AMRR-LMV Face Identification Example

February 23, 2018

Input Database

The database has 1640 faces, which are all 50 by 50 gray- scale images with .png suffix, numbering 1 to 1640. The data consists of two parts,number 1 to 640 represents the data from Yale Face Extended B data collection and which have been sparsely corrupted. There are 10 persons and each person has up to 64 pictures out of 640. There are also 1000 spurious pictures from a yearbook database, to augment the database, but the pictures itself are clean.

The input data matrix X^T is thus a 2500×1640 dense matrix (We convert the grayscale images into float between 0 and 1.). y is a 2500 dimension row vector, k represents numbers of corrupted elements in the data and s^* is the sparsity constraint of w.

Solution:
The overall optimization problem is:

$$\min_{w \in \mathbb{R}^p, b \in \mathbb{R}^n, \|b\|_0 \leq k} \|y - X^T w - b\|$$

where X^T is the input data matrix, and y is the input image (vectorized) and b is the noise under the sparse reconstruction. The optimal solution using the AM_RR algorithm :

We denote the AM_RR algorithm (without a sparsity case) as algorithm A, which implements step 3 and step 4 as above. The step 4 in AM_RR is equivalent to a one step Hard Thresholding(HT). The hard thresholding operator, in short can be defined as below:

$$HT(\mathbf{v}; k) = \{i \in [n] \mid \sigma_{\mathbf{v}}^{-1}(i) \leq k\}$$

where $\sigma_{\mathbf{v}}$ is a permutation such that

$$|\mathbf{v}_{\sigma_{\mathbf{v}}(1)}| \leq |\mathbf{v}_{\sigma_{\mathbf{v}}(2)}| \leq ... \leq |\mathbf{v}_{\sigma_{\mathbf{v}}(n)}|$$

this is equivalent to say that we only preserve the set of index whose corresponding components have lowest k in magnitude.

Algorithm B, is implemented by applying a sparsity constraint on w, i.e. step 3 in AM_RR above is changed into the following optimization problem:

$$\min_{w \in \mathbb{R}^p, \|w\|_0 \leq s^\star} \sum_{i \in S_t} (y_i - x_i^T w)^2$$

Algorithm 11 AltMin for Robust Regression (AM-RR)

Input: Data $X, \mathbf{y}$, number of corruptions k

Output: An accurate model $\widehat{\mathbf{w}} \in \mathbb{R}^p$

1: $\mathbf{w}^1 \leftarrow \mathbf{0}$, $S_1 = [1 : n-k]$

2: **for** $t = 1, 2, \ldots$ **do**

3: $\quad \mathbf{w}^{t+1} \leftarrow \arg\min_{\mathbf{w} \in \mathbb{R}^p} \sum_{i \in S_t} (y_i - \mathbf{x}_i^\top \mathbf{w})^2$

4: $\quad S_{t+1} \leftarrow \arg\min_{|S|=n-k} \sum_{i \in S} (y_i - \mathbf{x}_i^\top \mathbf{w}^{t+1})^2$

5: **end for**

6: **return** $\mathbf{w}^t$

Figure 1: AM_RR algorithm

where $s^\star$ is the sparsity constant that is chosen. The sample code below uses a Backtracking Iterative Hard Thresholding Algorithm (BIHT) to solve the minimization problem with a sparsity constraint on signal w. The codes of the two algorithms are listed below.

The IHT aims at solving a least square problem with sparsity constraint on the signal (w in our notation). The iteration step is:

$$w_{t+1} = HT(w_t + \mu X_S^T (y - X_S w_t)), w_0 = \mathbf{0}$$

if we rewrite step 3 in this matrix form manner:

$$\min_{w \in \mathbb{R}^p, \|w\|_0 \leq s^\star} \|y - X_S^T w\|$$

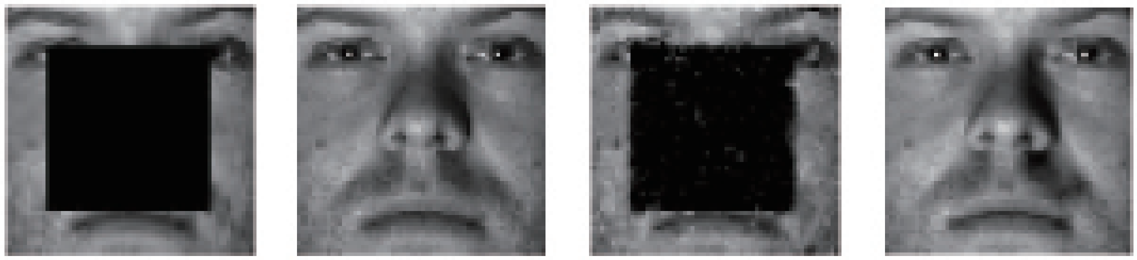
IHT is proved to be convergent if $\|X_S\|_2 \leq 1$.

Results and Analysis

The recovery results are shown in following, we have three test images and the result of two different implementations, i.e. Algorithm A and Algorithm B. We manually fix $s^\star = 20$ for all three tests.

The result shows that, visually, Algorithm A does not recover the target image, while recovery does occur when applying algorithm B.

Implementation Details



(a) input (b) ground truth (c) Result of Algorithm A (d) Result of Algorithm B

Figure 2: 40% corruption recovery. (a) is input image and (b) is ground truth. (c) ($RMSD = 0.2782$) refers to the result of Algorithm A (d) ($RMSD = 0.0686$) refers to the result of Algorithm B.

(a) input (b) ground truth (c) Result of Algorithm A (d) Result of Algorithm B

Figure 3: 40% corruption recovery. (a) is input image and (b) is ground truth. (c) ($RMSD = 0.1126$) refers to the result of Algorithm A (d) ($RMSD = 0.0125$) refers to the result of Algorithm B.

```
function [w, complete] = AM_RR_OLS(X,y,k,tol)
% AM_RR_OLS  Solves Robust Recovery problem with given data matrix
%   Input :
%       X   - Data matrix (n * p in this case,already transposed)
%       y   - Input vector
%       k   - Number of corrupted components
%       tol - Error tolerance, used to determining iteration steps
%   Output :
%       w   - Recovered signal (y = X*w)
%       complete - Recovered vector, i.e. X*w
%   See also AM_RR,BIHT

[n,p] = size(X);

% Initialization
w = zeros(p,1);
S = 1:1:n;
iteration_step = max(10,ceil(1/tol)); % Set up the iteration step.
size(X)
for t=1:iteration_step
    % Solve out the optimization problem using ordinary least square
    w = pinv(X(S,:))*y(S);
```

(a) input (b) ground truth (c) Result of Algorithm A (d) Result of Algorithm B

Figure 4: 40% corruption recovery. (a) is input image and (b) is ground truth. (c) ($RMSD = 0.1870$) refers to the result of Algorithm A (d) ($RMSD = 0.0726$) refers to the result of Algorithm B.

```
    % Calculate the residual
    r =  abs(y - X*w);
    % Hard Thresholding operation
    % Pick first n-k element as uncorrupted recovery
    [~,I] = sort(r);
    S = sort(I(1:n-k));
end

complete = X*w;

end
```

Listing 1: matlab code for the AM_RR algorithm A, without sparsity constraint on w

```matlab
function [w,complete] = AM_RR(X,y,k,tol,s0)
% AM_RR  Solves Robust Recovery problem for a given data matrix
%   Input :
%       X   - Data matrix (n * p in this case)
%       y   - Input vector
%       k   - Number of corrupted components
%       tol - Error tolerance, used to determining iteration steps
%               s0  - Sparsity constant w.r.t. w
%   Output :
%       w   - Recovered signal (y = X*w)
%       complete - Recovered vector, i.e. X*w
%   See also AM_RR,BIHT
[n,p]=size(X);

% Initialization
w=zeros(p,1);
S = 1:1:n;
iteration_step = max(10,ceil(1/tol)); % Set iteration_step

for t=1:iteration_step
    % Using BIHT algorithm to find solution with sparsity constraints
    w = BIHT(X(S,:),s0,y(S),0.01);
    % Calculate the residual
    r=  abs(y - X*w);
    % Hard Thresholding operation
    % Pick first n-k element as uncorrupted recovery
    [~,I]= sort(r);
    S= sort(I(1:n-k));
end

% This is the linear interpolation we have calculated
% which should give you the recovered target image
complete = X*w;

end
```

