One-Bit Compressed Sensing using the Soft-Thresholded Landweber Iteration

Andrew E. Yagle
Department of EECS, The University of Michigan, Ann Arbor, MI 48109-2122

Abstract— **One-bit compressed sensing reconstructs a sparse signal from only signs of linear combinations of the signal values. The problem formulates easily as a linear programming problem, but simpler methods are desirable. The binary iterative hard thresholded Landweber (BIHT) algorithm requires an oracle to know the exact sparsity of the signal. We reformulate the one-bit problem with noisy observation as a non-negative least-squares problem and solve the latter using the Landweber iteration with a non-negativity constraint, which becomes a soft-thresholded Landweber iteration with both shrinkage and thresholding. We apply this to image reconstruction from two bits of Fourer phase without an image support constraint.**

Phone: 734-763-9810. Fax: 734-763-1503.
Email: aey@eecs.umich.edu.

I. INTRODUCTION

A. Problem Statement

We are given as data only the *signs* $\{s_i\}$

$$s_i = \text{sign}[y_i] = \begin{cases} +1 & \text{if } y_i > 0 \\ -1 & \text{if } y_i < 0 \end{cases} \qquad (1)$$

of the M linear combinations y of an unknown x

$$y = Ax \qquad (2)$$

- x is an unknown sparse (mostly zero) N-vector;
- A is a known $M \times N$ full-rank matrix with $M \geq N$;
- Each component x_i of x is a random variable with

$$f_{x_i}(X) = C \underbrace{e^{-\lambda|X|}}_{\text{SPARSE}} \underbrace{e^{-\mu X^2}}_{\text{SMALL}}. \qquad (3)$$

The Gaussian prior penalizes too-large values of x. The Laplacian (two-sided exponential) sparsifies x. The goal is to reconstruct x from the signs s_i of y_i, i.e., from only one (sign) bit of each value y_i of y.

B. Problem Reformulation

We assume that the $\{s_i\}$ are independent and equally likely to be ± 1, and let matrix S=diag$[s_i]$. Premultiplying (2) by S gives (here $|y|$ means $[|y_i|]$)

$$z = |y| = Sy = (SA)x. \qquad (4)$$

- z_i=$|y_i|$. The given data is precisely that $z \geq 0$;
- SA is still a known full-rank matrix (S is known).

The cost functional to be minimized is then

$$\mathcal{E} = \underbrace{||\mathbf{01} + z - (SA)x||_2^2/2}_{\text{EQUALITY}} + \underbrace{\lambda||x||_1}_{\text{SPARSE}} + \underbrace{\mu||x||_2^2}_{\text{RIDGE}} \qquad (5)$$

with the additional constraint $z \geq 0$ (which is the data) and where we have defined the usual two norms

$$||x||_1 = \sum_{i=1}^{N} |x_i|; \quad ||x||_2^2 = \sum_{i=1}^{N} x_i^2; \quad \mathbf{01} = \begin{bmatrix} \mathbf{0}\}M-1 \\ 1\} \quad\quad 1 \end{bmatrix}.$$

$\mathcal{E}$ is the one-bit compressed sensing version of the elastic net (LASSO plus ridge) cost functional. The ridge provides regularization when A is near-singular.

The signs s_i of y_i only determine x to a positive scale factor. To keep x from being driven to zero, the last row of the first term of (5) is $1+z_M-s_M a_M^T x$, where a_M^T is the last row of A and $z_M \geq 0$ makes $s_M a_M^T x \geq 1$ and keeps $x \neq 0$. This has been done previously by constraining $||x||$=1, but this is a non-convex constraint, which complicates optimization.

The ℓ_1-norm penalty term $||x||_1$ penalizes small deviations of the elements of x from zero. A considerable amount of research since 2000 has proven what the geophysical community has observed since the 1960s: The ℓ_1 norm produces sparse solutions.

The ℓ_2-norm penalty term $||x||_2^2$ does not penalize small deviations, since its slope is zero at zero. But it does penalize large deviations more heavily than the ℓ_1 norm, and thus stabilizes the solution in the presence of noise. Use of an ℓ_2 norm penalty term for this purpose is called Tikhonov regularization.

