Online Sparsifying Transform Learning for Signal Processing

Saiprasad Ravishankar†, Bihan Wen†, and Yoram Bresler
Department of Electrical and Computer Engineering and the Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Champaign, IL, USA

***Abstract*—Many techniques in signal and image processing exploit the sparsity of natural signals in a transform domain or dictionary. Adaptive synthesis dictionaries have been shown to be useful in applications such as signal denoising, and compressed sensing. More recently, the data-driven adaptation of sparsifying transforms has received some interest. The sparsifying transform model allows for exact and cheap computations. In this work, we propose a framework for online learning of square sparsifying transforms. Such online learning can be particularly useful when dealing with big data, and for signal processing applications such as real-time sparse representation and denoising. The proposed online transform learning algorithm is shown to have a much lower computational cost than online synthesis dictionary learning. The sequential learning of a sparsifying transform also typically converges faster than batch mode transform learning. Preliminary experiments show the usefulness of the proposed schemes for sparse representation, and denoising.**

***Index Terms*—Sparse representations, Sparsifying transforms, Online learning, Big data, Dictionary learning, Denoising**

I. Introduction

The sparsity of natural signals in a certain transform domain or dictionary has been widely exploited in various applications in recent years. Various sparse models have been studied such as the *synthesis model* [1], *analysis model* [1], [2], and *transform model* [3], [4].

The *synthesis model* suggests that a signal $y \in \mathbb{R}^n$ can be sparsely represented in a dictionary $D \in \mathbb{R}^{n \times K}$ as $y = Dx$, where $x \in \mathbb{R}^K$ is sparse, i.e., $\|x\|_0 \ll K$. The l_0 quasi norm counts the number of non-zeros in x. Natural signals usually satisfy $y = Dx + e$, where e is an approximation error in the signal domain [5]. The alternative *analysis model* [1] for a signal y says that Ωy is sparse ($\|\Omega y\|_0 \ll m$), where $\Omega \in \mathbb{R}^{m \times n}$ is an analysis dictionary [6]. A more general *noisy signal analysis model* [6], [7] models the signal y as $y = z + e$, with Ωz sparse, and e a (small) noise term in the signal domain.

In this work, we focus our attention on the classical transform model, which suggests that a signal y is *approximately sparsifiable* using a transform $W \in \mathbb{R}^{m \times n}$, that is $Wy = x + e$ where $x \in \mathbb{R}^m$ is sparse in some sense, and e is a small residual in the transform rather than in the signal domain. Natural signals are known to be approximately sparsifiable by the discrete cosine transform (DCT), or Wavelets [8].

The transform model allows for much faster computations than the synthesis and noisy signal analysis models. Given a signal y and sparsifying transform W, the process of obtaining a sparse code x of sparsity s involves minimizing $\|Wy - x\|_2^2$ subject to $\|x\|_0 \leq s$. The solution $\hat{x}$ here is obtained exactly and cheaply by zeroing out all but the s coefficients of largest magnitude in Wy. In contrast, sparse coding with synthesis or analysis dictionaries involves solving NP-hard (Non-deterministic Polynomial-time hard) problems [6], [9], [10] approximately. Given the transform W and sparse code x, we can also obtain a (simple) least squares estimate of the signal y as $\hat{y} = W^\dagger x$, where $W^\dagger$ is the pseudo-inverse of W [4].

Recently, the adaptation of sparse models to data has received much attention [4], [6], [11]–[17]. Various applications such as denoising, and compressed sensing benefit from an adaptive sparse model. Importantly, the learning of transform models has been shown to be much cheaper than synthesis, or analysis dictionary learning [4], [18]. Adaptive transforms also provide competitive signal reconstruction quality in applications [4], [18]–[20].

Prior work on transform learning focused on batch learning [4], [21], [22], where the sparsifying transform is adapted using all the training data simultaneously. However, for big data, the size of the training data set is typically very large. Hence, batch learning of a sparsifying transform using existing alternating algorithms [4], [21], [22] is computationally expensive in both time and memory, and may be even infeasible. Moreover, in real-time applications, the data arrives sequentially, and must also be processed sequentially to limit latency. Thus, this setting renders batch learning infeasible, since in real-time applications, one does not have access to all the data at once. Hence, we introduce in this work a scheme for online, or sequential learning of a *square* sparsifying transform $W \in \mathbb{R}^{n \times n}$. Our framework iteratively adapts the sparsifying transform and sparse codes (and/or signal estimates) for signals (or, measurements) that arrive, or are processed sequentially. Such online/sequential learning is amenable to big data, and applications such as real-time sparse representation (compression), denoising, and compressed sensing. As we show in this work, online transform learning involves cheap computations and modest memory requirements. Moreover, online transform learning also admits strong convergence guarantees [23].

