Sparse Sensing for Compression and Denoising

Thomas Kost

May 24, 2022

Contents

1 Introduction

2 Dataset

For this project we will be using the cropped Extended Yale Face Database B[3]. This contains 16128 images of 28 human subjects under 9 poses and 64 illumination conditions.Each image is 192x168 pixels. For the purpose of this project we will be making use of a subset of these images. We will be choosing 50 images of the same individual in different illumination conditions in a single pose. The images will be vectorized to form a database in $\mathbb{R}^{32256x50}$.

3 Sparse Sensor Selection

$$y = C\Psi x$$

$$y_i = C_i\psi x$$

$$
\begin{aligned}
&\min_C -\log\det\left(\sum_{i=1}^{r}\Psi^T C_i^T C_i \Psi\right) + \lambda||C||_1 \\
&st. \quad C\mathbf{1} = \mathbf{1} \\
&\qquad 0 \leq C \leq 1
\end{aligned}
$$

3.1 PDHG

3.2 ADMM

We will now derive the ADMM implementation for this problem.We will first begin by reformatting our problem to replace our inequality constraint with an indicator function:

$$
\min_C -\log\det\left(\sum_{i=1}^{r}\Psi^T C_i^T C_i \Psi\right) + \lambda||C||_1 + \delta_{0\leq C_{i,j}\leq 1}(C) \;\; s.t. \;\; C\mathbf{1} = \mathbf{1}
$$

We can now form the augmented Lagrangian for our optimization problem. This is as follows:

$$
\begin{aligned}
\mathcal{L} &= -\log\det\left(\sum_{i=1}^{r}\Psi^T C_i^T C_i \Psi\right) + \lambda||C||_1 + \delta_{0\leq C_{i,j}\leq 1}(C) + z^T(C\mathbf{1}-\mathbf{1}) + \frac{t}{2}||C\mathbf{1}-\mathbf{1}||_2^2 \\
&= -\log\det\left(\sum_{i=1}^{r}\Psi^T C_i^T C_i \Psi\right) + \lambda||C||_1 + \delta_{0\leq C_{i,j}\leq 1}(C) + z^T(C\mathbf{1}-\mathbf{1}) + \frac{t}{2}||C\mathbf{1}-\mathbf{1}||_2^2
\end{aligned}
$$

4 Sparse Sensor Placement

$$
y = C\Psi a
$$

where y observation, C sparse measurement matrix, Ψ is tailored basis, and a is a low rank approximation. Note $\Theta = C\Psi$

$$
\begin{aligned}
&\min_C \;\; ||\Theta||_F^2 + ||C||_1 \\
&s.t \;\; \Theta > 0
\end{aligned}
$$

4.1 Algorithm and Derivation

4.1.1 PDHG

We derived the Primal-Dual Hybrid Gradient(PDHG) for the sparse sensor placement problem.

4.1.2 FISTA

We will first describe the derivation and implementation of FISTA(Fast Iterative Shrinkage-Thresholding Algorithm) for the sparse sensor placement problem. We will first augment our problem to fit the framework of the proximal gradient method. We will do this through incorporating our constraint into a indicator function:

$$\min_{C} \quad ||C\Psi||_F^2 + ||C||_1 + \delta_{C\Psi > 0}(C)$$

Here we will denote $g(X) = ||X\Psi||_F^2$ (to be consistent with the literature). Note that $g(X)$ is differentiable, smooth, and $dom(g) = \mathbb{R}^n$. We correspondingly denote $h(X) = ||X||_1 + \delta_{X\Psi>0}(X)$. We will see that $h(X)$ has a simple proximal operator. We find the following facts:

$$\begin{aligned}
\nabla_X g(X) =& 2X\Psi\Psi^T \\
prox_{th}(X) =& \arg\min_{U} ||U||_1 + \delta_{U\Psi>0}(U) + \frac{t}{2}||U - X||_F^2 \\
=& P_{S_+^n}\left(\hat{X} + \gamma I\right)\Psi^\dagger \\
\hat{X}_{i,j} =& sign(X_{i,j})max(|X_{i,j}| - t, 0)
\end{aligned}$$