Minimization of $\mathcal{E}$ in the limit $\lambda \to 0$ with μ=0 becomes minimization of $||x||_1$ subject to the equality constraint $\mathbf{01}$+z=$(SA)x$ and the inequality $z \geq 0$. This can be formulated as a linear programming problem and solved using any number of algorithms. Alternatively, any number of convex optimization algorithms can be applied to the minimization of $\mathcal{E}$.

However, in many applications A is not represented as a matrix, but as a sequence of operations, such as wavelet or fast Fourier transforms. In this case, Ax and $A^T y$ can be computed much more quickly than a typical matrix-vector multiplication, e.g., $N \log N$. This motivates use of the Landweber iteration below.

II. Reformulation of $\mathcal{E}$ Minimization as a Non-Negative Least Squares Problem

First, we use the usual procedure of defining the positive x^+ and negative x^- parts of x as

$$x_i^+ = \begin{cases} +x_i & \text{if } x_i \geq 0 \\ 0 & \text{if } x_i \leq 0 \end{cases} \quad x_i^- = \begin{cases} -x_i & \text{if } x_i \leq 0 \\ 0 & \text{if } x_i \geq 0 \end{cases} \geq 0. \tag{6}$$

Then x_i^+ and x_i^- are related to x_i and $|x_i|$ by

$$\begin{aligned} x &= x^+ - x^- \\ ||x||_1 &= \sum_{i=1}^{N}(x_i^+ + x_i^-) \\ ||x||_2^2 &= ||x^+||_2^2 + ||x^-||_2^2 \end{aligned} \tag{7}$$

Now consider the $(N+M) \times (2N+M)$ problem

$$\underbrace{\begin{bmatrix} \mathbf{01} \\ \frac{-\lambda}{\sqrt{2\mu}}\mathbf{1} \end{bmatrix}}_{\tilde{y}} = \underbrace{\begin{bmatrix} SA & -SA & -I \\ \sqrt{2\mu}I & \sqrt{2\mu}I & 0 \end{bmatrix}}_{\tilde{A}} \underbrace{\begin{bmatrix} x^+ \\ x^- \\ z \end{bmatrix}}_{\tilde{x}} \tag{8}$$

where we have defined the two column vectors

$$\mathbf{01} = [\underbrace{0 \dots 0}_{M-1} \underbrace{1}_{1}]^T; \quad \mathbf{1} = [\underbrace{1, 1 \dots 1}_{N}]^T \tag{9}$$

and we impose constraints $x_i^+ \geq 0$; $x_i^- \geq 0$; $z \geq 0$.

The squared ℓ_2 error is then (note that $x_i^+ x_i^- = 0$)

$$\begin{aligned} ||\tilde{y} - \tilde{A}\tilde{x}||_2^2 &= ||(SA)(x^+ - x^-) - z - \mathbf{01}||_2^2 \\ &+ ||\sqrt{2\mu}(x^+ + x^-) + \frac{\lambda}{\sqrt{2\mu}}||_2^2 \\ &= ||(SA)x - z - \mathbf{01}||_2^2 \\ &+ 2\mu||x||_2^2 + 2\lambda||x||_1 + \frac{N\lambda^2}{2\mu} \\ &= 2\mathcal{E} + \frac{N\lambda^2}{2\mu}. \end{aligned} \tag{10}$$

The final term in (10) does not affect the argmax, so computing the *non-negative least-squares* solution to (8) minimizes the cost function $\mathcal{E}$. Also note that

$$(SA)^T(SA) = A^T(S^TS)A = A^TA \tag{11}$$

independent of the diagonal matrix S of signs of y_i.

