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

Month 2021

MULTI-LAYER DICTIONARY LEARNING USING LOW-RANK UPDATES

ACKNOWLEDGEMENTS

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

CHAPTER 1

INTRODUCTION

Dictionaries and Dictionary Learning

Convolutional Dictionaries

Convolutional Neural Networks

Multi-Layer Dictionaries

Contributions and Organization of Dissertation

CHAPTER 2

LEARNING DICTIONARIES FOR MULTI-CHANNEL SIGNALS

Introduction

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. Much of the literature on using convolutional dictionaries is tailored to applications with signals that only have a small number of channels. This chapter presents a novel method to use learning convolutional dictionaries for multi-channel signals.

Dictionary Types

There are many ways to construct a convolutional sparse representation of a multi-channel signal, but broadly the distinctions reduce down to if and how signal channels share dictionaries and coefficients, and if and how those non-shared entities interact across channels.

It is common in many applications for dictionary models to share dictionaries across channels, which requires the use multi-channel coefficients. If such models were used in a multi-layer dictionary model, the tensor rank would increase with each subsequent layer.

For this work, I focus instead on the multi-channel dictionary with shared coefficients. This structure matches that of convolutional neural networks, and the number of channels for a subsequent dictionary is the number of filters for the dictionary from the previous layer.

Pursuit and Sparse Coding

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

$$\boldsymbol{s}_i \approx \boldsymbol{D}\boldsymbol{x}_i \tag{2.1}$$

(Here the subscript i specifies a particular signal and its corresponding coefficients.) A pursuit algorithm finds the coefficients $\boldsymbol{x}_i$ corresponding to a particular signal $\boldsymbol{s}_i$ 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 $\boldsymbol{x}_i$ represent $\boldsymbol{s}_i$ equally well (at least in an L2 sense). Researchers and practitioners commonly either impose a sparsity constraint on the coefficients or add a coefficient L1 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 L1 penalty, the pursuit optimization problem looks like this:

$$\boldsymbol{x}_i = \arg\min_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{s}_i - \boldsymbol{D}\boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{x}\|_1 \tag{2.2}$$

where λ is a nonnegative hyperparameter controlling how much the L1 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.

ADMM

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

$$\begin{aligned} &\underset{\boldsymbol{x},\boldsymbol{y}}{\text{minimize}} f(\boldsymbol{x}) + g(\boldsymbol{y}) \\ &\text{subject to } \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{y} + \boldsymbol{c} = \boldsymbol{0} \end{aligned} \tag{2.3}$$

where f and g are convex functions [1]. (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:

$$\mathrm{L}_\rho(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}) = f(\boldsymbol{x}) + g(\boldsymbol{y}) + \boldsymbol{u}^H(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{y} + \boldsymbol{c}) + \frac{\rho}{2}\|\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{y} + \boldsymbol{c}\|_2^2 \tag{2.4}$$

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 $\boldsymbol{u}$:

$$\boldsymbol{x}^{(t+1)} = \arg\min_{\boldsymbol{x}} \mathrm{L}_\rho(\boldsymbol{x}, \boldsymbol{y}^{(t)}, \boldsymbol{u}^{(t)}) \tag{2.5}$$

$$\boldsymbol{y}^{(t+1)} = \arg\min_{\boldsymbol{y}} \mathrm{L}_\rho(\boldsymbol{x}^{(t+1)}, \boldsymbol{y}, \boldsymbol{u}^{(t)}) \tag{2.6}$$

$$\boldsymbol{u}^{(t+1)} = \boldsymbol{u}^{(t)} + \rho(\boldsymbol{A}\boldsymbol{x}^{(t+1)} + \boldsymbol{B}\boldsymbol{y}^{(t+1)} + \boldsymbol{c}) \tag{2.7}$$

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 stepsize 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.[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:

$$\mathrm{L}_\rho(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}) = f(\boldsymbol{x}) + g(\boldsymbol{y}) + \frac{\rho}{2}\|\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{y} + \boldsymbol{c} + \frac{\boldsymbol{u}}{\rho}\|_2^2 - \frac{\rho}{2}\|\frac{\boldsymbol{u}}{\rho}\|_2^2 \tag{2.8}$$

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 $\frac{\boldsymbol{u}}{\rho}$ 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 2.7.

$$\frac{\boldsymbol{u}^{(t+1)}}{\rho} = \frac{\boldsymbol{u}^{(t)}}{\rho} + \boldsymbol{A}\boldsymbol{x}^{(t+1)} + \boldsymbol{B}\boldsymbol{y}^{(t+1)} + \boldsymbol{c} \tag{2.9}$$

This form is known as scaled ADMM.

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

$$\boldsymbol{x}^{(t+1)} = \arg\min_{\boldsymbol{x}} \mathrm{L}_\rho(\boldsymbol{x}, \boldsymbol{y}^{(t)}, \boldsymbol{u}^{(t)}) \tag{2.10}$$

$$\boldsymbol{u}^{(t+\frac{1}{2})} = \boldsymbol{u}^{(t)} + (\alpha - 1)\rho(\boldsymbol{A}\boldsymbol{x}^{(t+1)} + \boldsymbol{B}\boldsymbol{y}^{(t)} + \boldsymbol{c}) \tag{2.11}$$

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

$$\boldsymbol{y}^{(t+1)} = \arg\min_{\boldsymbol{y}} \mathrm{L}_\rho(\boldsymbol{x}^{(t+1)}, \boldsymbol{y}, \boldsymbol{u}^{(t+\frac{1}{2})}) \tag{2.12}$$

$$\boldsymbol{u}^{(t+1)} = \boldsymbol{u}^{(t+\frac{1}{2})} + \rho(\boldsymbol{A}\boldsymbol{x}^{(t+1)} + \boldsymbol{B}\boldsymbol{y}^{(t+1)} + \boldsymbol{c}) \tag{2.13}$$

When $\alpha > 1$, this is known as over-relaxation, and if $\alpha < 1$, this is known as under-relaxation.[2] α is always chozen to be greater than zero. In some applications, researchers have found using over-relaxation converges faster than without over-relaxation [2], but optimal choice of α is problem-dependent [3].

