EFFICIENT SPARSE BAYESIAN LEARNING VIA GIBBS SAMPLING

Xing Tan and Jian Li
Dept. of Electrical and Computer Eng., Univ. of Florida, Gainesville, FL, USA, email: tanxing@ufl.edu, li@dsp.ufl.edu

Peter Stoica
Dept. of Information Technology, Uppsala University, Uppsala, Sweden, Email: peter.stoica@it.uu.se

ABSTRACT
Sparse Bayesian learning (SBL) has been used as a signal recovery algorithm for compressed sensing. It has been shown that SBL is easy to use and can recover sparse signals more accurately than the well-known Basis Pursuit (BP) algorithm. However, the computational complexity of SBL is quite high, which limits its use in large-scale problems. We propose herein an efficient Gibbs sampling approach, referred to as GS-SBL, for compressed sensing. Numerical examples show that GS-SBL can be faster and perform better than the existing SBL approaches.

Index Terms— Compressed Sensing, Sparse Bayesian Learning, Gibbs Sampling

1. INTRODUCTION
We denote vectors by boldface lowercase letters and matrices by boldface uppercase letters. Other notations that we use throughout this paper are given in Table 1.

<table>
<thead>
<tr>
<th>Notations</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbf{A}$</td>
<td>transpose of a vector or a matrix</td>
</tr>
<tr>
<td>$\mathbf{B}$</td>
<td>conjugate transpose of a vector or a matrix</td>
</tr>
<tr>
<td>$| \cdot |_q$</td>
<td>$q$-norm</td>
</tr>
<tr>
<td>$\mu_{k,l}$</td>
<td>the $k,l$th element of the vector $\mu$</td>
</tr>
<tr>
<td>$\Sigma_{k,l}$</td>
<td>the $k,l$th element of the matrix $\Sigma$</td>
</tr>
<tr>
<td>$\text{diag}(\mathbf{v})$</td>
<td>A diagonal matrix whose diagonal entries are the elements of the vector $\mathbf{v}$</td>
</tr>
</tbody>
</table>

In the past few decades, transform coding has been widely used for data compression. Denote the signal to be compressed as an $N$-dimensional real-valued vector $\mathbf{s}$. Let $\mathbf{B}$ be an orthogonal basis matrix which satisfies $\mathbf{B}^T \mathbf{B} = \mathbf{B} \mathbf{B}^T = \mathbf{I}$ and define the transform coefficient vector $\mathbf{x}$ as $\mathbf{x} = \mathbf{B} \mathbf{s}$. If $\mathbf{x}$ can be approximated by a $K$-sparse vector $\hat{\mathbf{x}}$ (i.e., $\hat{\mathbf{x}}$ contains no more than $K$ non-zero elements and usually $K \ll N$), then we only need to encode the $K$ non-zero elements and their positions in $\hat{\mathbf{x}}$ instead of all $N$ elements of $\mathbf{s}$. Transform coding is inefficient, because it requires collecting $N$ data samples and computing $N$ transform coefficients in which only the $K$ largest ones are retained and encoded. Fortunately, with the recent advent of compressed sensing techniques (see, e.g., [1]), we can recover the original signal $\mathbf{s}$ from much fewer samples. In compressed sensing, the signal is measured through a linear process that can be modeled as projecting the signal $\mathbf{s}$ onto a set of vectors $\{\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_M\}$ to obtain the following samples:

$$y_m = \mathbf{m}_m^T \mathbf{s} + \epsilon_m, \quad m = 1, 2, \ldots, M,$$

where $\epsilon_m$ is the measurement noise. If we define the data vector $\mathbf{y} = [y_1, y_2, \ldots, y_M]^T$, the measurement matrix $\mathbf{M} = [\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_M]^T$ and the noise vector $e = [\epsilon_1, \epsilon_2, \ldots, \epsilon_M]^T$, then $\mathbf{y}$ can be expressed as $\mathbf{y} = \mathbf{M} \mathbf{s} + e$. Using the fact that $s = \mathbf{B} \mathbf{x}$, we obtain the following data model:

$$\mathbf{y} = \mathbf{A} \mathbf{x} + e,$$

where $A \triangleq MB$ is an $M \times N$ matrix.