Listing 2: matlab code for the AM_RR algorithm B, with a sparsity constraint on w applied

```matlab
function sol = BIHT(A,K,y,mu)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%Author                    Zhang Cheng, Yang Hairong
%%Modified time             2010/07/23
%%function                  Code for Backtracking Iterative Hard Thresholding
%% A  - Measurement matrix
%% K  - sparsity level
%% y  - measurement vector
%% mu - parameter as in IHT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[M,N]     = size(A);
s         = zeros(N,1);
Ps        = zeros(M,1);
```

```
Residual    = y;
MAXITER=200;
Phi=A;

done=0;
iter=1;


while ~done
        theta              =   s+mu*Phi'*(y-Phi*s);
        [ssort sortind]    =   sort(abs(theta),'descend');
        theta(sortind(K+1:end)) =   0;

        Index_S=find(s~=0);
        Index_theta=find(theta~=0);
        activeset=union(Index_S,Index_theta);

        Phi_x = Phi(:,activeset);
        beta=inv(Phi_x'*Phi_x)*Phi_x'*y;

        [bsort bsortind]    =   sort(abs(beta),'descend');
        beta(bsortind(K+1:end)) =   0;

        s(activeset) = beta;

     iter=iter+1;
     res=y-Phi_x*beta;
     err=norm(res);

     if iter >= MAXITER
         display('Stopping. Maximum number of iterations reached!')
         done = 1;
     end

     if err < 1e-6
         display('Success!')
         done = 1;
     end
end

sol = zeros(N,1);
sol(activeset) = beta;
```

Listing 3: matlab code for Backward Iterative Hard Thresholding(BIHT) algorithm

- Why does Algorithm A show only a little improvement for recovery?
- Why can we recover the target image via Algorithm B, just by adding a sparsity constraint on w?
- What about the error norm $\|w^\star - w\|_2$?

To answer these questions, we need to first define SSC and SSS properties.

Definition 1. (SSC and SSS Properties) A matrix $X \in \mathbb{R}^{n\times p}$ satisfies the Subset Strong Convexity Property (resp. Subset Strong Smoothness Property) at level γ with strong convexity constant λ_γ (resp. strong smoothness constant Λ_γ) if the following holds:

$$\lambda_\gamma \leqslant \min_{S \in S_\gamma} \lambda_{min}(X_S^T X_S) \leqslant \max_{S \in S_\gamma} \lambda_{max}(X_S^T X_S) \leqslant \Lambda_\gamma$$

where $S_\gamma = \{S \subset [n] \mid |S| = \gamma \times n\}$, and X_S represents the corresponding blocks of the data matrix X.

In theorem 3 of [Bhatia, 2015], they claimed if $\tilde{X} = \Sigma_0^{-1/2} X$ (here Σ_0 is an invertible matrix, it is used to explain the behavior of normalization of Gaussian variables in the paper) satisfies SSC and SSS properties at level γ with constant λ_γ and Λ_γ respectively, and if $\frac{(1+\sqrt{2})\Lambda_{\beta \cdot n}}{\lambda_{(1-\beta)\cdot n}} < 1$, then AM_RR algorithm converges.

Obviously, as $|S|$ increases (i.e. β increases), the constant $\frac{\Lambda_{\beta \cdot n}}{\lambda_{(1-\beta)\cdot n}}$ is increasing. In the paper they indicate: if the data matrix is generated by *i.i.d.* multivariate Gaussian and $\beta < \frac{1}{65}$, then the recovery converges with high probability.

Although we cannot directly compute Σ_0 or $\lambda_\gamma, \Lambda_\gamma$ numerically, we can still analyze the matrix $X^T X$. Let us calculate the SVD decomposition of $X^T X$, and we have:

$$\lambda_{max}(X^T X) = 1132592 \quad \lambda_{min}(X^T X) = 0.0035$$

This means the condition number is over 10^8, which is a numerically unstable matrix. We do not expect it to meet the convergence rate if we pick $\beta = 0.10, 0.15$ or 0.40. If you directly apply linear regression , as proposed in Algorithm A, then what you get is a linear combination of all possible 1640 images, and these pictures are not even for the same person! You can try, however, if you only contaminate around 1% pixels in a image, it should somehow give you a good recovery result.