Here $gamma \in \mathbb{R}$ is a small constant to enforce positive definiteness. As a result our update equation for a vanilla proximal gradient method is given by:

$$C_{k+1} = prox_{th(\cdot)}\left(C_k - t\nabla_C g(C)\right)$$

4.1.3 ADMM

We will now take a look at an ADMM (Alternating Directions Method of Multipliers) implementation for this problem. We will again modify our problem through incorporating our constraint into a indicator function. This will result in the following reformulation:

$$\min_{C} \quad ||C\Psi||_F^2 + ||C||_1 + \delta_{C\Psi > 0}(C)$$

We will then add a splitting variables to ease our formulation of the problem.

$$\begin{aligned}
\min_{C} \quad & ||C\Psi||_F^2 + ||C||_1 + \delta_{A>0}(\Theta) \\
s.t. \quad & \Theta = C\Psi \\
& B = C
\end{aligned}$$

This results in the following augmented Lagrangian:

$$\begin{aligned}
\mathcal{L} =& ||C\Psi||_F^2 + ||B||_1 + \delta_{\Theta>0}(\Theta) + tr(Z^T(C\Psi - \Theta)) + \frac{t}{2}||C\Psi - \Theta||_F^2 + tr(Y^T(C - B)) + \frac{t}{2}||C - B||_F^2 \\
=& ||C\Psi||_F^2 + ||B||_1 + \delta_{\Theta>0}(\Theta) + \frac{t}{2}||C\Psi - \Theta + \frac{Z}{t}||_F^2 + \frac{t}{2}||C - B + \frac{Y}{t}||_F^2
\end{aligned}$$

We will now minimize C and Θ in an alternating fashion. We will minimize over (Θ, B) as a group. This will yield the following update algorithms:

$$
\begin{aligned}
C_{k+1} &= \arg\min_{C} ||C\Psi||_F^2 + \frac{t}{2}||C\Psi - \Theta_k + \frac{Z_k}{t}||_F^2 + \frac{t}{2}||C - B_k + \frac{Y_k}{t}||_F^2 \\
C_{k+1} &= \frac{1}{2+t}\left(t\Theta_k\Psi^T + tB_k - Z_k\Psi^T - Y_k\right)\left(\Psi\Psi^T + tI\right)^{-1} \\
(\Theta_{k+1}, B_{k+1}) &= \arg\min_{\Theta, B} ||B||_1 + \delta_{\Theta>0}(\Theta) + \frac{t}{2}||C_{k+1}\Psi - \Theta + \frac{Z_k}{t}||_F^2 + \frac{t}{2}||C_{k+1} - B + \frac{Y_k}{t}||_F^2
\end{aligned}
$$

We can note here that Θ and B are separable. Thus, this yields the following separate update equations:

$$
\begin{aligned}
\Theta_{k+1} &= \arg\min_{\Theta} \delta_{\Theta>0}(\Theta) + \frac{t}{2}||C_{k+1}\Psi - \Theta + \frac{Z_k}{t}||_F^2 \\
&= P_{S_+^n}(C_{k+1}\Psi + \frac{Z_k}{t}) \\
B_{k+1} &= \arg\min_{B} ||B||_1 + \frac{t}{2}||C_{k+1} - B + \frac{Y_k}{t}||_F^2 \\
&= prox_{t^{-1}||\cdot||_1}(C_{k+1} + \frac{Y_k}{t})
\end{aligned}
$$

Given these updates we can update our dual multipliers via the following update:

$$
\begin{aligned}
Z_{k+1} &= Z_k + t(C_{k+1}\Psi - \Theta_{k+1}) \\
Y_{k+1} &= Y_k + t(C_{k+1} - B_{k+1})
\end{aligned}
$$

4.2 Results

CVX could not solve this problem as it ran out of memory within 2 iterations. As a result, general purpose solvers are incapable of solving this given problem.