Applying ADMM to the Sparse Coding Problem

Recall from section 2.3, equation 2.2 for sparse coding.

$$\boldsymbol{x}_i = \arg\min_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{s}_i - \boldsymbol{D}\boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{x}\|_1 \tag{2.14}$$

This can be rewritten to match the ADMM form from equation 2.3:

$$\begin{aligned} &\underset{\boldsymbol{x},\boldsymbol{y}}{\text{minimize}} \frac{1}{2}\|\boldsymbol{s}_i - \boldsymbol{D}\boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{y}\|_1 \\ &\text{subject to } \boldsymbol{y} - \boldsymbol{x} = \boldsymbol{0} \end{aligned} \tag{2.15}$$

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 an approximate solution to the sparse coding problem.

Computing the augmented Lagrangian of convex optimization problem in expression

[2]I have elected to notate over/under relaxation differently than standard, 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.

2.15 yields the following equation:

$$\mathrm{L}_\rho(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}) = \frac{1}{2}\|\boldsymbol{s}_i - \boldsymbol{D}\boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{y}\|_1 + \frac{\rho}{2}\|\boldsymbol{y} - \boldsymbol{x} + \frac{\boldsymbol{u}}{\rho}\|_2^2 - \frac{1}{2\rho}\|\boldsymbol{u}\|_2^2 \tag{2.16}$$

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

$$\boldsymbol{x}^{(t+1)} = \arg\min_{\boldsymbol{x}} \mathrm{L}_\rho(\boldsymbol{x}, \boldsymbol{y}^{(t)}, \boldsymbol{u}^{(t)}) \tag{2.17}$$

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

$$\nabla_{\boldsymbol{x}^{(t+1)}} \mathrm{L}_\rho(\boldsymbol{x}^{(t+1)}, \boldsymbol{y}(t), \boldsymbol{u}(t)) = \mathbf{0} \tag{2.18}$$

$$\mathbf{0} = \boldsymbol{D}^T\boldsymbol{D}\boldsymbol{x}^{(t+1)} - \boldsymbol{D}^T\boldsymbol{s}_i + \rho\boldsymbol{x}^{(t+1)} - \rho\left(\boldsymbol{y}^{(t)} + \frac{\boldsymbol{u}^{(t)}}{\rho}\right) \tag{2.19}$$

$$(\rho\mathbf{I} + \boldsymbol{D}^T\boldsymbol{D})\boldsymbol{x}^{(t+1)} = \boldsymbol{D}^T\boldsymbol{s}_i + \rho\left(\boldsymbol{y}^{(t)} + \frac{\boldsymbol{u}^{(t)}}{\rho}\right) \tag{2.20}$$

$$\boldsymbol{x}^{(t+1)} = (\rho\mathbf{I} + \boldsymbol{D}^T\boldsymbol{D})^{-1}\left(\boldsymbol{D}^T\boldsymbol{s}_i + \rho\left(\boldsymbol{y}^{(t)} + \frac{\boldsymbol{u}^{(t)}}{\rho}\right)\right) \tag{2.21}$$

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[3], there is a dual update:

$$\frac{\boldsymbol{u}^{(t+\frac{1}{2})}}{\rho} = \frac{\boldsymbol{u}^{(t)}}{\rho} + (\alpha - 1)(\boldsymbol{y}^{(t)} - \boldsymbol{x}^{(t+1)}) \tag{2.22}$$

[3]or under-relaxation

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

$$\boldsymbol{y}^{(t+1)} = \arg\min_{\boldsymbol{y}} \mathrm{L}_\rho(\boldsymbol{x}^{(t+1)}, \boldsymbol{y}, \boldsymbol{u}^{(t+\frac{1}{2})}) \tag{2.23}$$

Excluding the terms that don't include $\boldsymbol{y}$, I have

$$\boldsymbol{y}^{(t+1)} = \arg\min_{\boldsymbol{y}} \lambda\|\boldsymbol{y}\|_1 + \frac{\rho}{2}\|\boldsymbol{y} - \boldsymbol{x}^{(t+1)} + \frac{\boldsymbol{u}^{(t+\frac{1}{2})}}{\rho}\|_2^2 \tag{2.24}$$

This is a well-known problem, whose solution is

$$\boldsymbol{y}^{(t+1)} = \mathrm{S}_{\frac{\lambda}{\rho}}(\boldsymbol{x}^{(t+1)} - \frac{\boldsymbol{u}^{(t+\frac{1}{2})}}{\rho}) \tag{2.25}$$

where S is the shrinkage operator:

$$\mathrm{S}_b(x) = \begin{cases} x - b & x > b \\ 0 & -b < x < b \\ x + b & x < -b \end{cases} \tag{2.26}$$

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:

$$\frac{\boldsymbol{u}^{(t+1)}}{\rho} = \frac{\boldsymbol{u}^{(t+\frac{1}{2})}}{\rho} + \boldsymbol{y}^{(t+1)} - \boldsymbol{x}^{(t+1)} \tag{2.27}$$

Exploiting Dictionary Structure for the Inverse Problem

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

$$\boldsymbol{x}^{(t+1)} = \left(\rho\mathbf{I} + \boldsymbol{D}^T\boldsymbol{D}\right)^{-1}\left(\boldsymbol{D}^T\boldsymbol{s}_i + \rho(\boldsymbol{y}^{(t)} + \frac{\boldsymbol{u}^{(t)}}{\rho})\right) \tag{2.28}$$

