CHASING BUTTERFLIES: IN SEARCH OF EFFICIENT DICTIONARIES

Luc Le Magoarou, Rémi Gribonval
Inria
Centre Inria Rennes - Bretagne Atlantique

ABSTRACT

Dictionary learning aims at finding a frame (called dictionary) in which some training data admits a sparse representation. Traditional dictionary learning is limited to relatively small-scale problems, because high-dimensional dense dictionaries can be costly to manipulate, both at the learning stage and when used for tasks such as sparse coding. In this paper, inspired by usual fast transforms, we consider a *multi-layer sparse* dictionary structure allowing cheaper manipulation, and propose a learning algorithm imposing this structure. The approach is demonstrated experimentally with a factorization of the Hadamard matrix and on image denoising.

***Index Terms*—** Sparse representations, dictionary learning, low complexity, image denoising

1. INTRODUCTION

Sparse representations using dictionaries are a common way of providing concise descriptions of high-dimensional vectors. With an appropriate dictionary $\mathbf{D}$, one can find a sparse representation vector $\boldsymbol{\gamma}$ such that a vector $\mathbf{x}$ of interest can be approximated as $\mathbf{x} \approx \mathbf{D}\boldsymbol{\gamma}$.

Traditionally, one can distinguish two ways of choosing a good dictionary for the data at hand. The dictionary can be either an *analytic dictionary* derived from a mathematical formula, such as the Fourier and Hadamard transforms or wavelets, or a *learned dictionary*, that is automatically inferred from training data. Analytic dictionaries are in general computationally efficient because of associated fast algorithms (e.g., the Fast Fourier Transform (FFT) [1] or the Discrete Wavelet Transform (DWT) [2]), but in turn they lack adaptation to the data. On the other hand, learned dictionaries are in general better adapted to the data, but being dense matrices they are costly to manipulate. A survey on the topic can be found in [3].

In this paper the goal is to design dictionaries that are as well adapted to the data as learned dictionaries, while as fast to manipulate and as cheap to store as analytic ones. Generalizing approaches introduced recently in [4] and [5], we propose here a strategy to tackle this problem, through the notion of Flexible Approximate MUlti-layer Sparse Transform (FAμST). We build on the simple observation that the fast transforms associated with analytic dictionaries can be seen as sequences of cheap linear transformations, indicating that such dictionaries can be expressed as[1][6, Appendix A]:

$$\mathbf{D} = \prod_{j=1}^{M} \mathbf{S}_j, \tag{1}$$

where each factor $\mathbf{S}_j$ is sparse. For example, each step of the butterfly radix-2 FFT can be seen as a product with a sparse matrix having only two non-zero entries per row and column, which leads to the well-known complexity savings. The proposed approach consists in imposing this *multi-layer sparse* dictionary structure. With sparse enough factors, this potentially brings efficiency in terms of:

- *Storage*: The dictionary can be stored efficiently.
- *Speed*: The dictionary and its transpose can quickly multiply vectors and matrices.
- *Statistical relevance*: Learning the dictionary requires fewer training examples because there are fewer parameters to estimate.

Our goal is thus to "chase butterflies", i.e., to learn such dictionaries that somehow extend the butterfly structure of the FFT.

2. PROBLEM FORMULATION AND RELATED WORK

Notation. Throughout this paper, matrices are denoted by bold upper-case letters: $\mathbf{A}$; vectors by bold lower-case letters: $\mathbf{a}$; the ith column of a matrix $\mathbf{A}$ by: $\mathbf{a}_i$; and sets by calligraphical symbols: $\mathcal{A}$. The standard vectorization operator is denoted by $\text{vec}(\cdot)$. The ℓ_0-norm is denoted by $\|\cdot\|_0$ (it counts the number of non-zero entries), $\|\cdot\|_F$ denotes the Frobenius norm, and $\|\cdot\|_2$ the spectral norm. By abuse of notations, $\|\mathbf{A}\|_0 = \|\text{vec}(\mathbf{A})\|_0$. $\mathbf{I}$ is the identity matrix.