5 RPCA

$$
\begin{aligned}
\min_{L,S} \;\; & ||L||_* + \lambda||S||_1 \\
s.t \;\; & L + S = X
\end{aligned}
$$

5.1 Algorithm and Derivation

5.1.1 PDHG

We derived the Primal-Dual Hybrid Gradient(PDHG) method for the RPCA optimization problem. To do so we will write our problem as:

$$
\min_{L} ||L||_* + \lambda||X - L||_1
$$

We will denote $f(Y) = ||Y||_*$ and $g(Y) = \lambda||X - Y||_1$. Therefore the conjugates $f^*(X)$ and $g^*(Y)$ can be defined as below:

$$\begin{aligned}
f^*(Y) =& \sup_{X \in dom(f)} tr(Y^T X) - ||X||_* \\
=& \delta_{||\cdot||_2 \leq 1}(X) \\
g^*(Y) =& \sup_{Z \in dom(f)} tr(Y^T Z) - \lambda||X - Z||_* \\
=& \lambda \left(tr(\frac{Y^T X}{\lambda}) - \delta_{||\cdot||_\infty \leq 1}(\frac{Y}{\lambda}) \right)
\end{aligned}$$

Given this, we can now formulate our dual problem, and begin the derivation of the PDHG update steps. Our dual problem is given by:

$$\max_Z -g^*(Z) - f^*(-Z)$$

Correspondingly, our update steps are given by:

$$\begin{aligned}
L_{k+1} =& prox_{tf}(L_k - tZ_k) \\
Z_{k+1} =& prox_{\tau g^*}(Z_k + \tau(2L_{k+1} - L_k))
\end{aligned}$$

where we define the following proximal gradient functions:

$$\begin{aligned}
prox_{tf}(Y) =& \sum_i max(0, \sigma_i - t) u_i v_i^T \\
prox_{\tau g^*}(Y)_{i,j} =& Y_{i,j} - \frac{\tau}{\lambda} \left(prox_{\lambda^2 \tau^{-1} g}(\frac{\lambda}{\tau} Y - X)_{i,j} + X_{i,j} \right) \\
=& Y_{i,j} - \frac{\tau}{\lambda} X_{i,j} - \frac{\tau}{\lambda} sign(\frac{\lambda}{\tau} Y_{i,j} - X_{i,j}) max(|\frac{\lambda}{\tau} Y_{i,j} + X_{i,j}| - \frac{\lambda^2}{\tau}, 0)
\end{aligned}$$

Where σ_i,u_i, and v_i are the associated singular values, left singular vectors, and right singular vectors respectively.

5.1.2 ADMM

We derived the ADMM for the RPCA optimization problem. To do so we first derived the augmented Lagrangian for our problem:

$$\begin{aligned}
\mathcal{L} =& ||L||_* + \lambda||S||_1 + tr(Z^T(X - L - S)) + \frac{t}{2}||X - L - S||_F^2 \\
=& ||L||_* + \lambda||S||_1 + \frac{t}{2}||X - L - S + \frac{Z}{t}||_F^2
\end{aligned}$$

We can then minimize along L and S independently to form our alternating update algorithm:

$$
\begin{aligned}
L_{k+1} =& \arg\min_{L} \quad ||L||_* + \frac{t}{2}||X - L - S_k + \frac{Z_k}{t}||_F^2 \\
=& prox_{t^{-1}||\cdot||_*}\left(X - S_k + \frac{Z_k}{t}\right) \\
S_{k+1} =& \arg\min_{S} \quad \lambda||S||_1 + \frac{t}{2}||X - L_{k+1} - S + \frac{Z_k}{t}||_F^2 \\
=& prox_{\lambda t^{-1}||\cdot||_1}\left(X - L + \frac{Z_k}{t}\right)
\end{aligned}
$$

Here the proximal operator of the nuclear norm and the L_1 norm are defined below. Where σ_i,u_i, and v_i are the associated singular values, left singular vectors, and right singular vectors respectively.