To obtain a sparse solution of (1), an $l_1$-norm based optimization approach, referred to as Basis Pursuit (BP), was proposed in [2]:

$$\hat{x} = \arg \min_{x} \|x\|_1 \quad \text{subject to} \quad \|y - A \mathbf{x}\|_2 < \delta,$$

where $\delta$ is a user parameter. An alternative $l_1$-norm based optimization is

$$\hat{x} = \arg \min_{x} \|y - A \mathbf{x}\|_2 + \lambda \|x\|_1.$$  

From a Bayesian point of view, (3) is equivalent to the maximum a posteriori approach for estimating $\mathbf{x}$ in (1), assuming that $\epsilon$ is a normal random vector and $\mathbf{x}$ has a Laplacian a priori distribution. Many efficient methods for solving (3) have been proposed recently, including the iterative shrinkage/thresholding (IST) see, e.g., [3]), and gradient projection for sparse reconstruction (GPSR, e.g., [4]). All these methods depend on the fast computation of the product of $A$ and an arbitrary vector. If the complexity of the matrix vector product is $O(L)$, then IST and GPSR also have $O(L)$ complexity. Hence, when $\mathbf{A}$ is a fast Fourier Transform (FFT) matrix, a Discrete Cosine Transform (DCT) matrix, a Wavelet Transform matrix or a sparse matrix, then the complexities of IST and GPSR are nearly linear to $N$. Therefore, they are highly efficient in solving large-scale compressed sensing problems. Another efficient sparse signal recovery method is the so-called Compressive Sampling Matching Pursuit (CoSaMP, [5]), which also has $O(L)$ complexity.

One can also obtain a sparse estimate of $\mathbf{a}$ via sparse Bayesian learning (SBL), which was originally proposed in the machine learning literature [6]. The model used in SBL is a three-stage hierarchical Bayesian model. The parameters in the Bayesian model were obtained by using the maximum a posteriori (MAP) criterion. However, the MAP problem cannot be solved by setting the derivative
of the cost function to zero, since the derivative is highly nonlinear. Hence, the author proposed two iterative approaches: a direct differentiation approach and an Expectation Maximization (EM) approach (see Appendix A of [6]). The vector $x$ can be estimated by computing its \emph{a posteriori} mean during the iterative process. A greedy SBL method, referred to as the Fast Marginal Likelihood Maximization algorithm was proposed in [7] and later on applied to compressed sensing in [8] under the name Bayesian Compressive Sensing (BCS). In [8], the authors show that in some cases, the greedy SBL gives a more accurate estimate of the signal than BP algorithm in terms of the reconstruction error defined by $\text{err} = 10 \log_{10} \left\{ \frac{||x - \hat{x}||^2}{||x||^2} \right\}$ dB.

A major advantage of SBL over the $l_1$-norm based optimization is that the former provides not only a point estimate of $x$, but also the \emph{a posteriori} covariance matrix of $x$, which measures the accuracy of the point estimate. However, both the EM-based SBL and the direct differentiation based SBL have high computational complexities of $O(M^2 N)$ (see [6]). The Fast Marginal Likelihood Maximization approach has a complexity of $O(MN + K(2 + K^2))$ (see Appendix A in [7]). This method is indeed fast when $K$ is small. However, when $K$ is large, it can be rather slow.

Herein, we propose a novel Gibbs sampling based SBL approach which computes the \emph{a posteriori} pdf of $x$ in $O(N)$ steps. We refer to this approach as GS-SBL. It is known that, in general, the Markov Chain Monte Carlo (MCMC) methods are slower than other Bayesian methods. However, with an efficient sampling step, GS-SBL only has a complexity of $O(N)$. Although we still need to generate a few hundred samples to obtain a good estimate for $x$, in large-scale compressed sensing problems the total time needed by GS-SBL is still much less than that required by the original SBL proposed in [6]. Furthermore, numerical examples show that GS-SBL can outperform the existing SBL approaches in both small- and large-scale compressed sensing problems.

2. AN EFFICIENT GIBBS SAMPLING APPROACH

Consider the following Bayesian model:

$$
\begin{align*}
    y|x, \eta & \sim \mathcal{N}(Ax, \eta I) \\
    x|p & \sim \mathcal{N}(0, P), \quad f(\eta) \propto 1 \\
    p_n|a, b & \sim \mathcal{IG}(a, b),
\end{align*}
$$