Objective. The goal of this paper is to obtain dictionaries structured as in (1). To this end, we can either impose the multi-layer sparse structure while learning the dictionary (*multi-layer sparse dictionary learning* task), or impose it on a pre-learned dictionary (*multi-layer sparse dictionary factorization* task). Let $\mathbf{X} \in \mathbb{R}^{d\times n}$ be a data matrix, each of its n columns $\mathbf{x}_i$ being a training vector, $\mathbf{D} \in \mathbb{R}^{d\times a}$ be a dictionary with a atoms and $\boldsymbol{\Gamma} \in \mathbb{R}^{a\times n}$ be the corresponding sparse representation matrix such that $\mathbf{X} \approx \mathbf{D}\boldsymbol{\Gamma}$. Typically $n \gg a \geq d$. In this framework, *multi-layer sparse dictionary learning* boils down to a factorization of the data matrix $\mathbf{X}$ into $M+1$ sparse factors (the rightmost factor being the sparse representation matrix $\boldsymbol{\Gamma}$), while *multi-layer sparse dictionary factorization* reduces to a factorization of the dictionary $\mathbf{D}$ into M sparse factors. These two problems are very similar, and introducing the matrix $\mathbf{Y}$ to embody the matrix to factorize ($\mathbf{X}$ or $\mathbf{D}$) and the integer Q to embody the desired number of factors, we propose to consider one general optimization problem:

$$\underset{\mathbf{S}_1,\ldots,\mathbf{S}_Q}{\text{Minimize}} \quad \left\|\mathbf{Y} - \prod_{j=1}^{Q} \mathbf{S}_j\right\|_F^2 + \sum_{j=1}^{Q} g_j(\mathbf{S}_j), \tag{2}$$

where the g_js are penalties promoting sparsity and possibly additional constraints, and the factors $\mathbf{S}_j$s are of compatible sizes. We call (2) the Multi-layer Sparse Approximation (MSA) problem. Dictionaries factorized or learned this way are called FAμST.

Related work. Similar matrix factorization problems have been studied recently in several domains. In dictionary learning, two main lines of work have begun to explore this way. In [4], the authors propose the sparse-KSVD algorithm (KSVDS) to learn a dictionary whose atoms are sparse linear combinations of atoms of a so-called *base dictionary*. The base dictionary should be associated with a fast algorithm (it takes the form (1)), so that the whole learned dictionary can be efficiently stored and manipulated. It can be seen as having

This work was supported in part by the European Research Council, PLEASE project (ERC-StG- 2011-277906). The authors wish to thank François Malgouyres and Olivier Chabiron for the fruitful discussions that helped in producing that work.

[1]The product being taken from right to left: $\prod_{i=1}^{N} \mathbf{A}_i = \mathbf{A}_N \cdots \mathbf{A}_1$

the $Q-2$ leftmost factors fixed in (2) (let us call it $\mathbf{D}_{\text{base}}$), the second factor being the sparse representation of the dictionary over the base dictionary ($\mathbf{D} = \mathbf{D}_{\text{base}}\mathbf{S}_2$), and the first being the sparse representation matrix $\mathbf{\Gamma} = \mathbf{S}_1$. One drawback with this formulation is that the learned dictionary is highly biased toward the base dictionary, which decreases adaptability to the training data. In [5], the authors propose to learn a dictionary in which each atom is the composition of several circular convolutions using sparse kernels with known supports, so that the dictionary is fast to manipulate. Their problem can be seen as (2), with the g_js corresponding to the M leftmost factors imposing sparse circulant matrices. This formulation is limited in nature to the case where the dictionary is well approximated by a product of sparse circulant matrices.

In statistics, a related problem is to approximately diagonalize a covariance matrix by a unitary matrix in factorized form (1), which can be addressed greedily [7, 8] using a fixed number of elementary Givens rotations. Here we consider a richer family of sparse factors and leverage recent non-convex optimization techniques. In machine learning, similar models were explored with various viewpoints. For example, sparse multi-factor NMF [9] essentially solves (2) with the Kullback-Leibler divergence as data fidelity term and all $\mathbf{S}_j$s constrained to be non-negative. Optimization relies on multiplicative updates, while we rely on proximal iterations. In the context of deep neural networks, identifiability guarantees on the network structure have been established with a generative model where consecutive network layers are sparsely connected at random, and non-linearities are neglected [10, 11]. The network structure in these studies matches the factorized structure (1), with each of the leftmost factors representing a layer of the network and the last one being its input. Apart from its hierarchical flavor, the identification algorithm in [10, 11] differs from the proximal method proposed here.

An objective somewhat related to that of having a computationally efficient dictionary is that of being able to rapidly compute the sparse code $\mathbf{\Gamma}$ corresponding to the training data $\mathbf{X}$ given the dictionary $\mathbf{D}$. Models addressing this issue have been proposed in [12] and [13]. Since most of sparse coding methods rely on products with the dictionary and its transpose, the method proposed here contributes also to this objective because dictionaries taking the form (1) naturally lead to fast vector and matrix multiplication.

3. OPTIMIZATION FRAMEWORK

We now express the considered optimization problem, and describe an algorithm with convergence guarantees to a stationary point.

