsymbol{Q}^{-1}\boldsymbol{b}) - j\left(\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \boldsymbol{Q}^{-1}\hat{\boldsymbol{D}}^H\boldsymbol{e}_c\right)(\boldsymbol{e}_m^T\boldsymbol{Q}^{-1}\boldsymbol{b}) \quad \text{(C.56)}$$

$$\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \frac{\partial f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}{\partial \Im(\hat{D}_{c,m})} = j(\boldsymbol{e}_c^T\hat{\boldsymbol{D}}\boldsymbol{Q}^{-1}\boldsymbol{b})\left(\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \boldsymbol{Q}^{-1}\boldsymbol{e}_m\right) - j\left(\boldsymbol{e}_c^T\boldsymbol{D}\boldsymbol{Q}^{-1}\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^* (\boldsymbol{b}^H\boldsymbol{Q}^{-1}\boldsymbol{e}_m)^* \quad \text{(C.57)}$$

$$\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \frac{\partial f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}{\partial \Im(\hat{D}_{c,m})} = -j\left(\left(\boldsymbol{D}\boldsymbol{Q}^{-1}\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^* (\boldsymbol{b}^H\boldsymbol{Q}^{-1})^*\right)_{c,m} + j\left((\hat{\boldsymbol{D}}\boldsymbol{Q}^{-1}\boldsymbol{b})\left(\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \boldsymbol{Q}^{-1}\right)\right)_{c,m} \quad \text{(C.58)}$$

Assuming that $\hat{D}_{c,m}$ only affects $\mathscr{L}$ through $f_{\boldsymbol{D}}(\boldsymbol{b})$,

$$\frac{\partial \mathscr{L}}{\partial \Re(\hat{D}_{c,m})} = -\Re\left(\left(\hat{\boldsymbol{D}}\boldsymbol{Q}^{-1}\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)(\boldsymbol{b}^H\boldsymbol{Q}^{-1}) + (\hat{\boldsymbol{D}}\boldsymbol{Q}^{-1}\boldsymbol{b})\left(\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \boldsymbol{Q}^{-1}\right)\right)_{c,m} \quad \text{(C.59)}$$

$$\frac{\partial \mathscr{L}}{\partial \Im(\hat{D}_{c,m})} = \Re\left(j(\hat{\boldsymbol{D}}\boldsymbol{Q}^{-1}\boldsymbol{b})\left(\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \boldsymbol{Q}^{-1}\right) - j\left(\boldsymbol{D}\boldsymbol{Q}^{-1}\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^* (\boldsymbol{b}^H\boldsymbol{Q}^{-1})^*\right)_{c,m} \quad \text{(C.60)}$$

$$\frac{\partial \mathscr{L}}{\partial \Im(\hat{D}_{c,m})} = \Im\left(-\left(\boldsymbol{D}\boldsymbol{Q}^{-1}\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)(\boldsymbol{b}^H\boldsymbol{Q}^{-1}) - (\hat{\boldsymbol{D}}\boldsymbol{Q}^{-1}\boldsymbol{b})\left(\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \boldsymbol{Q}^{-1}\right)\right)_{c,m} \quad \text{(C.61)}$$

Therefore,

$$\frac{\partial \mathscr{L}}{\partial \hat{D}_{c,m}} = \left(-(\hat{\boldsymbol{D}}\boldsymbol{Q}^{-1}\boldsymbol{b})^*\left(\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \boldsymbol{Q}^{-1}\right)^* - \left(\hat{\boldsymbol{D}}\boldsymbol{Q}^{-1}\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^* (\boldsymbol{b}^H\boldsymbol{Q}^{-1})^*\right)_{c,m} \quad \text{(C.62)}$$

$$\nabla_{\hat{\boldsymbol{D}}}\mathscr{L} = -(\hat{\boldsymbol{D}}\boldsymbol{Q}^{-1}\boldsymbol{b})\left(\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^H \boldsymbol{Q}^{-1}\right) - \left(\hat{\boldsymbol{D}}\boldsymbol{Q}^{-1}\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)(\boldsymbol{b}^H\boldsymbol{Q}^{-1}) \quad \text{(C.63)}$$

If $\hat{\boldsymbol{D}}$ affects $\mathscr{L}$ through other functions (rather than just through $f_{\boldsymbol{D}}(\boldsymbol{b})$),

$$\nabla_{\hat{\boldsymbol{D}}}^{(\boldsymbol{b}\to f_{\boldsymbol{D}}(\boldsymbol{b}))}\mathscr{L} = -(\hat{\boldsymbol{D}}\boldsymbol{Q}^{-1}\boldsymbol{b})\left(\left(\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)^{H}\boldsymbol{Q}^{-1}\right) - \left(\hat{\boldsymbol{D}}\boldsymbol{Q}^{-1}\nabla_{f_{\hat{\boldsymbol{D}}}(\boldsymbol{b})}\mathscr{L}\right)(\boldsymbol{b}^{H}\boldsymbol{Q}^{-1}) \tag{C.64}$$