$$
\begin{aligned}
prox_{t||\cdot||_*}(X) =& \sum_i max(0, \sigma_i - t)u_i v_i^T \\
prox_{t||\cdot||_1}(X)_{i,j} =& sign(X_{i,j})max(|X_{i,j}| - t, 0)
\end{aligned}
$$

Given these updates we can update our dual multipliers via the following update:

$$
Z_{k+1} = Z_k + t\,(X - L_{k+1} - S_{k+1})
$$

This concludes the derivation of our ADMM algorithm.

5.2 Results

CVX was able to handle the RPCA image decomposition problem, however it took 3.5 hours for a single image.

References

[1] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[2] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control.* Cambridge University Press, USA, 1st edition, 2019.

[3] A.S. Georghiades, P.N. Belhumeur, and D.J. Kriegman. From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE Trans. Pattern Anal. Mach. Intelligence*, 23(6):643–660, 2001.

[4] Charles Guyon, Thierry Bouwmans, and El-hadi ZAHZAH. *Robust Principal Component Analysis for Background Subtraction: Systematic Evaluation and Comparative Analysis*. 03 2012.

[5] Siddharth Joshi and Stephen Boyd. Sensor selection via convex optimization. *IEEE Transactions on Signal Processing*, 57(2):451–462, 2009.