3.1. Choice of the sparsity-inducing penalties

Considering a dictionary $\mathbf{D} = \prod_{j=1}^{M} \mathbf{S}_j$ as in (1), the storage and multiplication cost of the dictionary are determined by the number of non-zero entries in the factors $\mathbf{S}_j$. Indeed, storing/multiplying the dictionary in the factorized form will cost $\mathcal{O}(\sum_{j=1}^{M} \|\mathbf{S}_j\|_0)$, whereas classical dictionary learning methods would typically provide dense dictionaries for which storing/multiplying would cost $\mathcal{O}(da)$. This simple statement allows to introduce the *Relative Complexity* (RC):

$$\text{RC}(\mathbf{D}) := \left(\sum_{j=1}^{M} \|\mathbf{S}_j\|_0 \right) / da. \tag{3}$$

This quantity is clearly positive and should not exceed 1 in order to yield complexity savings. In practice, we choose as sparsity-inducing penalties the indicator functions $\delta_{\mathcal{E}_j}(\cdot)$ of subsets $\mathcal{E}_j$s of "ℓ_0 balls": $\{\mathbf{A} \in \mathbb{R}^{r_j \times c_j} : \|\mathbf{A}\|_0 \leq p_j\}$, hence RC $\leq \sum_{j=1}^{M} p_j/da$.

3.2. Coping with the scaling ambiguity.

To avoid scaling ambiguities, it is common [5, 9] to normalize the factors and introduce a multiplicative scalar λ in the data fidelity term. This results in the optimization problem:

$$\underset{\lambda, \mathbf{S}_1, \ldots, \mathbf{S}_Q}{\text{Minimize}} \quad \Psi(\mathbf{S}_1, \ldots, \mathbf{S}_Q, \lambda) := \frac{1}{2} \left\| \mathbf{Y} - \lambda \prod_{j=1}^{Q} \mathbf{S}_j \right\|_F^2 + \sum_{j=1}^{Q} \delta_{\mathcal{E}_j}(\mathbf{S}_j), \tag{4}$$

with $\mathcal{E}_j = \{\mathbf{A} \in \mathbb{R}^{r_j \times c_j} : \|\mathbf{A}\|_0 \leq p_j, \|\mathbf{A}\|_F = 1\}$.

3.3. Proposed algorithm

The optimization problem (4) is highly non-convex, and the sparsity enforcing part is non-smooth. We leverage recent advances in non-convex optimization to express an algorithm with convergence guarantees to a stationary point of the problem. In [14], the authors consider cost functions depending on N blocks of variables:

$$\Psi(\mathbf{x}_1, \ldots, \mathbf{x}_N) := H(\mathbf{x}_1, \ldots, \mathbf{x}_N) + \sum_{j=1}^{N} f_j(\mathbf{x}_j), \tag{5}$$

where the function H is smooth, and the f_js are proper and lower semi-continuous. To handle this cost function, the authors propose an algorithm called Proximal Alternating Linearized Minimization (PALM)[14], that updates alternatively each block of variable by a proximal (or projected in our case) gradient step.

PALM can be instantiated to handle the objective (4): there is indeed a match between (4) and (5) taking $N = Q+1$, $\mathbf{x}_j = \mathbf{S}_j$ for $j \in \{1 \ldots Q\}$, $\mathbf{x}_{Q+1} = \lambda$, H the data fidelity term, $f_j(\cdot) = \delta_{\mathcal{E}_j}(.)$ for $j \in \{1 \ldots Q\}$ and $f_{Q+1}(\cdot) = 0$ (there is no constraint on λ). Although we do not discuss it here, it is shown in our technical report [6] that all the conditions for PALM to converge to a stationary point of the objective are satisfied.

Projection operator $P_{\mathcal{E}_j}$. With $\mathcal{E}_j$s defined as in Section 3.2, the projection operator $P_{\mathcal{E}_j}(\cdot)$ simply keeps the p_j greatest entries (in absolute value) of its argument, sets all the other entries to zero, and then normalizes its argument so that it has unit norm. Regarding the scalar λ, we can consider its constraint set as $\mathcal{E}_{Q+1} = \mathbb{R}$, hence the projection operator is simply the identity mapping, $P_{\mathbb{R}}(x) = x$.

Gradient and Lipschitz moduli. To specify the iterations of PALM specialized to our problem, we fix the iteration i and the factor j, and denote $\mathbf{S}_j^i$ the factor that we are updating, $\mathbf{L} := \prod_{k=j+1}^{Q} \mathbf{S}_k^i$ what is on its left and $\mathbf{R} := \prod_{k=1}^{j-1} \mathbf{S}_k^{i+1}$ what is on its right (with the convention $\prod_{k \in \varnothing} \mathbf{S}_k = \mathbf{I}$). With these notations we have, when updating the jth factor $\mathbf{S}_j$: $H(\mathbf{S}_1^{i+1}, \ldots, \mathbf{S}_{j-1}^{i+1}, \mathbf{S}_j^i, \ldots, \mathbf{S}_Q^i, \lambda^i) = \frac{1}{2}\|\mathbf{Y} - \lambda^i \mathbf{L}\mathbf{S}_j^i\mathbf{R}\|_F^2$. The gradient of this smooth part of the objective with respect to the jth factor reads:

$$\nabla_{\mathbf{S}_j^i} H(\mathbf{S}_1^{i+1}, \ldots, \mathbf{S}_{j-1}^{i+1}, \mathbf{S}_j^i, \ldots, \mathbf{S}_Q^i, \lambda^i) = \lambda^i \mathbf{L}^T(\lambda^i \mathbf{L}\mathbf{S}_j^i\mathbf{R} - \mathbf{Y})\mathbf{R}^T,$$

which has a Lipschitz modulus with respect to $\left\|\mathbf{S}^i\right\|_F$: $L_j(\mathbf{L}, \mathbf{R}, \lambda^i) = (\lambda^i)^2 \|\mathbf{R}\|_2^2 . \|\mathbf{L}\|_2^2$. Once the first Q factors are updated we need to update λ. Denoting $\hat{\mathbf{Y}} = \prod_{k=1}^{Q} \mathbf{S}_k^{i+1}$, we get: $H(\mathbf{S}_1^{i+1}, \ldots, \mathbf{S}_Q^{i+1}, \lambda^i) = \frac{1}{2}\|\mathbf{Y} - \lambda^i \hat{\mathbf{Y}}\|_F^2$, and the gradient with respect to λ reads:

$$\nabla_{\lambda^i} H(\mathbf{S}_1^{i+1}, \ldots, \mathbf{S}_Q^{i+1}, \lambda^i) = \lambda^i \text{Tr}(\hat{\mathbf{Y}}^T\hat{\mathbf{Y}}) - \text{Tr}(\mathbf{Y}^T\hat{\mathbf{Y}}).$$

An explicit version of the algorithm is given in Algorithm 1, in which the factors are updated alternatively by a projected gradient

step (line 6) with a stepsize controlled by the Lipschitz modulus of the gradient (line 5). We can solve for λ directly at each iteration (line 9) because of the absence of constraint on it (see our report [6] for more precisions on the convergence conditions of PALM).

Initialization. Except when specified otherwise, initialization by default is done with $\lambda^0 = 1$, $\mathbf{S}_1^0 = \mathbf{0}$ and $\mathbf{S}_j^0 = \mathbf{I}$, $j \geq 2$, with the convention that for rectangular matrices the identity has ones on the main diagonal and zeroes elsewhere.

Algorithm 1 PALM for Multi-layer Sparse Approximation (`palm4MSA`)

Input: Data matrix or dictionary $\mathbf{Y}$; desired number of factors Q; constraint sets $\mathcal{E}_j,\ j \in \{1 \dots Q\}$; initialization $\{\mathbf{S}_j^0\}_{j=1}^Q$, λ^0; stopping criterion (e.g., number of iterations N_i).

1: **for** $i = 0$ to $N_i - 1$ **do**
2: **for** $j = 1$ to Q **do**
3: $\mathbf{L} \leftarrow \prod_{k=j+1}^{Q} \mathbf{S}_k^i$
4: $\mathbf{R} \leftarrow \prod_{k=1}^{j-1} \mathbf{S}_k^{i+1}$
5: Set $c_j^i > (\lambda^i)^2 \|\mathbf{R}\|_2^2 . \|\mathbf{L}\|_2^2$
6: $\mathbf{S}_j^{i+1} \leftarrow P_{\mathcal{E}_j}\left(\mathbf{S}_j^i - \frac{1}{c_j^i}\lambda^i \mathbf{L}^T(\lambda \mathbf{L}\mathbf{S}_j^i\mathbf{R} - \mathbf{Y})\mathbf{R}^T\right)$
7: **end for**
8: $\hat{\mathbf{Y}} \leftarrow \prod_{k=1}^{Q} \mathbf{S}_k^{i+1}$
9: $\lambda^{i+1} \leftarrow \frac{\mathrm{Tr}(\mathbf{Y}^T\hat{\mathbf{Y}})}{\mathrm{Tr}(\hat{\mathbf{Y}}^T\hat{\mathbf{Y}})}$
10: **end for**

Output: The estimated factorization:
$\lambda^{N_i}, \{\mathbf{S}_j^{N_i}\}_{j=1}^Q$ = `palm4MSA`($\mathbf{Y}$, Q, $\{\mathcal{E}_j\}_{j=1}^Q, \dots$)

3.4. Practical strategy

Algorithm 1 presented above factorizes a data matrix into Q sparse factors and converges to a stationary point of problem (4). However, while we are primarily interested in stationary points where the data fidelity term of the cost function is small, there is no guarantee that the algorithm converges to such a stationary point. This was observed on a very simple experiment where Algorithm 1 was used for a *multi-layer sparse dictionary factorization* task, with a dictionary $\mathbf{Y} = \mathbf{D}$ with a known factorization in M factors: $\mathbf{D} = \prod_{j=1}^{M} \mathbf{S}_j$, such as the Hadamard dictionary. The naive approach consisted in taking directly $Q = M$ in Algorithm 1, and setting the constraints so as to reflect the actual sparsity of the true factors. This simple strategy performed quite poorly in practice, and the attained local minimum was very often not satisfactory (high data fidelity term).