While the online learning of overcomplete synthesis dictionaries has been studied previously by Mairal et al. [24], the online adaptation of the transform model allows for much cheaper computations. As we show, the sequential transform learning scheme also converges somewhat faster than the batch transform learning scheme [4], [22].

II. Problem Formulations

A. Batch Learning

In batch learning, the sparsifying transform is adapted to all the training data simultaneously. Given a matrix $Y \in \mathbb{R}^{n \times N}$, whose columns y_i ($1 \leq i \leq N$) represent all the training signals, the problem of learning an adaptive square sparsifying transform W (in batch mode) is formulated as follows [4], [21], [22]

$$\text{(P0)} \quad \min_{W,X} \|WY - X\|_F^2 + \lambda\, v(W) \quad s.t. \quad \|x_i\|_0 \leq s \ \forall\, i$$

where x_i denotes the i^{th} column of the sparse code matrix X, s is a given sparsity level, and $v(W) = -\log|\det W| + \|W\|_F^2$. The term $\|WY - X\|_F^2$ in (P0) is the sparsification error for the data Y in the transform W. The sparsification error is the modeling error in the transform model, and hence we minimize it in order to learn the best possible transform model. Problem (P0) also has $v(W)$ as a regularizer in the objective to prevent trivial solutions [4]. The log determinant and Frobenius norm penalty terms in $v(W)$ together fully control the condition number and scaling of the learnt transform [4], [22]. This eliminates badly conditioned transforms, which typically convey little information and may degrade performance in applications. To make the two terms in the cost of (P0) scale similarly, we set $\lambda = \lambda_0 \|Y\|_F^2$ with constant $\lambda_0 > 0$. The

† Equal contributors. This work was supported in part by the National Science Foundation (NSF) under grants CCF-1018660 and CCF-1320953.

constant λ_0 controls the condition number of the learnt W. In the limit $\lambda_0 \to \infty$ (and assuming $Y \neq 0$), the condition number of the optimal transform(s) in (P0) tends to 1. In practice, the transforms learnt via (P0) have condition numbers very close to 1 even for finite λ_0 [22]. The specific choice of λ_0 depends on the application and desired condition number.

B. Online Learning

Now, we introduce our problem formulation for online sparsifying transform learning. For time $t = 1, 2, 3, ...$, the optimization problem to update the sparsifying transform and sparse code based on new data $y_t \in \mathbb{R}^n$ is as follows

$$\text{(P1)} \quad \left\{\hat{W}_t, \hat{x}_t\right\} = \underset{W,\, x_t}{\arg\min} \frac{1}{t} \sum_{j=1}^{t} \left\{\|Wy_j - x_j\|_2^2 + \lambda_j\, v(W)\right\} \quad s.t. \;\; \|x_t\|_0 \leq s$$

where $\lambda_t = \lambda_0 \|y_t\|_2^2$, $\hat{W}_t$ is the optimal transform at time t, and $\hat{x}_t$ is the optimal sparse code for y_t. Note that only the latest sparse code is updated at time t. The condition $x_j = \hat{x}_j,\ 1 \leq j \leq t-1$ is assumed here and in the sequel. Problem (P1) is simply an online version of the batch problem (P0), and hence it shares some similar properties with (P0).

Although Problem (P1) outputs an optimal $\hat{W}_t$ for each t, it is typically impractical to store $\hat{W}_t$ for all t. In our experiments, we store only the latest $\hat{W}_t$, and use it as an initialization for the algorithm that solves for $\hat{W}_{t+1}$. At any instant t, one can obtain an estimate of signals $\{y_j\}_{j=1}^t$ from their sparse codes as $\hat{W}_t^{-1} x_j$ $\forall j$ (i.e., 'decompressing' the signals from the stored sparse codes).

For small values of t, Problem (P1) may highly overfit (this is typically undesirable) the transform to the data. In order to overcome this problem, for small values of t, we only perform an update of the sparse codes (with a fixed W – set to a reasonable initialization).