A MATLAB Code

The lines below contain all the code written with respect to the Sparse Sensor placement problem:

```
%%
 %  File: CS_images.m
 %  Author: Thomas Kost
 %
 %  Date: 6 May 2022
 %
 %  @brief Determination of Sparse sensors given a tailored basis
 %
 clc, clear all, close all;
%% Run Variables:
 run_CVX = false;
 run_ADMM = true;
%% Load Data:
 im_paths = dir(fullfile('CroppedYale\yaleB01\', '*0.pgm'));
 num_im = numel(im_paths);
 im_size = size(imread(fullfile(im_paths(1).folder,im_paths(1).name)))/2;
 dataset = zeros(num_im, im_size(1),im_size(2),'uint8');
 for i = 1:num_im
      im = imread(fullfile(im_paths(i).folder,im_paths(i).name));
      dataset(i,:,:) = im(1:2:end, 1:2:end);
 end
 disp("Data Read in...");
 % Pick a random image
rand_index = randi([0 num_im],1,1);

%% Pick Example and remove from basis
figure();
subplot(1,2,1);
vector_dim = im_size(1)*im_size(2);
orig_im_vec = reshape(dataset(1,:,:), (vector_dim,1));
orig_im = reshape(dataset(1,:,:), im_size);
dataset = dataset(2:end,:,:);
imshow(orig_im);
% Add Salt and Pepper Noise
p = 0.7;
noise_probs = rand(im_size);
noisy_image = orig_im - uint8(noise_probs<(p/2)).*orig_im  + (255-orig_im).*
    uint8(noise_probs>(1-p/2));
subplot(1,2,2);
imshow(noisy_image)

% Create Variables for optimization
y = reshape(noisy_image, [vector_dim,1]);
```

```
data = cast(reshape(dataset,[num_im-1, vector_dim]),'double')';
[U,S,V] = svd(data);
disp("SVD complete...");
r = 35;
psi = U(:,1:r);
% S_tilde = S(1:r,1:r);
% V_tilde = V(:,1:r)';
% psi = U_tilde;
%data_approx = U_tilde*S_tilde*V_tilde;
p = r;
shape_C = [p,vector_dim];
data_size = size(psi);

%% Find C via CVX
if run_CVX
    tStart_CVX = tic;
    cvx_begin
    variable C(shape_C(1), shape_C(2));
    minimize(norm(C*psi,'fro') + norm(C,1));
    subject to
         C*psi - eye(p) == semidefinite(p);
        %C*ones(num_im-1,1) == ones(p,1)
    cvx_end
    tEnd_CVX = toc(tStart_CVX);
% Visualize reconstruction
%C = C.*(abs(C)>1e-3);
theta = C*psi;
end


%% Find C via ADMM
if run_ADMM
    tStart_ADMM = tic;
    disp("Running ADMM Optimization...");
    %Initialize variables
    Theta = randn(shape_C(1));
    Z = randn(shape_C(1));
    C = randn(shape_C);
    B = randn(shape_C);
    Y = randn(shape_C);

    %constants
    gamma = 1e-4;
    t = data_size(1)*data_size(2)/(4*sum(abs(psi(:))));
    lambda = 1/sqrt(max(data_size));
    tolerance = 1e-7;
    tStart_MP = tic;
    H = pinv(psi*psi'+t*eye(data_size(1)));
    tEnd_MP = toc(tStart_MP);
    disp(['MP inverse complete...(', num2str(tEnd_MP), ' seconds)']);
    count = 0;
    while((norm(Theta-C*psi,'fro')> tolerance*norm(C*psi,'fro') ||...
            norm(B-C,'fro') > tolerance*norm(C,'fro'))...
            && count <1000)
        C = (t*Theta*psi'+t*B-Z*psi'-Y)*H/(2+t);
        Theta = P_posdef(C*psi +Z/t,gamma);
```

```
        B = prox_l1(C+Y/t,1/t);
        Z = Z+t*(C*psi-Theta);
        Y = Y+t*(C-B);
        if ~mod(count,10)
            disp(['ADDM itter: ', num2str(count)]);
        end
        count = count+1;

    end

    tEnd_ADMM = toc(tStart_ADMM);
    disp(['ADMM Algorithm Time: ', num2str(tEnd_ADMM)]);

    % Visualize results
    [M,I] = max(C);
    C_prime = zeros(shape_C);
    index = sub2ind(shape_C, [1:r],I);
    C_prime(index)=1;
    Theta_prime = C_prime*psi;
    measurement = orig_im_vec(I);
    x = Theta_prime\cast(measurement,'double');
    face_recon = psi*x;
    face_recon_scale = face_recon +abs(min(min(face_recon)));
    face_recon_scale = face_recon_scale*(255/max(max(face_recon_scale)));
    red_ch = orig_im_vec;
    green_ch = orig_im_vec;
    red_ch(I) = 255;
    green_ch(I) =0;
    rgb_im = cat(3,reshape(red_ch, im_size), reshape(green_ch, im_size), reshape(green_ch,im_size));
    figure;
    subplot(1,3,1)
    imshow(orig_im);
    subplot(1,3,2)
    imshow(rgb_im);
    subplot(1,3,3)
    imshow(uint8(face_recon_scale));
end


function proj_x = P_posdef(X,gamma)
    [V,D] = eig(X);
    D = diag(max(diag(D),gamma));
    proj_x = V*D*pinv(V);
end
function prox_x = prox_l1(X,t)
    prox_x = sign(X).*max(abs(X)-t,zeros);
end
```