To alleviate this problem, in theorem 9 of [Bhatia, 2015], they proposed that, in high dimensional case, by adding sparsity constraint on w, by rewriting the condition as

$$s \geq 32 \frac{\lambda_{(1-\beta, 2s+s^\star)}}{\Lambda_{(1-\beta, 2s+s^\star)}}$$

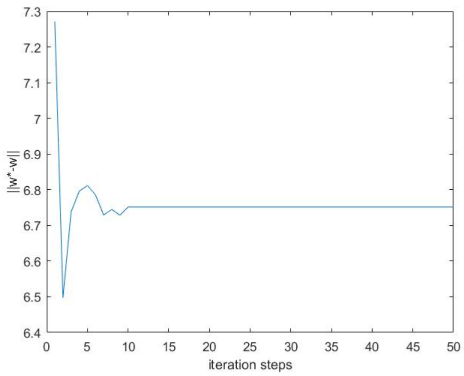
and

$$\frac{\lambda_{(1-\beta, s+s^\star)}}{\Lambda_{(1-\beta, s+s^\star)}} < \frac{1}{4}$$

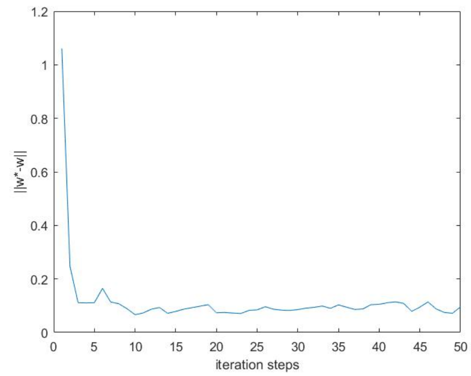
one can ensure the convergence. It is better than the guarantee in theorem 3 since we now only consider the SSS and SSC property with sparse vector w, this is equivalent to say to consider the singular value on each sparse subset of each block matrix $X_S^T X_S$, which will give you a smaller condition number in practice.

Lastly, to show that the robust recovery with sparsity constraints indeed converges (although not rigorously proved) , we plot the error norm between w in each iteration and ground truth $w^\star$. In this empirical test, the ground truth $w^\star$ satisfies: $\|w^\star\|_0 = 1, \|w^\star\|_1 = 1$, i.e. only one component shall be equal to 1.

$\square$

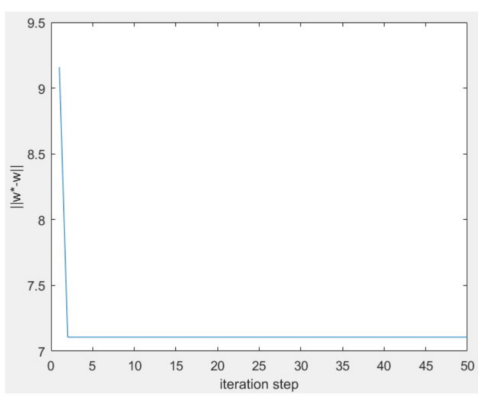


(a) Error norm plot for Algorithm A on Figure 3 (c)

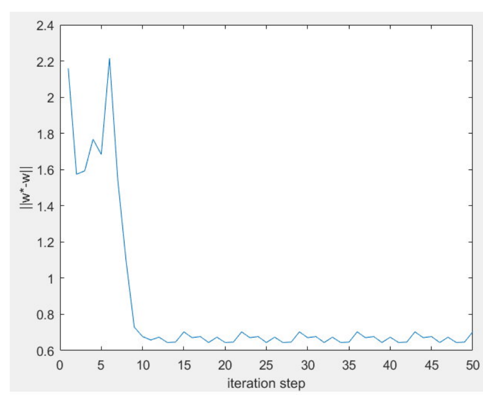


(b) Error norm plot for Algorithm B on Figure 3 (d)

Figure 5: Error norm plot, the horizontal axis represents the number of iteration steps while the vertical axis represents $\|w^\star - w\|^2$.



(a) Error norm plot for Algorithm A on Figure 4 (c)



(b) Error norm plot for Algorithm B on Figure 4 (d)

Figure 6: Error norm plot, the horizontal axis represents the number of iteration steps while the vertical axis represents $\|w^\star - w\|^2$.