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

$$\boldsymbol{D} = \begin{bmatrix} \boldsymbol{D}_{1,1}, & \ldots, & \boldsymbol{D}_{1,M} \\ \vdots & \ddots & \vdots \\ \boldsymbol{D}_{C,1}, & \ldots, & \boldsymbol{D}_{C,M} \end{bmatrix} \tag{2.29}$$

where $\boldsymbol{D}_{c,m}$ is a toplitz matrix capturing channel c of the mth filter of the dictionary. Toplitz matrices are diagonalizable with Fourier eigenvectors:

$$\boldsymbol{D} = \begin{bmatrix} \mathcal{F}^{-1}\hat{\boldsymbol{D}}_{1,1}\mathcal{F}, & \ldots, & \mathcal{F}^{-1}\hat{\boldsymbol{D}}_{1,M}\mathcal{F} \\ \vdots & \ddots & \vdots \\ \mathcal{F}^{-1}\hat{\boldsymbol{D}}_{C,1}\mathcal{F}, & \ldots, & \mathcal{F}^{-1}\hat{\boldsymbol{D}}_{C,M}\mathcal{F} \end{bmatrix} \tag{2.30}$$

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

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

$$(\rho\mathbf{I} + \boldsymbol{D}^T\boldsymbol{D})^{-1} = \mathcal{F}^{-1}(\rho\mathbf{I} + \hat{\boldsymbol{D}}^H\hat{\boldsymbol{D}})^{-1}\mathcal{F} \tag{2.31}$$

where

$$\hat{\boldsymbol{D}} = \begin{bmatrix} \hat{\boldsymbol{D}}_{1,1}, & \ldots, & \hat{\boldsymbol{D}}_{1,M} \\ \vdots & \ddots & \vdots \\ \hat{\boldsymbol{D}}_{C,1}, & \ldots, & \hat{\boldsymbol{D}}_{C,M} \end{bmatrix} \tag{2.32}$$

and in a slight abuse of notation, $\mathcal{F}$ computes the FFT separately on the coefficients for each filter. In [4], 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.)

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 [5] [6] [7].

According to the Woodbury matrix identity [8], for any invertible matrix $\boldsymbol{U}$ and any matrix $\boldsymbol{V}$:

$$(\boldsymbol{U} + \boldsymbol{V}^H\boldsymbol{V})^{-1} = \boldsymbol{U}^{-1} - \boldsymbol{U}^{-1}\boldsymbol{V}^H(\mathbf{I} + \boldsymbol{V}\boldsymbol{U}^{-1}\boldsymbol{V}^H)^{-1}\boldsymbol{V}\boldsymbol{U}^{-1} \tag{2.33}$$

So,

$$(\rho\mathbf{I} + \hat{\boldsymbol{D}}^H\hat{\boldsymbol{D}})^{-1} = \frac{1}{\rho}\mathbf{I} - \frac{1}{\rho}\hat{\boldsymbol{D}}^H(\rho\mathbf{I} + \hat{\boldsymbol{D}}\hat{\boldsymbol{D}}^H)^{-1}\hat{\boldsymbol{D}} \tag{2.34}$$

This means that instead of computing the inverse of an $M \times M$ matrix for every pixel in the image, one could instead choose to compute the inverse of a $C \times C$ matrix for each pixel in the image.[4]

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

In applying ADMM to the convolutional sparse coding problem, [5] [6] [7] 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. 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 [9][10], 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 simplied without

[4]Generally, Cholesky or LDLT decomposition would be preferable to explicitly computing the inverse, and the efficiency gains due to the Woodbury matrix identity are relevant regardless of the chosen representation.

using the frequency representation.

$$(\rho\mathbf{I} + \boldsymbol{D}^T\boldsymbol{D})^{-1} = \frac{1}{\rho}\mathbf{I} - \frac{1}{\rho(\rho+1)}\boldsymbol{D}^T\boldsymbol{D} \tag{2.35}$$

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

$$\boldsymbol{x}^{(t+1)} = \frac{1}{\rho+1}\boldsymbol{D}^T\boldsymbol{s} + \left(\mathbf{I} - \frac{1}{\rho+1}\boldsymbol{D}^T\boldsymbol{D}\right)\left(\boldsymbol{z}^{(t)} - \frac{\boldsymbol{\gamma}^{(t)}}{\rho}\right) \tag{2.36}$$

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 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.

$$\mathrm{prox}_g(\boldsymbol{\mu}) = \arg\min_{\boldsymbol{\nu}} \frac{1}{2}\|\boldsymbol{\nu} - \boldsymbol{\mu}\|_2^2 + g(\boldsymbol{\nu}) \tag{2.37}$$

Then, ISTA has the following update equation, where the constant L constrols step size.

$$\boldsymbol{x}^{(t+1)} = \mathrm{prox}_g\left(\boldsymbol{x}^{(t)} - \frac{1}{L}\nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(t)})\right) \tag{2.38}$$

FISTA is similar to ISTA, but adds momentum [11].

$$\boldsymbol{z}^{(t+1)} = \mathrm{prox}_g\left(\boldsymbol{x}^{(t)} - \frac{1}{L}\nabla_{\boldsymbol{x}} f(x^{(t)})\right) \tag{2.39}$$

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

$$r^{(t+1)} = \frac{1}{2}\left(1 + \sqrt{1 + 4(r^{(t)})^2}\right) \tag{2.40}$$

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{z}^{(t+1)} + \frac{r^{(t)} - 1}{r^{(t+1)}}(\boldsymbol{z}^{(t+1)} - \boldsymbol{x}^{(t)}) \tag{2.41}$$

Applying FISTA to the sparse coding problem, $\frac{1}{2}\|\boldsymbol{s} - \boldsymbol{D}\boldsymbol{x}\|_2^2$ is straightforward to differentiate and $\lambda\|\boldsymbol{x}\|_1$ has a simple proximal operator.