Problem (P1) can be further modified, or improved in certain scenarios. For example, for dynamically changing data, it may not be possible to fit a single transform W to y_t for all t. In this case, one can introduce a forgetting factor ρ^{t-j} (with a constant $\rho < 1$), that scales the j^{th} term (for each j) in the objective of (P1) [25]. Such a forgetting factor would diminish the influence of "old" data. For fixed size data sets, Problem (P1) can be used as a sequential learning and sparse coding (compression) strategy. In this case, it is typically useful to cycle through the data set a few times (i.e., make multiple passes through the data set), which crucially allows for a better updating of the sparse codes (which is done sequentially). Similar strategies have been used for online synthesis dictionary learning [24].

C. Mini-batch Learning

A useful variation of online learning is mini-batch learning, where we process more than one signal at a time. This scheme may provide potential speedups over online learning. However, the processing of blocks of signals increases memory requirements and latency.

Assuming a fixed block (mini-batch) size of M, the L^{th} ($L \geq 1$) block of signals (in terms of the time sequence $\{y_t\}$) is $Y_L = \left[y_{LM-M+1} \mid y_{LM-M+2} \mid \;\; ... \;\; \mid y_{LM}\right]$. For $L = 1, 2, 3, ...$, the mini-batch sparsifying transform learning problem is

$$\left\{\hat{W}_L, \hat{X}_L\right\} = \underset{W,\, X_L}{\arg\min} \frac{1}{LM} \sum_{j=1}^{L} \left\{\|WY_j - X_j\|_F^2 + \Lambda_j\, v(W)\right\} \quad s.t. \;\; \|x_{LM-M+i}\|_0 \leq s \;\; \forall\, i \in \{1, .., M\} \;\; \text{(P2)}$$

where the weight $\Lambda_j = \lambda_0 \|Y_j\|_F^2$, and the matrix $X_j = \left[x_{LM-M+1} \mid x_{LM-M+2} \mid \;\; ... \;\; \mid x_{LM}\right]$ contains the block of sparse codes corresponding to Y_j.

D. Online Denoising Formulation

Online transform learning can be used for various applications such as sparse representation (compression), denoising, compressed sensing, etc. Here, we consider an extension of (P1)/(P2) (which by themselves, can be used for sparse representation of signals) to denoising. Denoising aims to recover an estimate of the signal $z \in \mathbb{R}^n$ from its measurement $y = z + h$, corrupted by noise h. Here, we consider a time sequence of measurements $\{y_t\}$, with each $y_t = z_t + h_t$, and $h_t \in \mathbb{R}^n$ being the noise. We work with h_t whose entries are i.i.d. Gaussian with zero mean and variance σ^2. The goal of online denoising is to recover estimates of $z_t\ \forall\ t$. We model the underlying noiseless signals z_t as approximately sparse in a (unknown) transform domain. Previous work presented a formulation [4], [18] for adaptive transform-based batch denoising. Here we present a simple denoising formulation that is a modification of the online learning Problem (P1). For $t = 1, 2, 3, ...$, solve

$$\text{(P3)} \quad \min_{W,\, x_t} \frac{1}{t} \sum_{j=1}^{t} \left\{\|Wy_j - x_j\|_2^2 + \lambda_j\, v(W) + \tau_j^2 \|x_j\|_0\right\}$$

where $\tau_j \propto \sigma$. Problem (P3) estimates $\hat{W}_t$ and $\hat{x}_t$. The denoised signal is then computed simply as $\hat{z}_t = \hat{W}_t^{-1} \hat{x}_t$. We can also extend (P3) to its mini-batch version. Problem (P3) can also be used for patch-based denoising of images [13], [18]. The overlapping patches are processed sequentially, and the denoised image is obtained by averaging the denoised patches at their respective image locations.

III. Algorithms and Computational Properties

A. Batch Learning

Previous work [21], [22] proposed an alternating algorithm for solving Problem (P0) that alternates between solving for X (*sparse coding step*) and W (*transform update step*), with the other variable kept fixed. The sparse coding step is as follows

$$\min_X \|WY - X\|_F^2 \;\; s.t. \;\; \|x_i\|_0 \leq s \;\; \forall\, i \tag{1}$$

An optimal solution to (1) is computed exactly as $\hat{x}_i = H_s(Wy_i)$ $\forall\, i$, where the operator $H_s(\cdot)$ zeros out all but the s coefficients of largest magnitude in a vector. If there is more than one choice for the s coefficients of largest magnitude in a vector z, then we choose $H_s(z)$ as the vector for which the indices of the s largest magnitude elements in z are the lowest possible. The transform update step of (P0) involves the following unconstrained non-convex minimization.