We noticed experimentally that taking fewer factors (Q small) and allowing more non-zero entries per factor led to better results in general. This observation suggested to adopt a hierarchical strategy. Indeed, when $\mathbf{Y} = \prod_{j=1}^{Q} \mathbf{S}_j$ is the product of Q sparse factors, it is also the product $\mathbf{Y} = \mathbf{T}_1\mathbf{S}_1$ of 2 factors with $\mathbf{T}_1 = \prod_{j=2}^{Q} \mathbf{S}_j$, so that $\mathbf{S}_1$ is sparser than $\mathbf{T}_1$. Our strategy is then to factorize the input matrix $\mathbf{Y}$ in 2 factors, one being sparse (corresponding to $\mathbf{S}_1$), and the other less sparse (corresponding to $\mathbf{T}_1$). The process can then be repeated on the less sparse factor $\mathbf{T}_1$, and so on until we attain the desired number Q of factors. Denoting $\mathbf{T}_k = \prod_{j=k+1}^{Q} \mathbf{S}_j$, a simple calculation shows that if we expect each $\mathbf{S}_j$ to have roughly $\mathcal{O}(h)$ non-zero entries per row, then $\mathbf{T}_k$ cannot have more than $\mathcal{O}(h^{Q-(k+1)})$ non-zero entries per row. This suggests to decrease exponentially the number of non-zero entries in $\mathbf{T}_k$ with k. This strategy turns out to be surprisingly effective and the attained local minima are very good, as illustrated in the next section. In a sense, this hierarchical approach is similar to learning a deep neural network layer by layer, as done in [10, 11]. Our preliminary theoretical analysis of the proposed approach suggests that the results of [10, 11] could in fact be leveraged to explain the observed empirical success.

The proposed hierarchical strategy is summarized in Algorithm 2, where we need to specify at each step the constraint sets related to the two factors. For that, let us introduce some notation: $\mathcal{E}_k$ will be the constraint set for the right factor and $\tilde{\mathcal{E}}_k$ the one for the left factor at the kth factorization. The global optimization step (line 5) is done by initializing `palm4MSA` with the current values of $\{\mathbf{S}_j\}_{j=1}^k$ and $\mathbf{T}_k$. It is here to keep an attach to the data matrix $\mathbf{Y}$. Roughly we can say that line 3 of the algorithm is here to yield complexity savings, whereas line 5 is here to improve data fidelity.

Algorithm 2 Hierarchical factorization

Input: The data matrix or dictionary $\mathbf{Y}$, the desired number of factors Q and the constraint sets $\mathcal{E}_k$ and $\tilde{\mathcal{E}}_k,\ k \in \{1 \dots Q-1\}$.

1: $\mathbf{T}_0 \leftarrow \mathbf{Y}$
2: **for** $k = 1$ to $Q - 1$ **do**
3: Factorize the residual $\mathbf{T}_{k-1}$ into 2 factors:
$\lambda', \{\mathbf{F}_2, \mathbf{F}_1\}$ = `palm4MSA`($\mathbf{T}_{k-1}$, 2, $\{\tilde{\mathcal{E}}_k, \mathcal{E}_k\}, \dots$)
4: $\mathbf{T}_k \leftarrow \lambda'\mathbf{F}_2$ and $\mathbf{S}_k \leftarrow \mathbf{F}_1$
5: Global optimization:
$\lambda, \left\{\mathbf{T}_k, \{\mathbf{S}_j\}_{j=1}^k\right\}$ = `palm4MSA`($\mathbf{Y}$, $k+1$, $\left\{\tilde{\mathcal{E}}_k, \{\mathcal{E}_j\}_{j=1}^k\right\}$, $\dots$)
6: **end for**
7: $\mathbf{S}_Q \leftarrow \mathbf{T}_{Q-1}$

Output: The estimated factorization $\lambda, \{\mathbf{S}_j\}_{j=1}^Q$.