$$\nabla_{\boldsymbol{x}}\left(\frac{1}{2}\|\boldsymbol{s} - \boldsymbol{D}\boldsymbol{x}\|_2^2\right) = \boldsymbol{D}^T\boldsymbol{D}\boldsymbol{x} - \boldsymbol{D}^T\boldsymbol{s} \tag{2.42}$$

$$\mathrm{prox}_{\lambda\|\cdot\|_1}(\cdot) = \mathrm{S}_\lambda \tag{2.43}$$

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

$$\boldsymbol{z}^{(t+1)} = \mathrm{S}_\lambda\left(\boldsymbol{x}^{(t)} - \frac{1}{L}\boldsymbol{D}^T(\boldsymbol{D}\boldsymbol{x}^{(t)} - \boldsymbol{s})\right) \tag{2.44}$$

$$r^{(t+1)} = \frac{1}{2}\left(1 + \sqrt{1 + 4(r^{(t)})^2}\right) \tag{2.45}$$

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{z}^{(t+1)} + \frac{r^{(t)} - 1}{r^{(t+1)}}(\boldsymbol{z}^{(t+1)} - \boldsymbol{x}^{(t)}) \tag{2.46}$$

In [7], 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 [12], Chodosh and Lucey use updates prox-linear updates using more general convex solver methods detailed in [13].

The updates come from the formula:

$$\boldsymbol{x}^{(t+1)} = \arg\min_{\boldsymbol{x}} \left(\nabla f(\bar{\boldsymbol{x}}^{(t)})\right)^T (\boldsymbol{x} - \bar{\boldsymbol{x}}^{(t)}) + \frac{L}{2}\|\boldsymbol{x} - \bar{\boldsymbol{x}}^{(t)}\|_2^2 + \lambda\|\boldsymbol{x}\|_1 \tag{2.47}$$

where $\bar{\boldsymbol{x}}^{(t)} = \boldsymbol{x}^{(t)} + \omega_k(\boldsymbol{x}^{(t)} - \boldsymbol{x}^{(t-1)})$ and ω_t is a momentum factor.

This yields the update equation[6]:

$$\boldsymbol{x}^{(t+1)} = \mathrm{S}_{\frac{\lambda}{L}} \left(\bar{\boldsymbol{x}}^{(t)} + \boldsymbol{D}^T(\boldsymbol{s} - \boldsymbol{D}\bar{\boldsymbol{x}})\right) \tag{2.48}$$

While neither Chodosh and Lucey nor the work they cite mentions FISTA, the resemblance is very close. There are two distinctions:

1. Momentum is computed slightly differently: to match FISTA, the prox-linear updates would need to use $\boldsymbol{x}^{(t)} - \bar{\boldsymbol{x}}^{(t-1)}$ for momentum. Instead, they use $\boldsymbol{x}^{(t)} - \boldsymbol{x}^{(t-1)}$.

2. The prox-linear approach scales the momentum steps differently.

Given these similarities, it is likely the performance between the two methods is similar.

Dictionary Learning

The last few sections have focused on sparse coding. For sparse coding, the dictionary is fixed. Generally, dictionary learning algorithms alternate between pursuing coefficients and updating dictionary filters. There are many methods to updating dictionaries: FISTA, ADMM, projected stochastic gradient descent, et cetera. Dictionary updates generally adhere to the following three properties:

1. The updated dictionary filters better represents the data for the given coefficients.

2. The updated dictionary filters are normalized.

[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 constucting the equations specifically for a multi-layer network. I simplified their equations to illistrate how their approach relates to the FISTA algorithm.

3. The updated dictionary filters are spatially constrained.

Property 3 relates to the idea that convolutional filters are usually small 5×5 or 9×9, et cetera. Dictionary updates do not increase the size of the filters.

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

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.

Updating the Inverse Representation

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

$$\hat{\boldsymbol{D}} = \begin{bmatrix} \hat{\boldsymbol{D}}_{1,1}, & \dots, & \hat{\boldsymbol{D}}_{1,M} \\ \vdots & \ddots & \vdots \\ \hat{\boldsymbol{D}}_{C,1}, & \dots, & \hat{\boldsymbol{D}}_{C,M} \end{bmatrix} \tag{2.49}$$

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

$$\hat{\boldsymbol{D}}[\hat{k}] = \begin{bmatrix} \hat{\boldsymbol{D}}_{1,1}[\hat{k}], & \dots, & \hat{\boldsymbol{D}}_{1,M}[\hat{k}] \\ \vdots & \ddots & \vdots \\ \hat{\boldsymbol{D}}_{C,1}[\hat{k}], & \dots, & \hat{\boldsymbol{D}}_{C,M}[\hat{k}] \end{bmatrix} \tag{2.50}$$

Then $\hat{\boldsymbol{D}}[\hat{k}]$ is a $C \times M$ matrix collecting the $\hat{k}$th freqency 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.

$$\hat{\boldsymbol{D}}[\hat{k}]^{(n+1)} = \hat{\boldsymbol{D}}[\hat{k}]^{(n)} + \boldsymbol{U}\boldsymbol{V}[\hat{k}]^H \tag{2.51}$$

where $\boldsymbol{U}$ is an orthogonal matrix of size $C \times L$ and $\boldsymbol{V}[\hat{k}]$ is an orthogonal matrix of size $M \times L$.[7]

Then,

$$\rho\mathbf{I} + (\hat{\boldsymbol{D}}^{(n+1)})^H[\hat{k}]\hat{\boldsymbol{D}}^{(n+1)}[\hat{k}] = \rho\mathbf{I} + (\boldsymbol{D}^{(n)}[\hat{k}] + \boldsymbol{U}\boldsymbol{V}^H[\hat{k}])^H(\boldsymbol{D}^{(n)}[\hat{k}] + \boldsymbol{U}\boldsymbol{V}^H[\hat{k}]) \tag{2.52}$$