Minimization of $\mathcal{E}$ is a non-negative least-squares problem. It can be solved using Matlab code like:

```
clear;M=30;N=10;L=.0001;E=.0000005;
e=sqrt(2*E);randn('state',1);X(3)=1;X(7)=-2;
A=randn(M,N);X(N)=0;X=X';Y=A*X;
S=diag(sign(Y));AA=[S*A -S*A -eye(M)];
AA=[AA;e*eye(N) e*eye(N) zeros(N,M)];
YY=[zeros(M-1,1);1;-L/e*ones(N,1)];
Z=lsnonneg(AA,YY);[X Z(1:N)-Z(N+1:2*N)]
```

III. Solution of the Non-Negative Least-Squares Using Landweber

A. Review of Landweber Iteration

The basic Landweber iteration is

$$x^{k+1} = x^k + A^T(y - Ax), \quad x^0 = 0 \tag{12}$$

where x^k is the estimate of x at the k^{th} iteration. The Landweber iteration can be viewed as a steepest descent algorithm for minimizing the cost function

$$\begin{aligned} f(x) &= (1/2)||y - Ax||_2^2 \\ \nabla f(x) &= -A^T(y - Ax). \end{aligned} \tag{13}$$

The basic steepest descent algorithm is

$$\begin{aligned} x^{k+1} &= x^k - \nabla f(x) \\ &= x^k + A^T(y - Ax). \end{aligned} \tag{14}$$

Here, we use the basic Landweber iteration

$$\tilde{x}^{k+1} = \tilde{x}^k + \tilde{A}^T(\tilde{y} - \tilde{A}\tilde{x}^k) \tag{15}$$

with a non-negativity constraint at each iteration

$$\tilde{x}_i^{k+1} = \max[\tilde{x}_i^{k+1}, 0]. \tag{16}$$

We call this the *non-negative* Landweber iteration.

Since the cost functional $f(x)$ and the non-negativity constraints $x_i^+ \geq 0$ and $x_i^- \geq 0$ are all convex, the non-negative Landweber iteration will converge if the maximum eigenvalue of $\tilde{A}^T\tilde{A} \leq 2$.

B. Application to Present Problem

Substitution of (8) in (15) gives after some algebra

$$\begin{aligned} (x^+)^{k+1} &= (x^+)^k - A^TA[(x^+)^k - (x^-)^k] + A^TS\mathbf{01} \\ &+ A^TSz^k - \lambda\mathbf{1} - 2\mu[(x^+)^k + (x^-)^k] \\ (x^-)^{k+1} &= (x^-)^k + A^TA[(x^+)^k - (x^-)^k] - A^TS\mathbf{01} \\ &- A^TSz^k - \lambda\mathbf{1} - 2\mu[(x^+)^k + (x^-)^k] \\ z^{k+1} &= SA[(x^+)^k - (x^-)^k] - \mathbf{01} \end{aligned} \tag{17}$$

followed by the three non-negativity constraints

$$\begin{aligned} (x^+)_i^{k+1} &= \max[(x^+)_i^{k+1}, 0] \\ (x^-)_i^{k+1} &= \max[(x^-)_i^{k+1}, 0] \\ z_i^{k+1} &= \max[z_i^{k+1}, 0] \end{aligned} \tag{18}$$

so most of the computations in the Landweber iteration are the matrix-vector multiplications Ax and A^TSz, which can often be implemented using a fast algorithm such as the fast Fourier transform, the fast wavelet algorithm or a sparse matrix-times-vector.

C. Convergence of Non-Negative Landweber

$\tilde{A}$ is underdetermined. The nonzero eigenvalues of $\tilde{A}^T\tilde{A}$ are the eigenvalues of $\tilde{A}\tilde{A}^T$, which from

$$\begin{aligned}\tilde{A}\tilde{A}^T &= \begin{bmatrix} SA & -SA & -I \\ \sqrt{2\mu}I & \sqrt{2\mu}I & 0 \end{bmatrix}\begin{bmatrix} A^TS & \sqrt{2\mu}I \\ -A^TS & \sqrt{2\mu}I \\ -I & 0 \end{bmatrix} \\ &= \begin{bmatrix} 2(SA)(A^TS)+I & 0 \\ 0 & 4\mu I \end{bmatrix} \qquad (19)\end{aligned}$$

are double those of $(SA)(A^TS)$ plus one, and 4μ. The non-negative Landweber iteration will converge if μ and all eigenvalues of $(SA)(A^TS)$ are less than $\frac{1}{2}$.

Since SA is overdetermined, the nonzero eigenvalues of $(SA)(SA)^T$ in (19) are the eigenvalues of $(A^TS)(SA)=A^TA$, independent of data S.

