Chasing butterflies: In search of efficient dictionaries
Luc Le Magoarou, Rémi Gribonval

To cite this version:

HAL Id: hal-01104696
https://hal.archives-ouvertes.fr/hal-01104696v2
Submitted on 23 Jan 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Dictionary learning aims at finding a frame (called dictionary) in which training data admits a sparse representation. Traditional dictionary learning is limited to relatively small-scale problems, because dense matrices associated to high-dimensional dictionaries can be costly to manipulate, both at the learning stage and when using this dictionary for tasks such as sparse coding. In this paper, inspired by usual fast transforms, we consider a dictionary structure allowing cheaper manipulation, and we propose a learning algorithm imposing this structure. The approach is demonstrated experimentally with the 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 $D$, one can find a sparse representation vector $\gamma$ such that a vector $x$ of interest can be approximated as $x \approx D\gamma$.

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) \cite{1} or the Discrete Wavelet Transform (DWT) \cite{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 \cite{3}.

In this paper the goal is to design dictionaries that are well adapted to the data as learned dictionaries, while fast to manipulate and as cheap to store as analytic ones. Generalizing approaches introduced recently in \cite{4} and \cite{5}, we propose here a strategy to tackle this problem, that may seem unrealistic at first. 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

$$D = \prod_{j=1}^{M} S_j, \quad (1)$$

where each factor $S_j$ 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: $A$; vectors by bold lower-case letters: $a$; the $i$th column of a matrix $A$ by: $a_i$; and sets by calligraphic symbols: $\mathcal{A}$. The standard vectorization operator is denoted by $\text{vec}(\cdot)$. The $0$-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, $\|A\|_0 = \|\text{vec}(A)\|_0$. $I$ 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 $X \in \mathbb{R}^{d \times n}$ be a data matrix, each of its $n$ columns $x_i$ being a training vector, $D \in \mathbb{R}^{d \times n}$ be a dictionary with $a$ atoms and $Y \in \mathbb{R}^{a \times n}$ be the corresponding sparse representation matrix such that $X \approx DY$. Typically $n \gg a \geq d$. In this framework, multi-layer sparse dictionary learning boils down to a factorization of the data matrix $X$ into $M + 1$ sparse factors (the rightmost factor being the sparse representation matrix $Y$), while multi-layer sparse dictionary factorization reduces to a factorization of the dictionary matrix $D$ into $M$ sparse factors. These two problems are very similar, and introducing the matrix $Y$ to embody the matrix to factorize ($X$ or $D$) and the integer $Q$ to embody the desired number of factors, we propose to consider one general optimization problem:

$$\text{Minimize } \sum_{S_i} \sum_{j=1}^{Q} g_j(S_j), \quad (2)$$

where the $g_j$ are penalties promoting sparsity and possibly additional constraints, and the factors $S_j$ are of compatible sizes. We call (2) the Multi-layer Sparse Approximation (MSA) problem.

### 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 \cite{6}, the authors propose the sparse-KSVD algorithm (KSVD) 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 \cite{4}), so that the whole learned dictionary can be efficiently stored and manipulated. It can be seen as having the $Q - 2$ leftmost factors fixed in (2) (let us call it $D_{\text{base}}$), the second factor being the sparse representation of the dictionary over the base.
dictionary \(D = D_{\text{base}}S_2\), and the first being the sparse representation matrix \(\mathbf{F} = S_1\). 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 [3], 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 \(g_j\)'s corresponding to the \(M\) leftmost factors imposing sparse circular matrices. This formulation is limited in nature to the case where the dictionary is well approximated by a product of sparse circular matrices.

In statistics, a related problem is to approximately diagonalize a covariance matrix by a unitary matrix in factorized form \(\Pi\), which can be addressed greedily [6,9] 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 points of view. For example, sparse multi-factor NMF [8] can be seen as solving problem [2] with the Kullback-Leibler divergence as data fidelity term and all \(S_i, s\) 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 [9,10]. The network structure in these studies matches the factorized structure \(\Pi\), 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 [9,10] 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 \(\mathbf{F}\) corresponding to the training data \(\mathbf{X}\) given the dictionary \(\Pi\). Models addressing this issue have been proposed in [11] and [12]. 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 \(\Pi\) 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 \(D = \bigoplus_{j=1}^M S_j\) as in [1], the storage and multiplication cost of the dictionary are determined by the number of non-zero entries in the factors \(S_j\). Indeed, storing/multiplying the dictionary in the factorized form will cost \(O(\sum_j \|S_j\|_0)\), whereas classical dictionary learning methods would typically provide dense dictionaries for which storing/multiplying would cost \(O(da)\). This simple statement allows to introduce the Relative Complexity (RC):

\[
\text{RC}(D) := \left(\sum_{j=1}^M \|S_j\|_0\right)/da.
\]

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 \(\delta_{E_j}(\cdot)\) of subsets \(E_j\) of \(\mathbb{R}^{c_j}\) balls: \(\{A \in \mathbb{R}^{c_j} : \|A\|_0 \leq p_j\}\), hence \(\text{RC} \leq \sum_{j=1}^M p_j/da\).