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 $\hat{k}$, not the sparsely banded $\hat{\boldsymbol{D}}$ of size $KC \times M$ from

[7] $\boldsymbol{U}$ and $\boldsymbol{V}$ are also iteration specific, but to notate that would over-clutter the equations. For efficient, low-rank updates to the inverse representation, I could allow both $\boldsymbol{U}$ and $\boldsymbol{V}$ to vary in respect to frequency $\hat{k}$ (instead of just $\boldsymbol{V}$). However, I also need to limit the spatial support of the dictionary (so that the filter size is small), and preventing $\boldsymbol{U}$ from varying across frequency is part of a means to satisfy that constraint.

earlier in this section.

$$\rho\mathbf{I}+(\hat{\boldsymbol{D}}^{(n+1)})^H\hat{\boldsymbol{D}}^{(n+1)} = \rho\mathbf{I}+(\hat{\boldsymbol{D}}^{(n)})^H\hat{\boldsymbol{D}}^{(n)}+\boldsymbol{V}\boldsymbol{U}^H\boldsymbol{U}\boldsymbol{V}^H+\boldsymbol{V}\boldsymbol{U}^H\hat{\boldsymbol{D}}^{(n)}+(\hat{\boldsymbol{D}}^{(n)})^H\boldsymbol{U}\boldsymbol{V}^H \tag{2.53}$$

Given that $\boldsymbol{V}$ and $\boldsymbol{U}$ are orthogonal matrices, $\boldsymbol{V}\boldsymbol{U}^H\boldsymbol{U}\boldsymbol{V}^H$ can easily be broken into L rank-one Hermitian updates.

$$\boldsymbol{V}\boldsymbol{U}^H\boldsymbol{U}\boldsymbol{V}^H = \sum_{\ell=1}^{L} \boldsymbol{u}_\ell^H\boldsymbol{u}_\ell\boldsymbol{v}_\ell\boldsymbol{v}_\ell^H \tag{2.54}$$

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

$$\boldsymbol{V}\boldsymbol{U}^H\hat{\boldsymbol{D}} + \hat{\boldsymbol{D}}^H\boldsymbol{U}\boldsymbol{V}^H = \sum_{\ell=1}^{L} \boldsymbol{v}_\ell\boldsymbol{u}_\ell^H\hat{\boldsymbol{D}} + \hat{\boldsymbol{D}}^H\boldsymbol{u}_\ell\boldsymbol{v}_\ell^H \tag{2.55}$$

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

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 eigendecomposition.

$$\boldsymbol{v}_\ell\boldsymbol{u}_\ell^H\hat{\boldsymbol{D}} + \hat{\boldsymbol{D}}\boldsymbol{u}_\ell\boldsymbol{v}_\ell^H = \begin{bmatrix}\boldsymbol{v}_\ell & \hat{\boldsymbol{D}}^H\boldsymbol{u}_\ell\end{bmatrix}\begin{bmatrix}\hat{\boldsymbol{D}}^H\boldsymbol{u}_\ell & \boldsymbol{v}_\ell\end{bmatrix}^H \tag{2.56}$$

While matrix products are not communitive, some of the eigenvalues of matrix products are communitive.

Furthermore, for general matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ the eigenvectors of $\boldsymbol{AB}$ and $\boldsymbol{BA}$ are related:

$$\boldsymbol{BAx} = \lambda\boldsymbol{x} \implies \boldsymbol{ABAx} = \lambda\boldsymbol{Ax} \tag{2.57}$$

where λ is the eigenvalue and $\boldsymbol{x}$ is a vector.[8]

So, if $\boldsymbol{x}$ is an eigenvector of $\boldsymbol{BA}$, $\boldsymbol{Ax}$ is an eigenvector of $\boldsymbol{AB}$.

$$\begin{bmatrix} \hat{\boldsymbol{D}}^H \boldsymbol{u}_\ell & \boldsymbol{v}_\ell \end{bmatrix}^H \begin{bmatrix} \boldsymbol{v}_\ell & \hat{\boldsymbol{D}}^H \boldsymbol{u}_\ell \end{bmatrix} = \begin{bmatrix} \boldsymbol{u}_\ell^H \hat{\boldsymbol{D}} \boldsymbol{v}_\ell & \boldsymbol{u}_\ell^H \hat{\boldsymbol{D}} \hat{\boldsymbol{D}}^H \boldsymbol{u}_\ell \\ \boldsymbol{v}_\ell^H \boldsymbol{v}_\ell & \boldsymbol{v}_\ell^H \hat{\boldsymbol{D}}^H \boldsymbol{u}_\ell \end{bmatrix} \tag{2.58}$$

The eigenvalues and corresponding eigenvectors of a 2×2 matrix can be computed using the quadradic formula. Assuming that the 2×2 matrix has 2 distinct eigenvalues, the expressions for these are below.[9]

$$\text{eigval}\left(\begin{bmatrix} a & b \\ c & a^* \end{bmatrix}\right) = \text{real}(a) \pm \sqrt{bc - (\text{imag}(a))^2} \tag{2.59}$$

$$\text{eigvec}\left(\begin{bmatrix} a & b \\ c & a^* \end{bmatrix}\right) = \begin{bmatrix} b \\ -j\,\text{imag}(a) \pm \sqrt{bc - (\text{imag}(a))^2} \end{bmatrix} \tag{2.60}$$