In *multi-layer sparse dictionary learning*, the rightmost factor $\mathbf{S}_1$ corresponds to the sparse coefficients matrix $\mathbf{\Gamma}$, and the first iteration of factorization ($k = 1$) amounts to classical dictionary learning. It can thus be done with any classical dictionary learning method, such as K-SVD. Moreover, to avoid manipulating the coefficient matrix (which is often way bigger than the other factors) one can keep it fixed, and update it only by Orthogonal Matching Pursuit (OMP) [15] after the global optimization step (between lines 5 and 6). With this simple modification, Algorithm 2 resembles traditional dictionary learning [3] where update of the coefficients and update of the dictionary alternate, except that in Algorithm 2 the dictionary update is designed for complexity savings (reducing RC), and the coefficient update can be done efficiently taking advantage of the dictionary structure.

4. EXPERIMENTS

4.1. Dictionary based image denoising

Our experimental scenario for the *multi-layer sparse dictionary learning* task follows the workflow of a classical dictionary based image denoising. First, $n = 10000$ patches $\mathbf{x}_i$ of size 8×8 (dimension $d = 64$) are randomly picked from an input 512×512 noisy image (with PSNR= 22.1dB), and a dictionary is learned on these patches. Then the learned dictionary is used to denoise the entire input image by computing the sparse representation of all its patches in the dictionary using OMP, allowing each patch to use 5 dictionary atoms. The image is reconstructed by averaging the overlapping patches. Various dictionary learning methods were compared, each learning a four times overcomplete dictionary ($\mathbf{D} \in \mathbb{R}^{64\times256}$).

Settings of our algorithm. We tested several configurations for Algorithm 2, in order to get FAμST dictionaries and we present here one that exhibits a good tradeoff between relative complexity (RC) and adaptation to the data. Inspired by usual fast transforms, we chose a number of factors Q close to the logarithm of the signal dimension $d = 64$, here $Q = 5$. The sizes of the factors are: $\mathbf{\Gamma} = \mathbf{S}_1 \in \mathbb{R}^{256\times10000}$, $\mathbf{S}_2 \in \mathbb{R}^{64\times256}$ and $\mathbf{S}_3, \dots, \mathbf{S}_Q \in \mathbb{R}^{64\times64}$. Algorithm 2 was used with the modifications presented

	Learning (PSNR)	Denoising (PSNR)	Complexity (RC)
KSVD	**24.71**	27.55	1.00
ODL	24.62	27.51	1.00
KSVDS	24.16	27.64	0.41
FAμST	23.63	**29.38**	**0.13**

Table 1. Image denoising results, averaged over the standard image database taken from [22] (12 standard grey 512×512 images). The best result of each column is bold.

in the previous section i.e. the first factorization ($k = 1$ in Algorithm 2) is done by KSVD [16] and $\mathbf{S}_1$ was updated only by OMP allowing each patch to use 5 dictionary atoms. In this setting, $\mathcal{E}_1$ and $\tilde{\mathcal{E}}_1$ do not need to be specified, but implicitly correspond to $\mathcal{E}_1 = \{\mathbf{A} \in \mathbb{R}^{256\times 10000}, \|\mathbf{a}_i\|_0 \leq 5 \quad \forall i \in \{1 \dots 10000\}\}$ and $\tilde{\mathcal{E}}_1 = \{\mathbf{A} \in \mathbb{R}^{64\times 256}, \|\mathbf{a}_i\|_2 = 1 \quad \forall i \in \{1 \dots 256\}\}$. Regarding the other factorizations ($k > 1$ in Algorithm 2) we allowed in each factor 4 non-zero entries per column in average, so that the considered constraint sets were: $\mathcal{E}_2 = \{\mathbf{A} \in \mathbb{R}^{64\times 256}, \|\mathbf{A}\|_0 \leq 1024, \|\mathbf{A}\|_F = 1\}$ and for $k \in \{3 \dots Q-1\}$: $\mathcal{E}_k = \{\mathbf{A} \in \mathbb{R}^{64\times 64}, \|\mathbf{A}\|_0 \leq 256, \|\mathbf{A}\|_F = 1\}$. In order to decrease RC for the learned dictionary at each new factorization, the number of non-zero entries in the residual was divided by 2 at each step, starting from $p = 1.3 \times 2048$, leading to, for $k \in \{2 \dots Q-1\}$: $\tilde{\mathcal{E}}_k = \{\mathbf{A} \in \mathbb{R}^{64\times 64}, \|\mathbf{A}\|_0 \leq \frac{p}{2^{k-2}}, \|\mathbf{A}\|_F = 1\}$. The stopping criterion for `palm4MSA` was a number of iterations $N_i = 50$.