$$\min_W \|WY - X\|_F^2 + \lambda\, v(W) \tag{2}$$

The solution to (2) is also computed in closed-form. Let us factorize $YY^T + \lambda I$ (where I is the identity matrix) as LL^T, with $L \in \mathbb{R}^{n \times n}$. Further, let $L^{-1} Y X^T$ have a full singular value decomposition (SVD) of $Q \Sigma R^T$, where Q, Σ, and R are $n \times n$ matrices. Then, a closed-form solution or global minimizer of (2) is

$$\hat{W} = 0.5 R \left(\Sigma + \left(\Sigma^2 + 2\lambda I\right)^{\frac{1}{2}}\right) Q^T L^{-1} \tag{3}$$

where $(\cdot)^{\frac{1}{2}}$ denotes the positive definite square root. The solution above is unique if and only if $L^{-1} Y X^T$ is non-singular [22]. Furthermore, the solution is invariant to the choice of factor L.

The total cost per iteration (of sparse coding and transform update) of this batch transform learning algorithm scales as $O(Nn^2)$. This is much lower than the per-iteration cost of learning an $n \times K$ overcomplete ($K > n$) synthesis dictionary D using K-SVD [12], which scales (assuming that the synthesis sparsity level $s \propto n$) as $O(KNn^2)$. Previous work [4], [22] has demonstrated that batch

transform learning also converges quickly (in a small number of iterations) in practice. The (local) memory requirement of batch transform, or dictionary learning scales as $O(Nn)$.

B. Online Learning

Here, we solve Problem (P1) at each time instant t by alternating minimization (similar to (P0)). In the sparse coding step, we solve (P1) for x_t with fixed $W = \hat{W}_{t-1}$ (warm start) as follows

$$\min_{x_t} \|Wy_t - x_t\|_2^2 \;\; s.t. \;\; \|x_t\|_0 \leq s \tag{4}$$

The sparse coding solution is given as $\hat{x}_t = H_s(Wy_t)$. In the transform update step, we solve (P1) with fixed x_t as

$$\min_{W} \frac{1}{t} \sum_{j=1}^{t} \left\{ \|Wy_j - x_j\|_2^2 + \lambda_j \, v(W) \right\} \tag{5}$$

This problem has a closed-form solution (cf. [25] for the explanation of this *exact* transform update solution). However, the computation of the closed-form solution requires computing a full SVD, a matrix inverse, and matrix-matrix multiplications, which would scale computationally as $O(n^3)$. Instead, we propose a computationally efficient *approximate* transform update algorithm as follows.

First, let us factorize $t^{-1} \sum_{j=1}^{t} \left(y_j y_j^T + \lambda_j I \right)$ as $L_t L_t^T$ (i.e., take square root), with $L_t \in \mathbb{R}^{n \times n}$. Here, we work with the eigenvalue decomposition (EVD) square root (positive-definite square root). Denoting the full EVD of $\Gamma_{t-1} \triangleq (t-1)^{-1} \sum_{j=1}^{t-1} y_j y_j^T$ as $U_{t-1} \Delta_{t-1} U_{t-1}^T$, the full EVD of $\Gamma_t = (1 - t^{-1}) \Gamma_{t-1} + t^{-1} y_t y_t^T$ can be found via a rank-1 update to the (scaled) EVD of Γ_{t-1} [26]. Furthermore, let $\beta_{t-1} = \sum_{j=1}^{t-1} (t-1)^{-1} \lambda_j$. Then, $\beta_t = (1 - t^{-1}) \beta_{t-1} + t^{-1} \lambda_t$. The EVD square root of $t^{-1} \sum_{j=1}^{t} \left(y_j y_j^T + \lambda_j I \right)$ is then $L_t = U_t \left(\Delta_t + \beta_t I \right)^{\frac{1}{2}} U_t^T$.

Subsequently, $L_t^{-1} = U_t \left(\Delta_t + \beta_t I \right)^{-\frac{1}{2}} U_t^T$. The matrix-matrix products in the formula for L_t^{-1} are not explicitly computed. Instead, we will only need the application of L_t^{-1} to a vector, which can be performed efficiently with $O(n^2)$ computation by first applying U_t^T, then the diagonal matrix $\left(\Delta_t + \beta_t I \right)^{-\frac{1}{2}}$, and lastly applying U_t.