D. Derivation of Thresholded Landweber

Let μ=0 and recall $x^k=(x^+)^k-(x^-)^k$ and $-(x^-)^k$ are the negative values of x^k. Then (17) becomes

$$\begin{aligned} x^{k+1} &= x^k - A^TAx^k + A^TS(\mathbf{01}+z^k) \\ &= x^k + A^TS[(\mathbf{01}+z^k) - SAx^k] \\ z^{k+1} &= SAx^k - \mathbf{01} \qquad (20)\end{aligned}$$

followed by shrinkage of $|x^k|$ by λ, thresholding values of $|x^k|<\lambda$ to 0, and imposing non-negative z^{k+1}:

$$\begin{aligned} x_i^{k+1} &= \begin{cases} x_i^{k+1}-\lambda & \text{if } x_i^{k+1} > +\lambda \\ x_i^{k+1}+\lambda & \text{if } x_i^{k+1} < -\lambda \\ 0 & \text{if } |x_i^{k+1}| < \lambda \end{cases} \\ z^{k+1} &= \begin{cases} z^{k+1} & \text{if } z^{k+1} \geq 0 \\ 0 & \text{if } z^{k+1} \leq 0 \end{cases} \qquad (21)\end{aligned}$$

This is similar to soft-thresholded Landweber with

- "Observations" $(\mathbf{01}+z^k)$ instead of y, where the
- Signs of y_i (the data) are included in SA and
- $z^{k+1}\geq 0$ forces $(SA)x^k \geq 0$, which is the data.

E. Orthonormal Columns

Let overdetermined A have orthonormal columns $A^TA=I$. Then the above algorithm simplifies to

$$\begin{aligned} z^{k+1} &= (SA)x^k - \mathbf{01} \\ x^{k+1} &= (A^TS)(z^k+\mathbf{01}) \qquad (22)\end{aligned}$$

along with thresholding and shrinkage steps (21).

If z^k is replaced with z^{k+1} in the second equation, this is similar to an alternating-projections algorithm with a sign constraint in one domain and thresholding and shrinkage (for sparsity) in the other domain.

Multiplication by SA allows the sign constraint to be implemented as a non-negativity constraint. Multiplication by A^TS undoes multiplication by S in SA.

IV. Image Reconstruction from Two Bits of Discrete Fourier Transform Phase

A. Background

The problem of reconstructing a signal or image from two bits of the phase of its Fourier transform arises in phase-shifting digital holography. This reduced bit-depth allows real-time reconstruction by a spatial light modulator, which modulates phase.

Previous algorithms on reconstruction from two bits of Fourier phase have been alternating projection algorithms that impose a support constraint in the spatial domain and a sign constraint on the phase in the Fourier domain. If the support is chosen to be a half-plane, then one bit of phase is sufficient, since the even part of the image with such a constraint determines the entire image. This seems somewhat contrived. Knowledge of one bit of phase amounts to approximate knowledge of the zero crossings of the real part of the Fourier transform, which under mild assumptions determines the real part of the image to an overall scale factor. Two bits of phase amounts to approximate knowledge of the zero crossings of the real and imaginary parts of the Fourier transform, which determine the even and odd parts of the image, respectively, to separate scale factors for each.

Here we replace the support constraint (which is usually not available) with a sparsity constraint, which consists of soft thresholding. This arises directly from the above algorithm. The matrix A is the concatenation of a 2-D discrete Fourier transform and the inverse of a sparsifying operator, such as a wavelet transform. Both of these can be implemented directly as algorithms, so that the matrix A need never be formed or stored. A^H is then an inverse 2-D discrete Fourier transform followed by a wavelet transform, again implemented as algorithms.