Baselines. We compared Algorithm 2 with the following methods. All methods involve a coefficient update step which is performed using OMP allowing each patch to use 5 dictionary atoms:

- KSVD [16]. We used the implementation described in [17], running 50 iterations (empirically sufficient to ensure convergence).
- Online Dictionary Learning (ODL) [18], running 200 iterations (empirically sufficient to ensure convergence).
- Sparse KSVD (KSVDS) [4], a method that seeks to bridge the gap between *learned dictionaries* and *analytic dictionaries*. The implementation of [4] is used, with $\mathbf{D}_{\text{base}}$ the four times overcomplete 2D-Discrete Cosine Transform (DCT) matrix and 50 iterations, ensuring convergence in practice. The columns of the estimated dictionary $\mathbf{D}$ are 6-sparse in $\mathbf{D}_{\text{base}}$.

Performance measures. The computational efficiency of the dictionary is measured through the *Relative Complexity* (RC) quantity introduced in Section 3.1. For the KSVDS method, we used the formula provided in [4] to compute RC. The quality of approximation at the learning stage and at the denoising stage is expressed using the classical PSNR measure (even though at the learning stage we are manipulating patches and not entire images).

Discussion of the results. Table 1 summarizes the performance of the different methods. First of all, the proposed method is able to provide dictionaries that are much more computationally efficient than the others (RC$= 0.13$). Second, and perhaps more surprisingly, while the proposed method performs worse than the others at the learning stage, it outperforms them at the denoising stage. This indicates that its generalization properties are better [19, 20, 21], which is consistent with the lower number of parameters to learn compared to KSVD and ODL. Regarding KSVDS, its poorer performance may be explained by the fact that it is highly biased toward its base dictionary. Moreover, we suspect the proposed structure to be more intrinsically unable to fit noise than standard dictionaries.

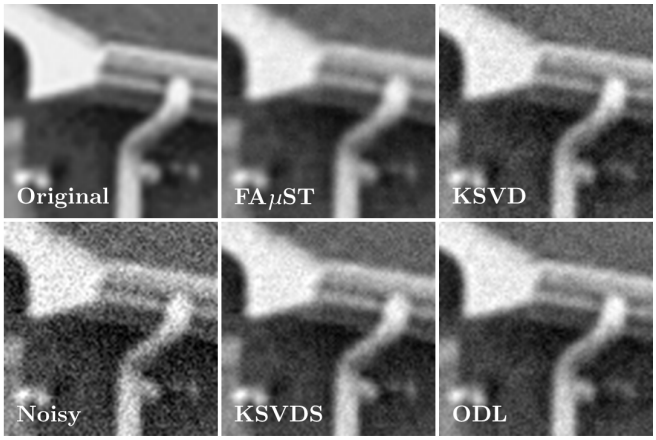

Fig. 1. Denoising example: zoom on the "house" standard image.

4.2. Retrieving the fast Hadamard transform

Here we consider *multi-layer sparse dictionary factorization*, given a matrix $\mathbf{Y} = \mathbf{D}$ with known factorization, $\mathbf{D} = \prod_{j=1}^{M} \mathbf{S}_j$: in Section 3.4, we evoked the failure of Algorithm 1 for this factorization problem. In contrast, Figure 2 illustrates the result of the proposed hierarchical strategy (Algorithm 2) with $\mathbf{D}$ the Hadamard dictionary in dimension $n = 32$. The obtained factorization is exact and *as good as the reference one* in terms of complexity savings ($2n \log n$ non-zero entries in the factors), so in a way the chased butterflies were caught. The running time is less than a second. Factorization of the Hadamard matrix in dimension up to $n = 1024$ showed identical behaviour, with running time $O(n^2)$ up to ten minutes.

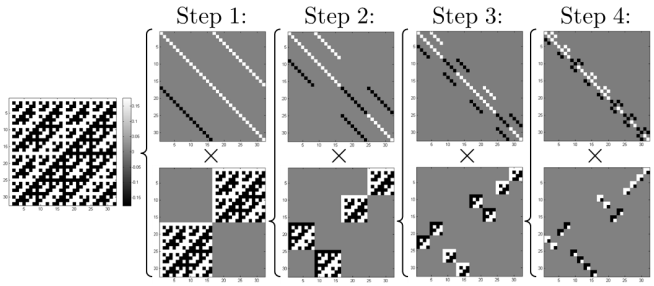

Fig. 2. Hierarchical factorization of the Hadamard matrix of size 32×32. The matrix is iteratively factorized in 2 factors, until we have $Q = 5$ factors, each having $p = 64$ non-zero entries.