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 $\eta_{\boldsymbol{u},\boldsymbol{v}} = \boldsymbol{u}^H \hat{\boldsymbol{D}} \boldsymbol{v}$:

$$\text{eigval}\left(\begin{bmatrix} \eta_{\boldsymbol{u},\boldsymbol{v}} & \eta_{\boldsymbol{u}} \\ \eta_{\boldsymbol{v}} & \eta_{\boldsymbol{u},\boldsymbol{v}}^* \end{bmatrix}\right) = \text{real}(\eta_{\boldsymbol{u},\boldsymbol{v}}) \pm \sqrt{\eta_{\boldsymbol{v}}\eta_{\boldsymbol{u}} - (\text{imag}(\eta_{\boldsymbol{u},\boldsymbol{v}}))^2} \tag{2.61}$$

$$\text{eigvec}\left(\begin{bmatrix} \eta_{\boldsymbol{u},\boldsymbol{v}} & \eta_{\boldsymbol{u}} \\ \eta_{\boldsymbol{v}} & \eta_{\boldsymbol{u},\boldsymbol{v}}^* \end{bmatrix}\right) = \begin{bmatrix} \eta_{\boldsymbol{u}} \\ -j\,\text{imag}(\eta_{\boldsymbol{u},\boldsymbol{v}}) \pm \sqrt{\eta_{\boldsymbol{v}}\eta_{\boldsymbol{u}} - (\text{imag}(\eta_{\boldsymbol{u},\boldsymbol{v}}))^2} \end{bmatrix} \tag{2.62}$$

[8] I appologize for the reuse of certain variables here. Please do not confuse this $\boldsymbol{x}$ for the sparse coding coefficients, λ for the L1 penalty factor applied to the coefficients, or $\boldsymbol{A}$ and $\boldsymbol{B}$ for the matrices in the ADMM constraints.

[9] It is not guarenteed the 2×2 matrix will have 2 distinct eigenvalues. In the practical considerations section, I consider those cases.

Therefore,

$$\text{eigvec}(\boldsymbol{v}\boldsymbol{u}^H\hat{\boldsymbol{D}} + \hat{\boldsymbol{D}}^H\boldsymbol{u}\boldsymbol{v}^H) = \eta_{\boldsymbol{u}}\boldsymbol{v} + \left(-j\,\text{imag}(\eta_{\boldsymbol{u},\boldsymbol{v}}) \pm \sqrt{\eta_{\boldsymbol{v}}\eta_{\boldsymbol{u}} - (\text{imag}(\eta_{\boldsymbol{u},\boldsymbol{v}}))^2}\right)\hat{\boldsymbol{D}}^H\boldsymbol{u} \tag{2.63}$$

$$\text{eigval}(\boldsymbol{v}\boldsymbol{u}^H\hat{\boldsymbol{D}} + \hat{\boldsymbol{D}}^H\boldsymbol{u}\boldsymbol{v}^H) = \text{real}(\eta_{\boldsymbol{u},\boldsymbol{v}}) \pm \sqrt{\eta_{\boldsymbol{v}}\eta_{\boldsymbol{u}} - (\text{imag}(\eta_{\boldsymbol{u},\boldsymbol{v}}))^2} \tag{2.64}$$

This decomposition splits the Hermitian rank-two update into two Hermitian rank-one updates that can be used to update the inverse representation for $\rho\mathbf{I} + \hat{\boldsymbol{D}}^H[\hat{k}]\hat{\boldsymbol{D}}[\hat{k}]$.

Recall once again, the update under consideration:

$$\hat{\boldsymbol{D}}^{(n+1)}[\hat{k}] = \hat{\boldsymbol{D}}^{(n)}[\hat{k}] + \boldsymbol{U}\boldsymbol{V}^H[\hat{k}] \tag{2.65}$$

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

Handling Dictionary Normalization

Consider the optimization problem:

$$\begin{aligned} &\min_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}} \frac{1}{2}\|\boldsymbol{s} - \boldsymbol{D}\boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{y}\|_1 \\ &\text{subject to } \boldsymbol{R}^{-1}\boldsymbol{y} - \boldsymbol{R}^{-1}\boldsymbol{x} = 0 \end{aligned} \tag{2.66}$$

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

$$\boldsymbol{R} = \begin{bmatrix} r_1\mathbf{I} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & r_2\mathbf{I} & & \vdots \\ \vdots & & \ddots & \\ \mathbf{0} & \dots & & r_M\mathbf{I} \end{bmatrix} \tag{2.67}$$

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

This optimization problem has the augmented Lagrangian function:

$$\mathrm{L}_\rho(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}) = \frac{1}{2}\|\boldsymbol{s} - \boldsymbol{D}\boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{y}\|_1 + \boldsymbol{u}^H\boldsymbol{R}^{-1}(\boldsymbol{y} - \boldsymbol{x}) + \frac{\rho}{2}\|\boldsymbol{R}^{-1}(\boldsymbol{y} - \boldsymbol{x})\|_2^2 \tag{2.68}$$

$$\nabla_{\boldsymbol{x}} \mathrm{L}_\rho(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}) = -\boldsymbol{R}^{-1}\boldsymbol{u} - \boldsymbol{D}^H\boldsymbol{s} + \boldsymbol{D}^T\boldsymbol{D}\boldsymbol{x} + \rho\boldsymbol{R}^{-2}\boldsymbol{x} - \rho\boldsymbol{R}^{-2}\boldsymbol{y} \tag{2.69}$$