In order to compute the closed-form solution to (5), we need to compute the full SVD of $t^{-1} \sum_{j=1}^{t} L_t^{-1} y_j x_j^T$. In order to simplify this computation, we perform the following approximation, where $\Theta_t = t^{-1} \sum_{j=1}^{t} y_j x_j^T$

$$\begin{aligned} L_t^{-1} \Theta_t &= L_t^{-1} \left\{ (1 - t^{-1}) \Theta_{t-1} + t^{-1} y_t x_t^T \right\} \\ &\approx (1 - t^{-1}) L_{t-1}^{-1} \Theta_{t-1} + t^{-1} L_t^{-1} y_t x_t^T \end{aligned} \tag{6}$$

With the above approximation, and the fact that $t^{-1} L_t^{-1} y_t x_t^T$ is a rank-1 matrix, the estimate of the full SVD of $t^{-1} \sum_{j=1}^{t} L_t^{-1} y_j x_j^T = L_t^{-1} \Theta_t$ can be obtained by performing a rank-1 update to the (scaled) SVD estimate of $L_{t-1}^{-1} \Theta_{t-1}$. The net error in the above approximation at time t is given as $E_t = \sum_{j=2}^{t} \frac{j-1}{t} \left(L_j^{-1} - L_{j-1}^{-1} \right) \Theta_{j-1}$ [25]. We have shown elsewhere that the net error E_t is bounded by some constant C for all t [23]. To prevent any undesirable error accumulations (over time), we monitor the relative error $\|E_t\|_F \, / \, \left\| L_t^{-1} \Theta_t \right\|_F$. The relative error can be (worst case) upper bounded (upto a scale factor) by $\sum_{j=2}^{t} \left(\left\| L_j^{-1} - L_{j-1}^{-1} \right\|_F \, / \, \left\| L_t^{-1} \right\|_F \right)$. The latter quantity is cheap to compute, and can be monitored. If it rises above a threshold ϵ, we compute the SVD of $L_t^{-1} \Theta_t$ directly, in which case any possible accumulated error is wiped out. In our experiments, we observed that L_t^{-1} converges quickly [1] (over t) for data consisting of natural signals. In such cases, the exact SVD can be performed for a few initial time instances, after which the approximation (6) is observed to work very well.

Now, once the full SVD estimate of $L_t^{-1} \Theta_t$ is computed as $Q_t \Sigma_t R_t^T$, the closed-form solution (3) for Problem (5) is simply

$$\hat{W}_t = 0.5 R_t \left(\Sigma_t + \left(\Sigma_t^2 + 2 \beta_t I \right)^{\frac{1}{2}} \right) Q_t^T L_t^{-1} \tag{7}$$

Again, we do not perform any of the matrix-matrix multiplications in (7). Instead, we store the individual matrices, and apply them one by one on vectors (during sparse coding), at a computational cost of $O(n^2)$.

An alternative way to perform transform update would be to use the stochastic gradient descent method. However, the gradient computation requires computing the inverse of a matrix (a term like W^{-T} [4]). This computation scales worse ($O(n^3)$) than the computation for the proposed method.

Our algorithm can be easily modified to accomodate the various modifications to Problem (P1) suggested in Section II-B for dynamically changing data, and for fixed data sets [25].

While one could alternate multiple times (for each t) between the sparse coding and transform update steps of (P1), we perform only a single alternation to save computations. The computational cost of the sparse coding step is dominated by the computation of the product Wy_t [4], and therefore scales as $O(n^2)$. The computational cost of the transform update step scales as $O(n^2 \log^2 n)$, and is dominated by the computations for the rank-1 SVD or EVD updates [26]. Thus, the total cost per signal (or, per time instant) of our algorithm (sparse coding and transform update) scales as $O(n^2 \log^2 n)$. This is better (especially for large signals) than the computational cost per signal for online learning of an $n \times K$ overcomplete synthesis dictionary D, which scales (assuming synthesis sparsity $s \propto n$, and $K \propto n$) at least as $O(n^3)$ [24]. The (local) memory requirement of our algorithm scales modestly as $O(n^2)$, since we need to store $n \times n$ matrices.

C. Mini-batch Learning

Here, we solve Problem (P2) by alternating minimization. In the sparse coding step, we solve for X_L in (P2), with fixed $W = \hat{W}_{L-1}$ as follows

$$\min_{X_L} \|WY_L - X_L\|_F^2 \;\; s.t. \;\; \|x_{LM-M+i}\|_0 \leq s \; \forall \, i \tag{8}$$

The optimal solution to (8) is obtained similarly as in (1), i.e., $\hat{x}_{LM-M+i} = H_s(Wy_{LM-M+i}) \; \forall \; i \in \{1, .., M\}$.