(4)

where $p = [p_1, p_2, \ldots, p_N]^T$, $P = \text{diag}(p)$, $\mathcal{N}(\mu, \Sigma)$ represents the multivariate normal distribution with mean $\mu$ and covariance matrix $\Sigma$, and $\mathcal{IG}(a, b)$ represents the inverse gamma distribution. We assume the inverse gamma prior for $p_n$ because we know that $p$ is sparse and the inverse gamma prior promotes sparsity in the estimate of $p$, when $b$ is small. But for $\eta$ we do not have any \emph{a priori} information, so it is proper to assume that $f(\eta) \propto 1$. Based on the Bayesian model (4), we can readily obtain:

$$
\begin{align*}
    f(p, \eta|y, x, a, b) & \propto f(y|x, \eta)f(x|p)f(p|a, b) \\
    & \propto \eta^{-\left(M/2\right)}e^{-\frac{1}{\eta}(y-Ax)^2} \prod_{n=1}^{N} p_n^{-(a+3/2)} e^{-\frac{1}{2b}(a + \frac{x_n^2}{2})},
\end{align*}
$$

(5)

and

$$
\begin{align*}
    f(x|y, p, \eta, a, b) & \propto f(y|x, \eta)f(x|p)f(p|a, b) \\
    & \propto e^{-\frac{1}{2}(x-\mu)\Sigma^{-1}(x-\mu)},
\end{align*}
$$

(6)

where $\Sigma = (\eta^{-1} A' A + P^{-1})^{-1}$, and $\mu = \eta^{-1} \Sigma A' y$.

It follows from (5) that $p$ and $\eta$ are conditionally independent and that they have the following distributions, respectively,

$$
\eta|y, x, a, b \sim \mathcal{IG} \left( \frac{M}{2} - 1, \frac{||y - Ax||^2}{2} \right),
$$

(7)

and

$$
p_n|y, x, a, b \sim \mathcal{IG} \left( a + \frac{1}{2}, b + \frac{x_n^2}{2} \right).
$$

(8)

From (6), $x$ has a conditional normal distribution:

$$
x|y, p, \eta, a, b \sim \mathcal{N}(\mu, \Sigma).
$$

(9)

Using (7), (8) and (9), we can derive the Gibbs sampling algorithm (see, e.g., [9]). Assume that an initial estimate $x^{(0)}$ is given (see the comments on the initial estimator at the end of this section). In the $t$th iteration, we perform the following two steps:

1. We first draw a sample of $p_n^{(t)}$ from the conditional pdf $f(p_n|y, x^{(t-1)}, a, b)$ for $n = 1, 2, \ldots, N$, and draw a sample of $\eta^{(t)}$ from the conditional pdf $f(\eta|y, x^{(t-1)}, a, b)$. By using Equation (8), we obtain

$$
p_n^{(t)} \sim \mathcal{IG} \left( a + \frac{1}{2}, b + \frac{(x_n^{(t-1)})^2}{2} \right), n = 1, 2, \ldots, N,
$$

(10)

and by using Equation (7), we obtain

$$
\eta^{(t)} \sim \mathcal{IG} \left( \frac{M}{2} - 1, \frac{||y - Ax^{(t-1)}||^2}{2} \right).
$$

(11)

2. Then we draw a sample of $x^{(t)}$ from the conditional pdf of $f(x|y, p^{(t)}, \eta^{(t)}, a, b)$. Define the matrix $P^{(t)} = \text{diag}(p^{(t)})$. By Equation (9), we obtain

$$
x^{(t)} \sim \mathcal{N}(\mu^{(t)}, \Sigma^{(t)}),
$$

(12)

where

$$
\Sigma^{(t)} = \left( \eta^{(t)} \right)^{-1} A' A + \left( P^{(t)} \right)^{-1}.
$$

(13)

and

$$
\mu^{(t)} = P^{(t)} A' \left( \eta^{(t)} I + AP^{(t)} A' \right)^{-1} y.
$$

(14)

Note that the matrix inversion lemma was used to obtain (14) from (13).

When $t$ is sufficiently large, $x^{(t)}$ can be viewed as a sample generated from the \emph{a posteriori} pdf $f(x|y, a, b)$. In the $t$th iteration of the GS-SBL approach, $\{p_n^{(t)}\}_{n=1}^{N}$ and $\eta^{(t)}$ can be generated by using the accept/reject algorithm (see, e.g., [10]) whose computational complexity is $O(N)$. The normal random vector $x^{(t)}$ can be generated by means of an affine transformation of a standard normal random vector $z$: $x^{(t)} = \mu^{(t)} + \Phi^{(t)} z$, where $z \sim \mathcal{N}(0, I)$ and $\Phi^{(t)}(\Phi^{(t)})' = \Sigma^{(t)}$. Conventionally, $\Phi^{(t)}$ is obtained by the Cholesky decomposition. However, the Cholesky decomposition has...
a complexity of $O(N^3)$, and therefore it is not suitable for large-scale compressed sensing applications. Instead, we present an efficient way to generate $x^{(t)}$ as follows.