#### 3.2. Coping with the scaling ambiguity

To avoid scaling ambiguities, it is common [5,8] to normalize the factors and introduce a multiplicative scalar \(\lambda\) in the data fidelity term. This results in the optimization problem:

\[
\begin{align*}
\text{Minimize} \quad & \lambda, S_1, \ldots, S_Q, \lambda \quad \Psi(S_1, \ldots, S_Q, \lambda) := \frac{1}{2} \|\mathbf{Y} - \lambda \sum_{j=1}^Q S_j\|_F^2 + \sum_{j=1}^Q \delta_{E_j}(S_j),
\end{align*}
\]

with \(E_j = \{A \in \mathbb{R}^{c_j} : \|A\|_0 \leq p_j, \|A\|_F = 1\}\).

### 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 [13], the authors consider cost functions depending on \(N\) blocks of variables:

\[
\Psi(x_1, \ldots, x_N) := H(x_1, \ldots, x_N) + \sum_{j=1}^N f_j(x_j),
\]

where the function \(H\) is smooth, and the \(f_j\)s are proper and lower semi-continuous. To handle this cost function, the authors propose an algorithm called Proximal Alternating Linearized Minimization (PALM [13]), 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 \(N = Q + 1\), \(x_1 = S_1\) for \(j \in \{1, \ldots, Q\}\), \(x_{Q+1} = \lambda\), \(H\) the data fidelity term, \(f_j(\cdot) = \delta_{E_j}(\cdot)\) for \(j \in \{1, \ldots, Q\}\) and \(f_{Q+1}(\cdot) = 0\) (there is no constraint on \(\lambda\)). Although we do not discuss it here, it is shown in our technical report [14] that all the conditions for PALM to converge to a stationary point of the objective are satisfied.

Projection operator \(P_{E_j}\). With \(E_j\) defined as in Section 3.2 the projection operator \(P_{E_j}\) 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 \(\lambda\), we can consider its constraint set as \(E_{Q+1} = \mathbb{R}\), hence the projection operator is simply the identity mapping, \(P_{E_j}(x) = x\).

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 \(S_j\) the factor that we are updating. \(L := \prod_{k=1}^N S_k\) what is on its left and \(R := \prod_{k=i+1}^N S_k\) what is on its right (with the convention \(\prod_{k \in \emptyset} S_k = I\)). With these notations we have, when updating the \(j\)th factor \(S_j\):

\[
\nabla S_j H(S_1^{i+1}, \ldots, S_j^{i+1}, S_j, \ldots, S_Q^{i+1}; \lambda^i) = \lambda^i \mathbf{L}^T (\lambda^i \mathbf{L} S_j^i \mathbf{R} - \mathbf{Y}) \mathbf{R}^T,
\]

which has a Lipschitz modulus with respect to \(\|S_j\|_F^2\):

\[
L_j(L, R, \lambda^i) = (\lambda^i)^2 \|\mathbf{R}\|^2 \|\mathbf{L}\|^2.\]

Once the first \(Q\) factors are updated we need to update \(\lambda^i\). Denoting \(\hat{Y} = \prod_{k=1}^Q S_k^{i+1}\), we get:

\[
\nabla \lambda H(S_1^{i+1}, \ldots, S_Q^{i+1}, \lambda^i) = \lambda^i \text{Tr}(\hat{Y}^T \hat{Y}) - \text{Tr}(Y^T \hat{Y}),
\]