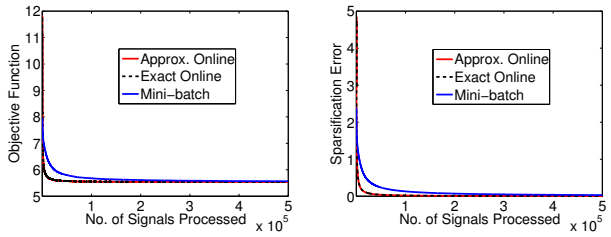
The transform update problem solves

$$\min_{W} \; \frac{1}{LM} \sum_{j=1}^{L} \left\{ \|WY_j - X_j\|_F^2 + \Lambda_j \, v(W) \right\} \tag{9}$$

When M is small ($M \ll n$), we can use the same transform update procedure as in Section III-B, but with the rank-1 updates replaced by rank-M updates [25]. The rank-M updates can be performed as M rank-1 updates for small M. For larger M ($M \sim O(n)$, or larger), (9) is solved without any approximations (cf. Fig. 2 in [25]).

The computational cost per block of the sparse coding step scales as $O(Mn^2)$. For small M, the cost of the transform update step scales as $O(Mn^2 \log^2 n)$. For large M, the cost of transform update per block scales as $C_1 Mn^2 + C_2 n^3$, where C_1 and C_2 are constants [25]. Assuming that $C_2 n \ll C_1 M$ (large M), the cost scales as $O(Mn^2)$. Thus, the total computation per block of our mini-batch algorithm (sparse coding and transform update) scales as $O(Mn^2)$ for large M, and $O(Mn^2 \log^2 n)$ for small M. In either case, the cost is better than the cost per block (of size M) for mini-batch

[1] For example, when y_t are independent and identically distributed, $t^{-1} \sum_{j=1}^{t} y_j y_j^T$ converges (as $t \to \infty$) with probability 1 to a covariance matrix, and L_t^{-1} would also converge.



(a) (b)

Fig. 1. Objective (left) and Sparsification error (right) for the online and mini-batch algorithms as a function of amount of data processed.

learning of an $n \times K$ synthesis dictionary D, which scales (assuming synthesis sparsity $s \propto n$, and $K \propto n$) as $O(Mn^3)$ [24]. The memory requirement of mini-batch learning scales as $O(Mn)$ for large M.

D. Denoising

Here, we solve (P3) at each t by alternating minimization. The sparse coding step (with fixed $W = \hat{W}_{t-1}$) solves

$$\min_{x_t} \ \|Wy_t - x_t\|_2^2 + \tau_t^2 \|x_t\|_0 \tag{10}$$

A solution $\hat{x}_t$ of (10) is $\hat{x}_t = \hat{H}_{\tau_t}(Wy_t)$, where the hard thresholding operator $\hat{H}_\tau(\cdot)$ is defined as

$$\left(\hat{H}_\tau(b)\right)_k = \begin{cases} 0 & , \ |b_k| < \tau \\ b_k & , \ |b_k| \geq \tau \end{cases} \tag{11}$$

where $b \in \mathbb{R}^n$, and the subscript k indexes vector entries. The transform update step of (P3) is identical to (P1). The denoised signal is computed as $\hat{W}_t^{-1}\hat{x}_t$. By, (7), we have

$$\hat{W}_t^{-1} = \beta_t^{-1} L_t Q_t \left(\left(\Sigma_t^2 + 2\beta_t I\right)^{\frac{1}{2}} - \Sigma_t\right) R_t^T \tag{12}$$

Thus, $\hat{W}_t^{-1}\hat{x}_t$ can be computed using matrix-vector multiplications at a cost of $O(n^2)$. The net computational cost of adaptive denoising per signal then scales as $O(n^2 \log^2 n)$ (as in (P1)). For mini-batch based denoising, the computational cost per block is as mentioned in Section III-C.

IV. Numerical Experiments

First, we illustrate the convergence behavior of our online and mini-batch learning algorithms. We generate the input data sequence $\{y_t\}$ as $\left\{W^{-1}x_t\right\}$, using a randomly generated unitary 20×20 matrix W, and randomly generated x_t's that have sparsity level $s = 3$. The parameters are set to $\lambda_0 = 3.1 \times 10^{-2}$, $s = 3$, and $M = 320$.

Fig. 1 plots the objective and sparsification error for our online (using both the exact and approximate alternating algorithms [25]) and mini-batch algorithms. Both the objective and sparsification error converge quickly for our schemes. Importantly, the sparsification error converges to zero, and the condition number to 1, which indicates that a good transform has been learnt. The proposed approximate online learning scheme behaves identically to the one involving exact transform updates. As a function of the number of signals processed, the online scheme converges slightly faster than the mini-batch scheme, because the transform update step is performed more frequently in the former case. However, the approximate online scheme typically has a higher run time (due to the extra $\log^2 n$ factor in computations – see Section III-B) than the mini-batch scheme.