Using similar derivations for the function $g_{\hat{\boldsymbol{D}}}(\boldsymbol{y})$ yields the following equation for the gradient term:

$$\nabla_{\hat{\boldsymbol{D}}}^{(\boldsymbol{y}\to g_{\boldsymbol{D}}(\boldsymbol{y}))}\mathscr{L} = -\left(\boldsymbol{Q}_g^{-1}\nabla_{g_{\hat{\boldsymbol{D}}}(\boldsymbol{y})}\mathscr{L}\right)(\boldsymbol{y}^{H}\boldsymbol{Q}_g^{-1}\hat{\boldsymbol{D}}) - (\boldsymbol{Q}_g^{-1}\boldsymbol{y})\left(\left(\nabla_{g_{\hat{\boldsymbol{D}}}(\boldsymbol{y})}\mathscr{L}\right)^{H}\boldsymbol{Q}_g^{-1}\hat{\boldsymbol{D}}\right) \tag{C.65}$$

REFERENCES

[1] B. Wohlberg, "Convolutional sparse representations as an image model for impulse noise restoration," in *2016 IEEE 12th Image, Video, and Multidimensional Signal Processing Workshop (IVMSP)*, IEEE, 2016, pp. 1–5.

[2] S. Kong and D. Wang, "A dictionary learning approach for classification: Separating the particularity and the commonality," in *European conference on computer vision*, Springer, 2012, pp. 186–199.

[3] B. T. Carroll, B. M. Whitaker, W. Dayley, and D. V. Anderson, "Outlier learning via augmented frozen dictionaries," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 25, no. 6, pp. 1207–1215, 2017.

[4] G. Polatkan, M. Zhou, L. Carin, D. Blei, and I. Daubechies, "A bayesian nonparametric approach to image super-resolution," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 2, pp. 346–358, 2014.

[5] S. Gu, W. Zuo, Q. Xie, D. Meng, X. Feng, and L. Zhang, "Convolutional sparse coding for image super-resolution," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1823–1831.

[6] V. Papyan, Y. Romano, and M. Elad, "Convolutional neural networks analyzed via convolutional sparse coding," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 2887–2938, 2017.

[7] Y. Zhu and S. Lucey, "Convolutional sparse coding for trajectory reconstruction," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 3, pp. 529–540, 2013.

[8] N. Chodosh and S. Lucey, "When to use convolutional neural networks for inverse problems," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 8226–8235.

[9] B. Chen, G. Polatkan, G. Sapiro, D. Blei, D. Dunson, and L. Carin, "Deep learning with hierarchical convolutional factor analysis," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1887–1901, 2013.

[10] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, "Deconvolutional networks," in *2010 IEEE Computer Society Conference on computer vision and pattern recognition*, IEEE, 2010, pp. 2528–2535.

[11] Y. Pu, X. Yuan, and L. Carin, "Bayesian deep deconvolutional learning," *stat*, vol. 1050, p. 18, 2014.

[12] C. Murdock, M. Chang, and S. Lucey, "Deep component analysis via alternating direction neural networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 820–836.

[13] N. Chodosh, C. Wang, and S. Lucey, "Deep convolutional compressed sensing for lidar depth completion," in *Asian Conference on Computer Vision*, Springer, 2018, pp. 499–513.

[14] M. F. Baumgardner, L. L. Biehl, and D. A. Landgrebe, *220 band aviris hyperspectral image data set: June 12, 1992 indian pine test site 3*, 2015.

[15] C. Garcia-Cardona and B. Wohlberg, "Convolutional dictionary learning for multi-channel signals," in *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, 2018, pp. 335–342.

[16] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[17] J. Eckstein, "Parallel alternating direction multiplier decomposition of convex programs," *Journal of Optimization Theory and Applications*, vol. 80, no. 1, pp. 39–62, 1994.

[18] R. Nishihara, L. Lessard, B. Recht, A. Packard, and M. Jordan, "A general analysis of the convergence of admm," in *International Conference on Machine Learning*, PMLR, 2015, pp. 343–352.

[19] H. Bristow, A. Eriksson, and S. Lucey, "Fast convolutional sparse coding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 391–398.

[20] M. Šorel and F. Šroubek, "Fast convolutional sparse coding using matrix inversion lemma," *Digital Signal Processing*, vol. 55, pp. 44–51, 2016.

[21] F. Heide, W. Heidrich, and G. Wetzstein, "Fast and flexible convolutional sparse coding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 5135–5143.