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

$$(\rho\boldsymbol{R}^{-2} + \boldsymbol{D}^T\boldsymbol{D})\boldsymbol{x} = \rho\boldsymbol{R}^{-2}\boldsymbol{y} + \boldsymbol{R}^{-1}\boldsymbol{u} + \boldsymbol{D}^T\boldsymbol{s} \tag{2.70}$$

$$\boldsymbol{R}^{-1}\left(\rho\mathrm{I} + (\boldsymbol{D}\boldsymbol{R})^T(\boldsymbol{D}\boldsymbol{R})\right)\boldsymbol{R}^{-1}\boldsymbol{x} = \rho\boldsymbol{R}^{-2}\boldsymbol{y} + \boldsymbol{R}^{-1}\boldsymbol{u} + \boldsymbol{D}^T\boldsymbol{s} \tag{2.71}$$

$$\left(\rho\mathbf{I} + (\boldsymbol{D}\boldsymbol{R})^T(\boldsymbol{D}\boldsymbol{R})\right)\boldsymbol{R}^{-1}\boldsymbol{x} = \rho\boldsymbol{R}^{-1}\boldsymbol{z} + \boldsymbol{u} + (\boldsymbol{D}\boldsymbol{R})^T\boldsymbol{s} \tag{2.72}$$

$$\boldsymbol{R}^{-1}\boldsymbol{x} = \left(\rho\mathbf{I} + (\boldsymbol{D}\boldsymbol{R})^H(\boldsymbol{D}\boldsymbol{R})\right)^{-1}\left(\rho\boldsymbol{R}^{-1}\boldsymbol{y} + \boldsymbol{u} + (\boldsymbol{D}\boldsymbol{R})^T\boldsymbol{s}\right) \tag{2.73}$$

So,

$$\min_{\boldsymbol{x}} \mathrm{L}_\rho(\boldsymbol{x},\boldsymbol{y},\boldsymbol{u}) = \boldsymbol{R}\left(\rho\mathbf{I} + (\boldsymbol{D}\boldsymbol{R})^T(\boldsymbol{D}\boldsymbol{R})\right)^{-1}\left(\rho\boldsymbol{R}^{-1}\boldsymbol{y} + \boldsymbol{u} + (\boldsymbol{D}\boldsymbol{R})^T\boldsymbol{s}\right) \tag{2.74}$$

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

$$\boldsymbol{R}^{-1}\boldsymbol{x}^{(t+1)} = \left(\rho\mathbf{I} + (\boldsymbol{D}\boldsymbol{R})^T(\boldsymbol{D}\boldsymbol{R})\right)^{-1}\left((\boldsymbol{D}\boldsymbol{R})^T\boldsymbol{s} + \rho\left(\boldsymbol{R}^{-1}\boldsymbol{y}^{(t)} + \frac{\boldsymbol{u}^{(t)}}{\rho}\right)\right) \tag{2.75}$$

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

$$\min_{\boldsymbol{y}} \mathrm{L}_\rho(\boldsymbol{x},\boldsymbol{y},\boldsymbol{u}) = \mathrm{S}_{\frac{\lambda\boldsymbol{R}^2}{\rho}}\left(\boldsymbol{x} - \frac{\boldsymbol{R}\boldsymbol{u}}{\rho}\right) \tag{2.76}$$

$$\boldsymbol{R}^{-1}\boldsymbol{y}^{(t+1)} = \mathrm{S}_{\frac{\lambda\boldsymbol{R}}{\rho}}\left(\boldsymbol{R}^{-1}\boldsymbol{x}^{(t+1)} - \frac{\boldsymbol{u}^{\left(t+\frac{1}{2}\right)}}{\rho}\right) \tag{2.77}$$

Finally, the dual updates are

$$\frac{\boldsymbol{u}^{\left(t+\frac{1}{2}\right)}}{\rho} = \frac{\boldsymbol{u}^{(t)}}{\rho} + (\alpha - 1)(\boldsymbol{R}^{-1}\boldsymbol{y}^{(t)} - \boldsymbol{R}^{-1}\boldsymbol{x}^{(t+1)}) \tag{2.78}$$

$$\frac{\boldsymbol{u}^{(t+1)}}{\rho} = \frac{\boldsymbol{u}^{\left(t+\frac{1}{2}\right)}}{\rho} + \boldsymbol{R}^{-1}\boldsymbol{y}^{(t+1)} - \boldsymbol{R}^{-1}\boldsymbol{x}^{(t+1)} \tag{2.79}$$

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 $\boldsymbol{R}^{-1}$.

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 when used within a dictionary-learning algorithm.

CHAPTER 3
LEARNING MULTI-LAYER DICTIONARIES

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 [14] and can be thought of as a deep autoencoder [15, Chapter 14][16]. 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 [17]. In this chapter, I explain how to apply the novel dictionary learning algorithm from the prior chapter to the multi-layer dictionary learning problem.

Literature Review

In 2010, Zeiler et al. proposed a multi-layer dictionary model termed a deconvolutional network. 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 [18]. While this research drew a lot of attention at the time, as the success of alternative models like convolutional neural networks grew, the popularity of deconolutional networks decreased.

Multi-layer dictionaries also appear in Bayesian models, going by names such as hierarchical convolutional factor analysis [19][20] and deep deconvolutional learning [21]. These networks use probabilistic models to prune network architecture and provide interpretable dictionaries. Inference can be slow.

In more recent work, [10] and [9] 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 [12], Chodosh and Lucey use a similar model to [10] and [9], but replace the ADMM approach with FISTA-like linear-proximal iterative steps.

Multi-Layer ADMM with Low-Rank Updates

This chapter demostrates 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 the same multi-layer dictionary model as [10] and [9], but I use different ADMM constraints:

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

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 [10] and [9]. Again, keeping things compact, let $\boldsymbol{z}_0 = \boldsymbol{s}$.

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

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 algoritm (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.

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 $\boldsymbol{x}$.

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

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

Therefore,

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

For $\boldsymbol{x}$, $\boldsymbol{z}$, $\boldsymbol{\gamma}$, such that $\nabla_{\boldsymbol{x}_\ell} \mathcal{L}_\rho(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_L, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_L, \gamma_1, \ldots, \gamma_L) = 0$:

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

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

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

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.

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,

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

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

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

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

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

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

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

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

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

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

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

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

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

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.

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

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

Dual Updates

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

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

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

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.

CHAPTER 4
JPEG ARTIFACT REMOVAL

Introduction

Despite the existance 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.

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 reodered 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.