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 \(\lambda\) directly at each iteration.
Indeed, when general. This observation suggested to adopt a hierarchical strategy. and allowing more non-zero entries per factor led to better results in practice, and the attained local minimum performed quite poorly in practice, and the attained local minimum also the product directly was observed on a very simple experiment where Algorithm 1 was used for a dictionary factorization task, with a dictionary Y = D with a known factorization in M factors: D = \( \prod_{j=1}^{M} S_j \), 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 Y = \( \prod_{j=1}^{Q} S_j \) is the product of Q sparse factors, it is also the product Y = T1S1 of 2 factors with T1 = \( \prod_{j=2}^{Q} S_j \), so that S1 is sparser than T1. Our strategy is then to factorize the input matrix Y in 2 factors, one being sparse (corresponding to S1), and the other less sparse (corresponding to T1). The process can then be repeated on the less sparse factor T1, and so on until we attain the desired number Q of factors. Denoting T_k = \( \prod_{j=k+1}^{Q} S_j \), a simple calculation shows that if we expect each S_j to have roughly O(h) non-zero entries per row, then T_k cannot have more than O(h^{Q-(k+1)}) non-zero entries per row. This suggests to decrease exponentially the number of non-zero entries in 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 [9,10]. Our preliminary theoretical analysis of the proposed approach suggests that the results of [9,10] 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: \( E_k \) will be the constraint set for the right factor and \( \tilde{E}_k \) the one for the left factor at the k-th factorization. The global optimization step (line 5) is done by initializing palm4MSA with the current values of \( \{S_j\}_{j=1}^{k-1} \) and \( T_k \). It is here to keep an attach to the data matrix 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 1 PALM for Multi-layer Sparse Approximation (palm4MSA)

**Input:** Data matrix or dictionary Y; desired number of factors Q; constraint sets \( E_j, j \in \{1 \ldots Q\} \); initialization \( \{S_j\}_{j=1}^{Q} \); \( \lambda^{0} \); 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. \( L \leftarrow \prod_{k=j+1}^{Q} S_k \)
4. \( R \leftarrow \prod_{k=1}^{j-1} S_k^{-1} \)
5. Set \( c_j > (\lambda^{j+1})^2 \|R\|_F^2 \|L\|_F^2 \)
6. \( S_j^{-1} \leftarrow P_E \left( S_j^{-1} - \frac{1}{c_j} \lambda^{j+1} L^T (\lambda LS_j^2 R - Y)R^T \right) \)
7. end for
8. \( Y \leftarrow \prod_{k=1}^{Q} S_k^{-1} \)
9. \( \lambda^{i+1} \leftarrow \frac{\text{tr}(S)}{\text{tr}(Y)} \)
10. end for

**Output:** The estimated factorization:
\( \lambda^{N_i}, \{S_j\}_{j=1}^{Q} = \text{palm4MSA}(Q, \{E_j\}_{j=1}^{Q}) \)

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 dictionary factorization task, with a dictionary Y = D with a known factorization in M factors: D = \( \prod_{j=1}^{M} S_j \), 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 Y = \( \prod_{j=1}^{Q} S_j \) is the product of Q sparse factors, it is also the product Y = T1S1 of 2 factors with T1 = \( \prod_{j=2}^{Q} S_j \), so that S1 is sparser than T1. Our strategy is then to factorize the input matrix Y in 2 factors, one being sparse (corresponding to S1), and the other less sparse (corresponding to T1). The process can then be repeated on the less sparse factor T1, and so on until we attain the desired number Q of factors. Denoting T_k = \( \prod_{j=k+1}^{Q} S_j \), a simple calculation shows that if we expect each S_j to have roughly O(h) non-zero entries per row, then T_k cannot have more than O(h^{Q-(k+1)}) non-zero entries per row. This suggests to decrease exponentially the number of non-zero entries in 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 [9,10]. Our preliminary theoretical analysis of the proposed approach suggests that the results of [9,10] 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: \( E_k \) will be the constraint set for the right factor and \( \tilde{E}_k \) the one for the left factor at the k-th factorization. The global optimization step (line 5) is done by initializing palm4MSA with the current values of \( \{S_j\}_{j=1}^{k-1} \) and \( T_k \). It is here to keep an attach to the data matrix 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 Y, the desired number of factors Q and the constraint sets \( E_k \) and \( \tilde{E}_k \), \( k \in \{1 \ldots Q - 1\} \).

1. \( T_0 \leftarrow Y \)
2. for \( k = 1 \) to \( Q - 1 \) do
3. Factorize the residual \( T_{k-1} \) into 2 factors:
   \( \lambda^{k-1} \{F_2, F_1\} = \text{palm4MSA}(T_{k-1}, 2, \{E_k, \tilde{E}_k\}, \ldots) \)
4. \( T_k \leftarrow \lambda^{k} F_2 \) and \( S_k \leftarrow F_1 \)
5. Global optimization:
   \( \lambda^{k} \{T_k, \{S_j\}_{j=1}^{k-1}\} = \text{palm4MSA}(Y, k + 1, \{E_k, \tilde{E}_j\}_{j=1}^{k}) \)
6. end for
7. \( S_Q \leftarrow T_{Q-1} \)