[22] B. Wohlberg, "Efficient algorithms for convolutional sparse representations," *IEEE Transactions on Image Processing*, vol. 25, no. 1, pp. 301–315, 2015.

[23] H. V. Henderson and S. R. Searle, "On deriving the inverse of a sum of matrices," *Siam Review*, vol. 23, no. 1, pp. 53–60, 1981.

[24] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM journal on imaging sciences*, vol. 2, no. 1, pp. 183–202, 2009.

[25] Y. Xu and W. Yin, "A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion," *SIAM Journal on imaging sciences*, vol. 6, no. 3, pp. 1758–1789, 2013.

[26] M. Aharon, M. Elad, and A. Bruckstein, "K-svd: An algorithm for designing overcomplete dictionaries for sparse representation," *IEEE Transactions on signal processing*, vol. 54, no. 11, pp. 4311–4322, 2006.

[27] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[28] A. Rangamani, A. Mukherjee, A. Basu, A. Arora, T. Ganapathi, S. Chin, and T. D. Tran, "Sparse coding and autoencoders," in *2018 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2018, pp. 36–40.

[29] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories," in *2004 conference on computer vision and pattern recognition workshop*, IEEE, 2004, pp. 178–178.

[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[31] B. Chen, G. Polatkan, G. Sapiro, L. Carin, and D. B. Dunson, "The hierarchical beta process for convolutional factor analysis and deep learning," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 361–368.

[32] Y. Pu, X. Yuan, and L. Carin, "Generative deep deconvolutional learning," *arXiv preprint arXiv:1412.6039*, 2014.

[33] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*, PMLR, 2013, pp. 1139–1147.

[34] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: `1412.6980 [cs.LG]`.

[35] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[36] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015.

[37] C. Garcia-Cardona and B. Wohlberg, "Convolutional dictionary learning: A comparative review and new algorithms," *IEEE Transactions on Computational Imaging*, vol. 4, no. 3, pp. 366–381, 2018.

[38] M. Šorel and M. Bartoš, "Efficient jpeg decompression by the alternating direction method of multipliers," in *2016 23rd International Conference on Pattern Recognition (ICPR)*, IEEE, 2016, pp. 271–276.

[39] B. Wohlberg, "Boundary handling for convolutional sparse representations," in *2016 IEEE International Conference on Image Processing (ICIP)*, 2016, pp. 1833–1837.

[40] B. Wohlberg and P. Rodríguez, "Convolutional sparse coding: Boundary handling revisited," *CoRR*, vol. abs/1707.06718, 2017. arXiv: 1707.06718.

[41] F. Chollet *et al.*, *Keras*, https://keras.io, 2015.

[42] J. V. Dillon, I. Langmore, D. Tran, E. Brevdo, S. Vasudevan, D. Moore, B. Patton, A. Alemi, M. Hoffman, and R. A. Saurous, "Tensorflow distributions," *arXiv preprint arXiv:1711.10604*, 2017.

[43] O. Krause and C. Igel, "A more efficient rank-one covariance matrix update for evolution strategies," in *Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII*, 2015, pp. 129–136.

[44] B. He, H. Yang, and S. Wang, "Alternating direction method with self-adaptive penalty parameters for monotone variational inequalities," *Journal of Optimization Theory and applications*, vol. 106, no. 2, pp. 337–356, 2000.

[45] Z. Xu, M. Figueiredo, and T. Goldstein, "Adaptive admm with spectral penalty parameter selection," in *Artificial Intelligence and Statistics*, PMLR, 2017, pp. 718–727.

[46] M. Elad and M. Aharon, "Image denoising via sparse and redundant representations over learned dictionaries," *IEEE Transactions on Image processing*, vol. 15, no. 12, pp. 3736–3745, 2006.

[47] J. Yang, J. Wright, T. S. Huang, and Y. Ma, "Image super-resolution via sparse representation," *IEEE transactions on image processing*, vol. 19, no. 11, pp. 2861–2873, 2010.

[48] P. Turquais, E. G. Asgedom, and W. Söllner, "A method of combining coherence-constrained sparse coding and dictionary learning for denoising," *Geophysics*, vol. 82, no. 3, pp. V137–V148, 2017.

[49] F. Jiang, Z. Chen, A. Nazir, W. Shi, W. Lim, S. Liu, and S. Rho, "Combining fields of experts (foe) and k-svd methods in pursuing natural image priors," *Journal of Visual Communication and Image Representation*, vol. 78, p. 103 142, 2021.

[50] K. B. Petersen, M. S. Pedersen, *et al.*, "The matrix cookbook," *Technical University of Denmark*, vol. 7, no. 15, p. 510, 2008.