Literature Review

Modelling Compressed JPEG Images

Some researchers have observed convolutional dictionary models struggle with large smooth components of signals, 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 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.

$$\boldsymbol{s} = \boldsymbol{Q}\boldsymbol{W}\boldsymbol{s}_{\text{clean}} \tag{4.3}$$

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

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

A means of estimating $\boldsymbol{s}_{\text{smth}}$ from JPEG-compressed image $\boldsymbol{s}$ is discussed in the Practical Considerations chapter.

From this idea, I construct the pursuit problem:

$$\begin{aligned}&\underset{\boldsymbol{x}}{\text{minimize}}\frac{\mu_1}{2}\|\boldsymbol{s}-\boldsymbol{QW}(\boldsymbol{D}_1\boldsymbol{x}_1+\boldsymbol{s}_{\text{smth}})\|_2^2+\sum_{\ell=2}^{L}\frac{\mu_\ell}{2}\|\boldsymbol{x}_{\ell-1}-\boldsymbol{D}_\ell\boldsymbol{x}_\ell\|_2^2+\sum_{\ell=1}^{L}\lambda_\ell\|\boldsymbol{x}_\ell\|_1\\&\text{subject to } \boldsymbol{x}_\ell>\boldsymbol{0}\end{aligned} \tag{4.5}$$

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

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 contraints on $\boldsymbol{z}_1,\ldots,\boldsymbol{z}_L$. Here is the corresponding optimization problem:

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

The constraint $\boldsymbol{QW}(\boldsymbol{v})-\boldsymbol{s}=\boldsymbol{0}$ is not an affine constraint because of the quantization. To resolve this, [12]) approximate the quantization as a linear operator. However, the constraint is convex, so the constraint can be handled without approximation implicitly using and indicator function. 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 identitical to those from the last chpater.

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

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

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

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

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

The only remaining update equation is for $\boldsymbol{v}$. This deals with the non-affine constraint, which complicates the problem. I will present a method for handling the quantization operator in the constraint in the next section.

Handling Quantization

Recall the optimization problem:

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

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

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

The function is convex, and when included in an objective function, it implicitly enforces the constraint $\boldsymbol{Q}(\boldsymbol{W}\boldsymbol{v}) - \boldsymbol{s} = \boldsymbol{0}$ This can be rewritten as the following:[1]

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

Adding the indicator function to the objective produces the following Lagrangian function:

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

where $h\psi(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\gamma})$ 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:

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

[1]The set $\{\boldsymbol{v} : \boldsymbol{Q}(\boldsymbol{W}\boldsymbol{v}) - \boldsymbol{s} = \boldsymbol{0}\}$ 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 $\boldsymbol{v}$.

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 $\boldsymbol{W}$.

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.

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 uncompressed 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.

Results

My code is still running, so I do not have results to share yet. I intend to compare to a proximal algorithm like [12] or FISTA, single-layer ADMM, and the tight-frame approximation for the inverse problem.

Conclusion

It is hard to reach conclusions until my code is finished, but I am hopeful that my approach will be a competitive alternative to FISTA, and outperform single-layer ADMM and tight-frame approximations.

CHAPTER 5
PRACTICAL CONSIDERATIONS

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 [**wohlberg2016boundary**], but with the constraint and objective term swapped). Initialization is important however [**DBLP:journals/corr/WohlbergR17**]. 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.

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. Tikonov regularization is a straightforward means to remove low-frequency content:

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

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.

Tikonov regularation 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. Jut alternate between a Tikonov update for $\boldsymbol{x}$ and an update for $\boldsymbol{v}$ that closely resembles the $\boldsymbol{v}$ update from the last chapert. 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,

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

Tensorflow and Keras

Most of the computations for my research rely on TensorFlow version 2.3.1 and version 2.4.1 [22], 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 exampes freely available online, so I will not replicate that work here.

This chapter section the reader already has some familiarity with TensorFlow and Keras [23] (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.

Why Not Use Gradient Tape and TensorFlow-1-Stye 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.

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 be 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]

[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.

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:

```
@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.

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 PostProcess:
   update = {}
   def add_update(varName,update_fun):
      assert varName not in PostProcess.update
      PostProcess.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 Tensorflow 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 PostProcess.update:
            PostProcess.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 dependancies 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 PostProcess class is still fine, but the update functions must be instead be called using Keras callbacks.

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

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

      for an_update in self.updates:
         an_update()

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

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 GPU errors. The TensorFlow Probability version 0.11.1 [24] is an extension of TensorFlow mosly 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, users should either write their own implementation or use my code (included in supplementary material). 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.

Appendices

APPENDIX A
CHOLESKY HERMITIAN RANK-1 UPDATES

I am using the methods described in [**krause2015more**], modified to handle complex numbers. I have not written this up yet.

APPENDIX B
RANK-2 EIGENDECOMPOSITION EDGE CASES

Less than 2 Independent Eigenvectors

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

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.
$$\boldsymbol{D}^H\boldsymbol{u} = \boldsymbol{0} \tag{B.2}$$

2.
$$\boldsymbol{v} = \boldsymbol{0} \tag{B.3}$$

3.
$$\boldsymbol{D}^H\boldsymbol{u} = \alpha\boldsymbol{v} \tag{B.4}$$

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 $\boldsymbol{BA}$.

$$\boldsymbol{AB} = 2\,\mathrm{real}(\alpha)\boldsymbol{v}\boldsymbol{v}^H \tag{B.5}$$

$$\lambda = 2\,\mathrm{real}(\alpha)\|\boldsymbol{v}\|_2^2 \tag{B.6}$$

$$\boldsymbol{x} = \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|_2} \tag{B.7}$$

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.)

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.

REFERENCES

[1] 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.

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

[3] 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.

[4] 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.

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

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] Y. Pu, X. Yuan, and L. Carin, “Generative deep deconvolutional learning,” *ArXiv preprint arXiv:1412.6039*, 2014.

[22] 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.

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

[24] 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.