The lines below contain all the code written with respect to the RPCA problem.

```
%%
 %  File: RPCA.m
 %  Author: Thomas Kost
 %
 %  Date: 17 May 2022
 %
```

```
%  @brief use of RPCA for noise reduction
%
clc, clear all, close all;
%% Run Variables:
run_CVX = false;
run_ADMM = true;
run_PDHG = true;
run_noise_result = false;
%% Load Data:
im_paths = dir(fullfile('CroppedYale\yaleB01\', '*0.pgm'));
num_im = numel(im_paths);
im_size = size(imread(fullfile(im_paths(1).folder,im_paths(1).name)));
dataset = zeros(num_im, im_size(1),im_size(2),'uint8');
for i = 1:num_im
    dataset(i,:,:) = imread(fullfile(im_paths(i).folder,im_paths(i).name));
end
vector_dim = im_size(1)*im_size(2);
data = cast(reshape(dataset,[num_im, vector_dim]),'double')';
data_size = size(data);
disp("Data Read in...");

% Set Lambda
lambda =1/sqrt(max(data_size));
rand_index = randi([1 num_im],1,1);

%% CVX
if run_CVX
    disp("Running CVX Optimization...")
    tStart_CVX = tic;
    cvx_begin
    variables L(data_size(1) data_size(2)) S(data_size(1) data_size(2))
    minimize(norm_nuc(L) +lambda*norm(S,1));
    subject to
        L+S==img_dbl
    cvx_end
    tEnd_CVX = toc(tStart_CVX);
end

%% ADMM
if run_ADMM
    disp("Running ADMM Optimization...");
    L = zeros(data_size);
    S = zeros(data_size);
    Z = zeros(data_size);
    X = data;
    tolerance = 1e-7;
    t = data_size(1)*data_size(2)/(4*sum(abs(X(:))));
    count=0;
    tStart_ADMM = tic;

    while(norm(X-L-S,'fro')>tolerance*norm(X,'fro') && count <1000)
        L = prox_nuc(X-S+Z/t,1/t);
        S = prox_l1(X-L+Z/t,lambda/t);
        Z = Z+ t*(X-L-S);
        if ~mod(count,10)
            disp(['ADDM itter: ', num2str(count)]);
```

```
        end
        count = count+1;
    end
    tEnd_ADMM = toc(tStart_ADMM);
    disp(['ADMM Algorithm Time: ', num2str(tEnd_ADMM)]);
    % Visualize ADMM
    % Pick a random image
    img = reshape(dataset(rand_index,:,:), im_size);
    L_img = uint8(reshape(L(:,rand_index), im_size));
    S_img = uint8(reshape(S(:,rand_index), im_size));
    RPCA_result = figure();
    subplot(1,3,1);
    imshow(img);
    subplot(1,3,2);
    imshow(L_img);
    subplot(1,3,3);
    imshow(S_img);
end
if run_PDHG
    tStart_PDHG = tic;
    disp("Running ADMM Optimization...");
    L = zeros(data_size);
    L_prev = zeros(data_size);
    Z = zeros(data_size);
    X = data;
    tolerance = 1e-12;
    t = data_size(1)*data_size(2)/(4*sum(abs(X(:))));
    count=0;
    delta=inf;
    while(delta>tolerance && count <1000)
        L_prev = L;
        L = prox_nuc(X-S+Z/t,1/t);
        Z = prox_g_star(Z+ t*(2*L-L_prev),X,t,lambda);
        if ~mod(count,10)
            disp(['PDHG itter: ', num2str(count)]);
        end
        count = count+1;
        primal = norm_nuc(L) +lambda*norm(X-L,1);
        dual = trace(Z'*X); %rest is indicator functions
        delta = norm(L-L_prev,'fro')/norm(L,'fro');
    end
    tEnd_PDHG = toc(tStart_PDHG);
    disp(['PDHG Algorithm Time: ', num2str(tEnd_PDHG)]);
    % Visualize results
    S = X-L;
    img = reshape(dataset(rand_index,:,:), im_size);
    L_img = uint8(reshape(L(:,rand_index), im_size));
    S_img = uint8(reshape(S(:,rand_index), im_size));
    PDHG_result = figure();
    subplot(1,3,1);
    imshow(img);
    subplot(1,3,2);
    imshow(L_img);
    subplot(1,3,3);
    imshow(S_img);
end
```

```

if run_noise_result
    ;
end
function prox_x = prox_nuc(X,t)
    [U,S,V] = svd(X,'econ');
    [n,m] = size(S);
    S = max(S-t,0);
    prox_x = U*S*V';
end
function prox_x = prox_l1(X,t)
    prox_x = sign(X).*max(abs(X)-t,zeros);
end
function prox_x = prox_g_star(Y,X,t,lambda)
        prox_x = Y-(t/lambda)*(X +sign((lambda/t)*Y-X).*max(...
            abs((lambda/t)*Y-X)-(lambda^2/t),zeros));
end
```