B. Problem Formulation

The $N\times N$-point 2-D discrete Fourier transform of the $M\times M$ image $x[m,n]$ is defined as

$$X_{k_1,k_2} = \sum_{m=0}^{M-1}\sum_{n=0}^{M-1} x[m,n]e^{-j\frac{2\pi}{N}(mk_1+nk_2)}. \qquad (23)$$

The inverse 2-D discrete Fourier transform is

$$x[m,n] = \frac{1}{N^2}\sum_{k_1=0}^{N-1}\sum_{k_2=0}^{N-1} X_{k_1,k_2}e^{j\frac{2\pi}{N}(mk_1+nk_2)}. \qquad (24)$$

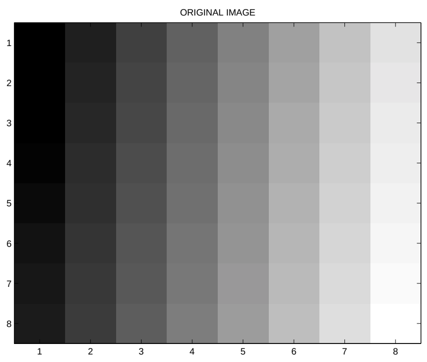
The 2-D DFT is a unitary linear transformation, except for a factor of $\frac{1}{N}$ (these are collected in (24)).

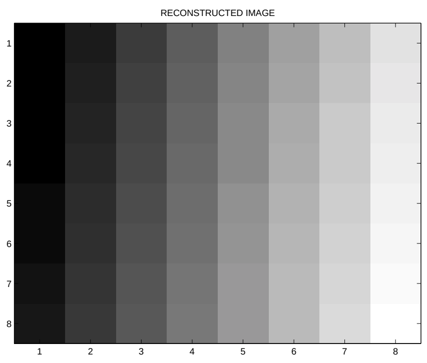
"Two bits of phase" means the signs of $\text{Re}[X_{k_1,k_2}]$ and $\text{Im}[X_{k_1,k_2}]$. This specifies the phase as lying in one of four quadrants, hence "two bit" quantization.

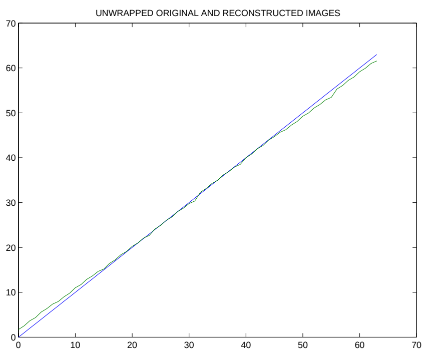
C. Tiny Numerical Example

We deliberately choose a tiny example to make some important points about the problem. The 64×64-point 2-D DFT of the 8×8 image shown was computed, and only the signs of the real and imaginary parts were used as data. Note that half of these are redundant by conjugate symmetry. Three resolutions of the 2-D Haar wavelet transform was used as the sparsifying transform. The shrinkage and thresholding parameter λ=0.0001 and 3000 iterations used.

The reconstructed image visually closely resembles the original image, even though no support or non-negativity constraints were used. A better picture of the reconstruction is the 1-D plot of the unwrapped original and reconstructed images. Note that the reconstruction is off slightly at the two ends. This is because the single scale factor specified (the DC term) only applied to the even part of the image. The odd part is computed to sparsify the overall image, but the lack of a scale factor introduces a slight error.