5. CONCLUSION

We proposed a matrix factorization framework with convergence guarantees that provides a flexible tradeoff between computational efficiency and data fidelity. It shows promising results in image denoising and is able to automatically retrieve an ideally compact factorization of the fast Hadamard transform. Besides the obvious need to better understand the role of its parameters in the control of the desired tradeoff, a particular challenge will be to leverage the gained complexity to speed up the factorization process itself, in order to *efficiently learn* efficient dictionaries. Moreover, the algorithm underlying the factorization is very general, and it would be interesting to take advantage of its versatility in order to incorporate other structures in the factors. This line of work could lead for example to learn data-driven efficient wavelets on graphs [23], by constraining the support of the factors. Finally, a concern in the numerical analysis community is to be able to approximate certain integral operators with a butterfly structure [24, 25]. The method proposed in this paper could help finding automatically such approximate fast transforms.

6. REFERENCES

[1] James Cooley and John Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[2] Stéphane Mallat. A theory for multiresolution signal decomposition : the wavelet representation. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 11:674–693, June 1989.

[3] Ron Rubinstein, A.M. Bruckstein, and Michael Elad. Dictionaries for Sparse Representation Modeling. *Proceedings of the IEEE*, 98(6):1045 –1057, 2010.

[4] Ron Rubinstein, Michael Zibulevsky, and Michael Elad. Double sparsity: learning sparse dictionaries for sparse signal approximation. *IEEE Transactions on Signal Processing*, 58(3):1553–1564, March 2010.

[5] Olivier Chabiron, Francois Malgouyres, Jean-Yves Tourneret, and Nicolas Dobigeon. Toward fast transform learning. Technical report, November 2013.

[6] Luc Le Magoarou and Rémi Gribonval. Learning computationally efficient dictionaries and their implementation as fast transforms. *CoRR*, abs/1406.5388, 2014.

[7] Ann B. Lee, Boaz Nadler, and Larry Wasserman. Treelets - an adaptive multi-scale basis for sparse unordered data. *The Annals of Applied Statistics*, 2(2):435–471, July 2008.

[8] Guangzhi Cao, L.R. Bachega, and C.A. Bouman. The sparse matrix transform for covariance estimation and analysis of high dimensional signals. *Image Processing, IEEE Transactions on*, 20(3):625–640, 2011.

[9] Siwei Lyu and Xin Wang. On algorithms for sparse multi-factor NMF. In *Advances in Neural Information Processing Systems 26*, pages 602–610. 2013.

[10] Behnam Neyshabur and Rina Panigrahy. Sparse matrix factorization. *CoRR*, abs/1311.3315, 2013.

[11] Sanjeev Arora, Aditya Bhaskara, Rong Ge, and Tengyu Ma. Provable bounds for learning some deep representations. *CoRR*, abs/1310.6343, 2013.

[12] Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th Annual International Conference on Machine Learning*, ICML '10, pages 399–406, 2010.

[13] Pablo Sprechmann, Alexander M. Bronstein, and Guillermo Sapiro. Learning efficient sparse and low rank models. *CoRR*, abs/1212.3631, 2012.

[14] Jérôme Bolte, Shoham Sabach, and Marc Teboulle. Proximal Alternating Linearized Minimization for nonconvex and nonsmooth problems. *Mathematical Programming*, pages 1–36, 2013.

[15] S.G. Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415, December 1993.

[16] M. Aharon, M. Elad, and A. Bruckstein. K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation. *Signal Processing, IEEE Transactions on*, 54(11):4311–4322, Nov 2006.

[17] Ron Rubinstein, Michael Zibulevsky, and Michael Elad. Efficient Implementation of the K-SVD Algorithm using Batch Orthogonal Matching Pursuit. Technical report, 2008.

[18] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online learning for matrix factorization and sparse coding. *Journal of Machine Learning Research*, 11(1):19–60, January 2010.

[19] Rémi Gribonval, Rodolphe Jenatton, Francis Bach, Martin Kleinsteuber, and Matthias Seibert. Sample Complexity of Dictionary Learning and other Matrix Factorizations. *ArXiv e-prints*, December 2013.

[20] Daniel Vainsencher, Shie Mannor, and Alfred M Bruckstein. The sample complexity of dictionary learning. *The Journal of Machine Learning Research*, 12:3259–3281, 2011.

[21] Augusto Maurer and Massimiliano Pontil. K -dimensional coding schemes in Hilbert spaces. *IEEE Transactions on Information Theory*, 56(11):5839–5846, 2010.

[22] http://www.imageprocessingplace.com/.

[23] David K. Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129 – 150, 2011.

[24] Michael O'Neil, Franco Woolfe, and Vladimir Rokhlin. An algorithm for the rapid evaluation of special function transforms. *Applied and Computational Harmonic Analysis*, 28(2):203 – 226, 2010. Special Issue on Continuous Wavelet Transform in Memory of Jean Morlet, Part I.

[25] Emmanuel J. Candès, Laurent Demanet, and Lexing Ying. A fast butterfly algorithm for the computation of Fourier integral operators. *Multiscale Modeling & Simulation*, 7(4):1727–1750, 2009.