Let $z_1$ and $z_2$ be two independent normal random vectors of dimensions $M$ and $N$, respectively. Then,

$$z_3 = \left( \eta^{(t)} \right)^{-1/2} A' z_1 + \left( P^{(t)} \right)^{-1/2} z_2$$

(16)

is a normal random vector with mean 0 and covariance matrix $(\Sigma^{(t)})^{-1}$ (see (13)). The complexity of (16) is determined by the complexity of the matrix vector product $A' z_1$. In other words, Equation (16) has $O(N^3)$ complexity.

Next, we can verify that

$$x^{(t)} = \mu^{(t)} + \Sigma^{(t)} z_3,$$

(17)

is a normal random vector with mean $\mu^{(t)}$ and covariance matrix $\Sigma^{(t)}$. Using (14) and (15), we can simplify (17) as follows:

$$x^{(t)} = P^{(t)} z_1 - P^{(t)} A' \left( \eta^{(t)} I + A P^{(t)} A' \right)^{-1} v,$$

(18)

where $v = y - A P^{(t)} z_2$.

We can use a conjugate gradient (CG) method to compute

$$\left( \eta^{(t)} I + A P^{(t)} A' \right)^{-1} v.$$  

CG only involves the product of the matrix $\eta^{(t)} I + A P^{(t)} A'$ and a vector. Let the vector be $d$. Then,

$$(\eta^{(t)} I + A P^{(t)} A') d = \eta^{(t)} d + A \left[ P^{(t)} \odot (A' d) \right],$$

(19)

where $\odot$ is the Hadamard matrix product. Hence, if the matrix vector product has $\mathcal{O}(N^3)$ complexity, then the computational complexity of CG for a fixed number of iterations is $\mathcal{O}(N^3)$. As a result, each iteration of GS-SBL requires a complexity of $\mathcal{O}(N^3)$ flops.

Let $T_b$ be the time to begin sampling (also called time of burn-in) and $T_s$ be the time of stop sampling. We can estimate $x_n$ by the sample median:

$$\hat{x}_n = \text{the median of } \{x_n(T_b), x_n(T_b+1), \ldots, x_n(T_s)\}.$$  

(20)

When there are sufficiently many samples (i.e., $T_s \to \infty$), the sample median $\hat{x}_n$ approaches the a posteriori median (i.e., the u defined by $
abla x_n(y)dx_n = \int_{-\infty}^{\infty} f(x_n(y))dx_n = 0.5$, which is the solution to

$$\min_{x_n} E_{x_n|y}||x_n - \hat{x}_n||.$$  

(21)

We conclude this section by making the following comments:

1. We recommend using CoSaMP [5] to obtain the initial estimate of $x$, because CoSaMP has $\mathcal{O}(N)$ complexity and it converges faster than GS-SBL. CoSaMP requires the knowledge of the sparsity level, i.e., the number of non-zero elements in $x$ (i.e., $K$), which is unknown a priori in most applications. Since $K$ is unknown, we simply set it to $M/2$.

2. Since each iteration of GS-SBL has $\mathcal{O}(N^3)$ complexity and the number of iterations is fixed to $T_s$ (we set $T_s$ to 500 in the numerical examples), the total computational complexity of GS-SBL is $\mathcal{O}(N^3)$.

3. GS-SBL needs $N(T_s - T_b)$ memory to compute (20). To reduce the memory requirement, we can simply store the recently-generated $T$ samples (we set $T$ to 200 in the numerical examples) and we compute the sample mean of these $T$ samples every $T_i$ iterations (we set $T_i$ to 40 in the numerical examples). Then, after $T_i$ iterations, we select the sample mean which minimizes $||y - Ax||_2$ as the estimate for $x$.

### 3. Numerical Examples

In this section, we first compare the performance of GS-SBL with those of the direct differentiation based SBL (which we simply refer to as SBL), the EM-based SBL (which we refer to as EM-SBL), the Fast Marginal Likelihood Maximization (which we refer to as BCS according to [8]) by using a small- and a large-scale compressed sensing problem.