Now, we evaluate the quality of the transforms learnt using our algorithms for representing the Barbara (512×512) and Cameraman (256×256) images. We learn on the 8×8 non-overlapping mean-subtracted patches of the images. We use the recovery PSNR (rPSNR) metric (see [4]). The parameters are set to $\lambda_0 = 3.1 \times 10^{-3}$, $s = 11$, and $M = 256$.

Image	DCT	Batch	Mini-batch		
			$P = 2$	$P = 20$	$P = 70$
Barbara	32.85	34.56	33.66	34.43	34.60
		3.52	0.12	0.66	2.23
Cameraman	29.91	31.94	30.93	31.85	31.93
		0.82	0.05	0.16	0.55

TABLE I

Comparison of the DCT, batch learning, and mini-batch learning with 3 different values for the no. of passes (denoted as P) through the data. The first row for each image indicates the rPSNR (dB), and the second row provides the run time of learning (seconds).

Image	σ	Batch K-SVD	Batch TL	Mini-batch TL	
				without ρ	with ρ
Couple	5	37.28	37.33	37.29	37.33
	10	33.51	33.62	33.53	33.62
	20	30.02	30.02	29.95	30.03
Man	5	36.47	36.66	36.64	36.75
	10	32.71	32.96	32.89	33.00
	20	29.40	29.57	29.46	29.52

TABLE II

PSNR values for denoising with batch K-SVD [13], batch transform learning (TL) [21], and mini-batch TL. The settings for the forgetting factor ρ are mentioned in [25].

Table I shows the recovery PSNRs obtained using the mini-batch (the online scheme works similarly, but is slower) as well as batch schemes, along with the values for the patch-based 2D DCT. The run times of learning are also shown. The mini-batch scheme performs (in rPSNR) better than the 2D DCT even with 2 passes. When the number of passes through the data is increased, the mini-batch performance becomes comparable to, or better than batch learning. Importantly, the mini-batch scheme is faster (in learning) than batch learning. Thus, our mini-batch transform learning scheme can be used as an alternative to batch learning.

We now present results for the simple image denoising framework proposed in Section II-D. We work with the images Couple (512×512) and Man (768×768) [18] and simulate i.i.d. Gaussian noise at 3 different noise levels for the images. We apply mini-batch denoising (both with and without a forgetting factor ρ) on the 8×8 overlapping noisy image patches, that are processed sequentially (with only 1 pass). We set $\lambda_0 = 3.1 \times 10^{-2}$, $\tau_j = 1.73\sigma$, and $M = 64$. Our results are compared to batch K-SVD denoising [12], [13], [27], and to batch transform denoising [21] (parameters set as in [18]).

Table II lists the denoising PSNRs for the various methods. The mini-batch transform denoising method provides comparable, or better performance compared to the batch-based methods. Importantly, the presence of a forgetting factor leads to improved denoising. The mini-batch scheme (with forgetting factor) also provides an average run-time speedup of $26.0\times$ and $3.4\times$ respectively, over the batch K-SVD, and batch transform denoising schemes.

We expect to see greater speedups for our online schemes with efficient implementation, and for larger n, e.g., for 3D or higher-dimensional data. We have presented more extensive results, including for large-scale image denoising elsewhere [25].

V. Conclusions

In this paper, we presented novel formulations for online, and mini-batch sparsifying transform learning. The proposed algorithms were shown to be computationally much cheaper than online synthesis dictionary learning. In practice, our sequential scheme provides speedups over batch transform learning, while achieving the same or better transform quality. We presented experiments demonstrating the usefulness of our schemes in sparse representation, and denoising.

REFERENCES

[1] M. Elad, P. Milanfar, and R. Rubinstein, "Analysis versus synthesis in signal priors," *Inverse Problems*, vol. 23, no. 3, pp. 947–968, 2007.

[2] S. Nam, M. E. Davies, M. Elad, and R. Gribonval, "Cosparse analysis modeling - uniqueness and algorithms," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2011, pp. 5804–5807.

[3] W. K. Pratt, J. Kane, and H. C. Andrews, "Hadamard transform image coding," *Proc. IEEE*, vol. 57, no. 1, pp. 58–68, 1969.

[4] S. Ravishankar and Y. Bresler, "Learning sparsifying transforms," *IEEE Trans. Signal Process.*, vol. 61, no. 5, pp. 1072–1086, 2013.