**Output:** The estimated factorization \( \lambda^{Q} \{S_j\}_{j=1}^{Q} \).

In the case of structured dictionary learning, the first factor \( S_1 \) corresponds to the sparse coefficients matrix \( \Gamma \), 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) after the global optimization step (between lines 5 and 6). This simple modification of Algorithm 2 makes the hierarchical strategy look like a traditional dictionary learning algorithm [3] where update of the coefficients and update of the dictionary alternate, except that in the present case the dictionary update is different and seeks to make 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 structured dictionary learning task follows the workflow of a classical dictionary based image denoising. First, \( n = 10000 \) patches \( x_i \) of size 8 \times 8 (dimension \( d = 64 \)) are randomly picked from an input 512 \times 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 \( D \in \mathbb{R}^{64 \times 2560} \).

Settings of our algorithm. We tested several configurations for Algorithm 2 and we present here one that exhibits a good tradeoff between relative complexity (RC) and adaptation to the data (we call it MSA for Multi-layer Sparse Approximation). 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: \( \Gamma = S_1 \in \mathbb{R}^{256 \times 10000}, S_2 \in \mathbb{R}^{64 \times 256} \) and \( S_3, \ldots, S_Q \in \mathbb{R}^{64 \times 64} \). Algorithm 2 was used with the modifications presented in the previous section i.e. the first factorization (\( k = 1 \) in Algorithm 2) is done by K-SVD [10] and \( S_1 \) was updated only...
by OMP allowing each patch to use 5 dictionary atoms. In this setting, $\mathcal{E}_1$ and $\mathcal{E}_2$ do not need to be specified, but implicitly correspond to $\mathcal{E}_1 = \{ A \in \mathbb{R}^{256 \times 10000}, \|a\|_0 \leq 5 \ \forall i \in \{1 \ldots 10000\}\}$ and $\mathcal{E}_2 = \{ A \in \mathbb{R}^{64 \times 256}, \|a\|_2 = 1 \ \forall i \in \{1 \ldots 256\}\}$. 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: $\mathcal{E}_2 = \{ A \in \mathbb{R}^{64 \times 256}, \|a\|_0 \leq 1024, \|A\|_F = 1\}$ and for $k \in \{3 \ldots Q - 1\}$: $\mathcal{E}_k = \{ A \in \mathbb{R}^{64 \times 64}, \|a\|_0 \leq 256, \|A\|_F = 1\}$. 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 $k \in \{2 \ldots Q - 1\}$: $\mathcal{E}_k = \{ A \in \mathbb{R}^{64 \times 64}, \|a\|_0 \leq \frac{p}{2^k}, \|A\|_F = 1\}$. The stopping criterion for palm4MSA was a number of iterations $N_i = 50$.

**Baselines.** We compared Algorithm 2 with the following methods.

- **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 $D_{base}$, the 4 times overcomplete 2D-Discrete Cosine Transform (DCT) matrix and 50 iterations, ensuring convergence in practice. The columns of the estimated dictionary $D$ are 6-sparse in $D_{base}$.

**Performance measures.** The computational efficiency of the dictionary is measured through the Relative Complexity (RC) quantity introduced in Section 3.3 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 factorized structure to have the interesting property of being more intrinsically unable to fit noise than standard dictionaries.

### 4.2. Retrieving the fast Hadamard transform

Here is presented an experiment regarding the dictionary factorization task. Consider a data matrix $Y = D$ with a known factorization $D = \prod_{i=1}^{M} S_i$; 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 $D$ 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. 1.** Example of denoising result. It is a zoom on a small part of the “house” standard image.

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

#### Table 1. Image denoising results, averaged over the standard image database taken from [23] (12 standard grey 512 x 512 images). The best result of each column is bold.

<table>
<thead>
<tr>
<th>Method</th>
<th>Learning (PSNR)</th>
<th>Denoising (PSNR)</th>
<th>Complexity (RC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>KSVD</td>
<td>24.71</td>
<td>27.55</td>
<td>1.00</td>
</tr>
<tr>
<td>ODL</td>
<td>24.62</td>
<td>27.51</td>
<td>1.00</td>
</tr>
<tr>
<td>KSVDS</td>
<td>24.16</td>
<td>27.64</td>
<td>0.41</td>
</tr>
<tr>
<td>MSA</td>
<td>23.63</td>
<td><strong>29.38</strong></td>
<td><strong>0.13</strong></td>
</tr>
</tbody>
</table>

**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 [24, 25]. The method proposed in this paper could help finding automatically such approximate fast transforms.
6. REFERENCES