#### 3.1. Small-Scale Problem

First, let us consider a spectral estimation example in time series analysis, which has many applications in diverse fields. The measurement vector $y$ and the spectral component $x$ satisfy the following noisy DCT equations:

$$y_n \triangleq y(t_m) = w(t_m) \sum_{n=1}^{N} x_n \cos(\frac{2\pi n - 1}{2N}(t_m - 1)) + e(t_m),$$

where $w(t_m) = \begin{cases} 1/\sqrt{N}, & t_m = 1; \\ 2/\sqrt{N}, & t_m > 1 \end{cases}$ is a random subset of the set $\{1, 2, \ldots, N\}$ and the noise term $e(t_m)$ has a normal distribution with zero mean and variance $\eta$. In this case, the measurement matrix $M$ is a random sampling matrix and the basis matrix $B$ is a DCT matrix. Similar to the example in [8], we set $K = 20, M = 128$ and $N = 512$. The 20 non-zero elements in $x$ are either 0 or $-1$ and their locations are chosen randomly.

Since GS-SBL is initialized by CoSaMP, we also initialize all the algorithms in the numerical examples by CoSaMP. We run SBL and EM-SBL until they satisfy the convergence criterion $||x^{(t+1)} - x^{(t)}||_2/||x^{(t)}||_2 < 10^{-3}$, where $x^{(t)}$ is the estimate in the $t$th iteration. The code for BCS is downloaded from the website “http://people.ee.duke.edu/ lihan/cs/”. BCS also assumes the true noise power $\eta$, since otherwise its performance can be rather poor. Note that SBL, EM-SBL and GS-SBL do not assume such a knowledge. The default stopping criteria is used for BCS. We set $a = 0.5, b = 10^{-4}, T_b = 101$ and $T_s = 500$ for GS-SBL.

From Figure 1 we can see that GS-SBL gives the most accurate estimate of $x$. Figure 2(a) shows the average reconstruction errors (from 50 Monte-Carlo trials) as functions of the signal-to-noise ratio (SNR) for $K = 20, M = 128$ and $N = 512$. Here, since the signal is either 1 or $-1$, the SNR is simply defined as the inverse noise power. We can see from Figure 2(a) that GS-SBL has better performance than the existing SBL approaches. For SNR $\geq 25$ dB, GS-SBL has 0 dB better than BCS, 1$-8$ dB better than SBL and 6$-26$ dB better than EM-SBL. Figure 2(b) shows the average reconstruction errors (based on 50 Monte-Carlo trials) as functions of $M$ for SNR $\geq 30$ dB, $K = 20$, and $N = 512$. Again, GS-SBL has the best performance. For $M \geq 128$, GS-SBL is 4$-5$ dB better than BCS, 5$-6$ dB better than SBL and 4$-14$ dB better than EM-SBL.

#### 3.2. Large-Scale Problem

We consider the previous spectral analysis example once again. This time, we set $K = 500, M = 3000$ and $N = 8192$. The 500 non-zero elements in $x$ are either 1 or $-1$ and their locations are chosen randomly. The noise power $\eta$ is set to be $-30$ dB. We compare the performance of GS-SBL with that of BCS in this scenario. We make use of the DCT transform to compute the matrix-vector product in the BCS code and in the GS-SBL code. We use the default stopping criteria for BCS, and we set the maximum number of iterations to
10^5. As before, we set \(a = 0.5, b = 10^{-4}, T_b = 101\) and \(T_s = 500\) for GS-SBL.

We can see from Figure 3 that the result generated by GS-SBL is more accurate than that of BCS, and its reconstruction error is smaller than that of BCS. The average reconstruction error of GS-SBL (from 50 Monte-Carlo Trials) is \(-24.1\) dB and the average reconstruction error of BCS is \(-19.7\) dB. On average, GS-SBL needs 476 seconds and BCS needs 1105 seconds on a server with Intel Xeon 2.33 GHz Quad-Core Processor. Hence, GS-SBL is both faster and more accurate than BCS in this case. For problems of larger sizes, GS-SBL offers further computational savings over BCS.

4. CONCLUSIONS

We have proposed a Gibbs sampling based SBL approach for compressed sensing. We refer to this algorithm as GS-SBL. With an efficient sampling step, GS-SBL can have a lower computational complexity than those of the direct differentiation based SBL, the EM-based SBL and the Fast Marginal Likelihood Maximization algorithms. We have also shown by using numerical examples that GS-SBL outperforms these SBL approaches.

5. REFERENCES