[5] A. M. Bruckstein, D. L. Donoho, and M. Elad, "From sparse solutions of systems of equations to sparse modeling of signals and images," *SIAM Review*, vol. 51, no. 1, pp. 34–81, 2009.

[6] R. Rubinstein, T. Faktor, and M. Elad, "K-SVD dictionary-learning for the analysis sparse model," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 5405–5408.

[7] R. Rubinstein and M. Elad, "K-SVD dictionary-learning for analysis sparse models," in *Proc. SPARS11*, June 2011, p. 73.

[8] S. Mallat, *A Wavelet Tour of Signal Processing*. Academic Press, 1999.

[9] B. K. Natarajan, "Sparse approximate solutions to linear systems," *SIAM J. Comput.*, vol. 24, no. 2, pp. 227–234, Apr. 1995.

[10] G. Davis, S. Mallat, and M. Avellaneda, "Adaptive greedy approximations," *Journal of Constructive Approximation*, vol. 13, no. 1, pp. 57–98, 1997.

[11] B. A. Olshausen and D. J. Field, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature*, vol. 381, no. 6583, pp. 607–609, 1996.

[12] M. Aharon, M. Elad, and A. Bruckstein, "K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation," *IEEE Trans. Signal Process.*, vol. 54, no. 11, pp. 4311–4322, 2006.

[13] M. Elad and M. Aharon, "Image denoising via sparse and redundant representations over learned dictionaries," *IEEE Trans. Image Process.*, vol. 15, no. 12, pp. 3736–3745, 2006.

[14] K. Skretting and K. Engan, "Recursive least squares dictionary learning algorithm," *IEEE Trans. Signal Process.*, vol. 58, no. 4, pp. 2121–2130, 2010.

[15] M. Yaghoobi, S. Nam, R. Gribonval, and M. Davies, "Analysis operator learning for overcomplete cosparse representations," in *European Signal Processing Conference (EUSIPCO)*, 2011, pp. 1470–1474.

[16] B. Ophir, M. Elad, N. Bertin, and M. Plumbley, "Sequential minimal eigenvalues - an approach to analysis dictionary learning," in *Proc. European Signal Processing Conference (EUSIPCO)*, 2011, pp. 1465–1469.

[17] M. Yaghoobi, S. Nam, R. Gribonval, and M. E. Davies, "Constrained overcomplete analysis operator learning for cosparse signal modelling," *IEEE Trans. Signal Process.*, vol. 61, no. 9, pp. 2341–2355, 2013.

[18] S. Ravishankar and Y. Bresler, "Learning doubly sparse transforms for images," *IEEE Trans. Image Process.*, vol. 22, no. 12, pp. 4598–4612, 2013.

[19] ——, "Sparsifying transform learning for compressed sensing MRI," in *Proc. IEEE Int. Symp. Biomed. Imag.*, 2013, pp. 17–20.

[20] L. Pfister, "Tomographic reconstruction with adaptive sparsifying transforms," Master's thesis, University of Illinois at Urbana-Champaign, Aug. 2013.

[21] S. Ravishankar and Y. Bresler, "Closed-form solutions within sparsifying transform learning," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013, pp. 5378–5382.

[22] ——, "ℓ_0 sparsifying transform learning with efficient optimal updates and convergence guarantees," *IEEE Trans. Signal Process.*, 2014, submitted. Online: https://uofi.box.com/s/oq8xokc7ozgv3yl47ytc.

[23] ——, "Online sparsifying transform learning - part II: Convergence analysis," *IEEE Journal of Selected Topics in Signal Process.*, 2014, accepted for publication. Online: https://uofi.box.com/s/e691o7vjiuv6rfwvwo6d.

[24] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online learning for matrix factorization and sparse coding," *J. Mach. Learn. Res.*, vol. 11, pp. 19–60, 2010.

[25] S. Ravishankar, B. Wen, and Y. Bresler, "Online sparsifying transform learning - part I: Algorithms," *IEEE Journal of Selected Topics in Signal Process.*, 2014, accepted for publication. Online: https://uofi.app.box.com/s/dry7pjyjo09p2pcv1ooi.

[26] P. Stange, "On the efficient update of the singular value decomposition," *PAMM*, vol. 8, no. 1, pp. 10 827–10 828, 2008.

[27] M. Elad, "Michael Elad personal page," http://www.cs.technion.ac.il/~elad/Various/KSVD_Matlab_ToolBox.zip, 2009.