D. MATLAB Program

```
%Image Reconstruction from 2 Bits of Phase.
%N=image size; M=DFT size; l=lambda; L=#iterations.
clear;N=8;M=64;l=0.0001;L=3000;%Make Image:
x=[0:63]';x=reshape(x,8,8);B(1,1)=sum(sum(x));
%Initializations (and clearing variable spaces):
FX(M,M)=0;SR(M,M)=0;SI(M,M)=0;B(M,M)=0;
FV(M,M)=0;FVR(M,M)=0;FVI(M,M)=0;W0(N,N,1)=0;
W1(N/2,N/2,4)=0;W2(N/4,N/4,4)=0;W3(N/8,N/8,4)=0;
V0(N,N)=0;V1(N/2,N/2)=0;V2(N/4,N/4)=0;V3(N/8,N/8)=0;
%Goal:  Reconstruct image x from SR and SI, where:
Fx=fft2(x,M,M);SR=sign(real(Fx));SI=sign(imag(Fx));
%Reconstruction Algorithm:
X(N,N)=0;for J=1:L;%Initialize, and L=#iterations
%Haar Transform:
W0(:,:,1)=X;
W1(N/2,N/2,4)=0;W2(N/4,N/4,4)=0;W3(N/8,N/8,4)=0;
I11=[1 -1 -1 1]/2;I12=[1 1 -1 -1]/2;
I21=[1 -1 1 -1]/2;I22=[1 1 1 1]/2;
for I=1:4;%1st stage:
A=I11(I)*W0(1:2:N,1:2:N,1)+I12(I)*W0(1:2:N,2:2:N,1);
B=I21(I)*W0(2:2:N,1:2:N,1)+I22(I)*W0(2:2:N,2:2:N,1);
W1(:,:,I)=A+B;end;
for I=1:4;%2nd stage:
A=I11(I)*W1(1:2:N/2,1:2:N/2,1)+I12(I)*W1(1:2:N/2,2:2:N/2,1);
B=I21(I)*W1(2:2:N/2,1:2:N/2,1)+I22(I)*W1(2:2:N/2,2:2:N/2,1);
W2(:,:,I)=A+B;end;
for I=1:4;%3rd stage:
A=I11(I)*W2(1:2:N/4,1:2:N/4,1)+I12(I)*W2(1:2:N/4,2:2:N/4,1);
B=I21(I)*W2(2:2:N/4,1:2:N/4,1)+I22(I)*W2(2:2:N/4,2:2:N/4,1);
W3(:,:,I)=A+B;end;
%End Haar Transform.
%Shrink and Threshold W (Soft Threshold):
W1(abs(W1)<l)=0;W2(abs(W2)<l)=0;W3(abs(W3)<l)=0;
W1(abs(W1)>l)=W1(abs(W1)>l)-l*sign(W1(abs(W1)>l));
W2(abs(W2)>l)=W2(abs(W2)>l)-l*sign(W2(abs(W2)>l));
W3(abs(W3)>l)=W3(abs(W3)>l)-l*sign(W3(abs(W3)>l));
%Inverse Haar Transform:
V3=W3(:,:,1); %3rd stage:
V2(2:2:N/4,2:2:N/4)=(V3+W3(:,:,2)+W3(:,:,3)+W3(:,:,4))/2;
V2(2:2:N/4,1:2:N/4)=(V3-W3(:,:,2)+W3(:,:,3)-W3(:,:,4))/2;
V2(1:2:N/4,2:2:N/4)=(V3+W3(:,:,2)-W3(:,:,3)-W3(:,:,4))/2;
V2(1:2:N/4,1:2:N/4)=(V3-W3(:,:,2)-W3(:,:,3)+W3(:,:,4))/2;
%2nd stage:
V1(2:2:N/2,2:2:N/2)=(V2+W2(:,:,2)+W2(:,:,3)+W2(:,:,4))/2;
V1(2:2:N/2,1:2:N/2)=(V2-W2(:,:,2)+W2(:,:,3)-W2(:,:,4))/2;
V1(1:2:N/2,2:2:N/2)=(V2+W2(:,:,2)-W2(:,:,3)-W2(:,:,4))/2;
V1(1:2:N/2,1:2:N/2)=(V2-W2(:,:,2)-W2(:,:,3)+W2(:,:,4))/2;
%1st stage:
V0(2:2:N,2:2:N)=(V1+W1(:,:,2)+W1(:,:,3)+W1(:,:,4))/2;
V0(2:2:N,1:2:N)=(V1-W1(:,:,2)+W1(:,:,3)-W1(:,:,4))/2;
V0(1:2:N,2:2:N)=(V1+W1(:,:,2)-W1(:,:,3)-W1(:,:,4))/2;
V0(1:2:N,1:2:N)=(V1-W1(:,:,2)-W1(:,:,3)+W1(:,:,4))/2;
%End Inverse Haar Transform.
%Impose Data Sign Constraints:
FV=fft2(V0,M,M);FVR=SR.*real(FV)-B;FVR(FVR<0)=0;
FVI=SI.*imag(FV);FVI(FVI<0)=0;
FV=SR.*(FVR+B)+j*SI.*FVI;
X=real(ifft2(FV));X=X(1:N,1:N);
%Output Figures:
end;figure,plot(0:63,x(:),0:63,X(:))
figure,imagesc(x),colormap(gray)
figure,imagesc(X),colormap(gray)
```