



Learning computationally efficient dictionaries and their implementation as fast transforms

Luc Le Magoarou, Rémi Gribonval

► To cite this version:

Luc Le Magoarou, Rémi Gribonval. Learning computationally efficient dictionaries and their implementation as fast transforms. 2014. hal-01010577v2

HAL Id: hal-01010577

<https://inria.hal.science/hal-01010577v2>

Preprint submitted on 30 Jun 2014 (v2), last revised 26 Feb 2015 (v3)

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.

Learning computationally efficient dictionaries and their implementation as fast transforms

Luc Le Magoarou Rémi Gribonval

Inria

Centre Inria Rennes - Bretagne Atlantique

{luc.le-magoarou, remi.gribonval}@inria.fr

Abstract

Dictionary learning is a branch of signal processing and machine learning that aims at finding a frame (called dictionary) in which some training data admits a sparse representation. The sparser the representation, the better the dictionary. The resulting dictionary is in general a dense matrix, and its manipulation can be computationally costly both at the learning stage and later in the usage of this dictionary, for tasks such as sparse coding. Dictionary learning is thus limited to relatively small-scale problems. In this paper, inspired by usual fast transforms, we consider a general dictionary structure that allows cheaper manipulation, and propose an algorithm to learn such dictionaries –and their fast implementation– over training data. The approach is demonstrated experimentally with the factorization of the Hadamard matrix and with synthetic dictionary learning experiments.

1 Introduction

Sparse representations using dictionaries are a popular way of providing concise descriptions of high-dimensional vectors. The goal of dictionary learning is to find an appropriate dictionary \mathbf{D} allowing the sparse approximation of a training collection, gathered in a data matrix \mathbf{X} , as:

$$\mathbf{X} \approx \mathbf{D}\mathbf{\Gamma}, \quad (1)$$

where $\mathbf{\Gamma}$ has sparse columns. Historically, the only way to come up with a dictionary was to analyse mathematically the data and derive a “simple” formula to construct the dictionary. Dictionaries designed this way are called *analytic dictionaries* [1] (e.g.: associated to Fourier, wavelets and Hadamard transforms). Due to the relative simplicity of analytic dictionaries, they are often associated with a fast algorithm such as the Fast Fourier Transform (FFT) [2] or the Discrete Wavelet Transform (DWT) [3]. On the other hand, the development of modern computers allowed the surfacing of automatic methods that learn a dictionary directly from the data. Such *learned dictionaries* are usually well adapted to the data at hand, but due to their lack of structure, they do not lead to fast algorithms and are costly to store. A survey on dictionaries, analytic or learned, can be found in [1].

Can one design dictionaries as well adapted to the data as learned dictionaries, while as fast to manipulate and as cheap to store as analytic ones? Such an objective can seem unrealistic, but in [4], and more recently in [5], the authors introduced new dictionary structures that seek to bridge the gap between the two categories. The model we introduce actually generalizes these approaches. We build on the simple observation that the fast transforms associated with analytic dictionaries can be seen as consecutive multiplications of the input vector by sparse matrices, indicating that such dictionaries can be expressed as a product of sparse matrices¹:

$$\mathbf{D} = \prod_{j=1}^M \mathbf{S}_j. \quad (2)$$

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

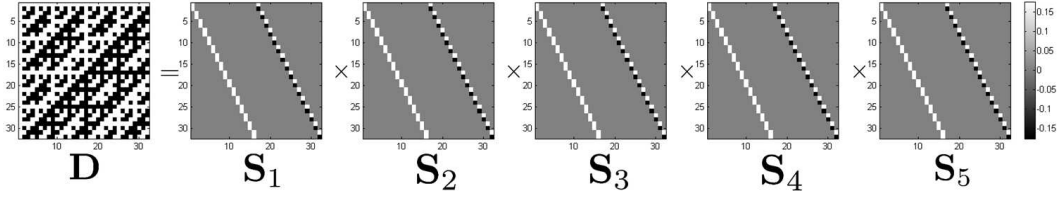


Figure 1: The Hadamard dictionary in size $n \times n$ with $n = 32$ (left) and its factorization. The dictionary is totally dense so that the naive storage and multiplication cost $\mathcal{O}(n^2 = 1024)$. On the other hand, we show the factorization of the dictionary in $\log_2(n) = 5$ factors, each having $2n = 64$ non-zero entries, so that the storage and multiplication in the factorized form cost $\mathcal{O}(2n \log_2(n) = 320)$.

This factorizable structure is precisely what enables fast algorithms to multiply a vector by the dictionary or its adjoint. For example each step of the butterfly radix-2 FFT can be seen as the multiplication by a sparse matrix having only two non-zero entries per row and per column, which leads to the well-known complexity savings. Another example can be found in Figure 1 where we show the Hadamard dictionary along with its factorized form.

Our objective is thus to learn a factorizable dictionary (i.e. taking the form of eq.(2)), making it intrinsically fast to manipulate and cheap to store. We will express this as an highly non-convex optimization problem, and rely on recent advances in optimization such as the PALM algorithm proposed in [6] to address it. In Section 2 we formulate the problem and link our work with relevant others, in Section 3 we present a general algorithm to solve it, and finally in Section 4 we present experimental results showing the interest of the proposed method. In particular, we demonstrate its ability to factor the Hadamard matrix in a way that enables its multiplication by an arbitrary vector as efficiently as with the fast Hadamard transform.

2 Problem formulation and related works

Notation. Throughout this paper, matrices are denoted by bold upper-case letters: \mathbf{A} . Vectors are denoted by bold lower-case letters: \mathbf{a} . The i th column of a matrix \mathbf{A} is denoted by: \mathbf{a}_i . Sets are denoted by calligraphical symbols: \mathcal{A} . The standard vectorization operator is denoted by $\text{vec}(\cdot)$ and the Kronecker product by \otimes . The usual ℓ_0 -norm is denoted by $\|\cdot\|_0^0$ (it counts the number of non-zero elements), $\|\cdot\|_F$ denotes the Frobenius norm, and $\|\cdot\|_2$ the operator norm.

Objective. As stated in the introduction, our goal is to learn dictionaries that are intrinsically fast to manipulate and cheap to store. In order to meet these requirements, we impose that our dictionary be the product of several sparse matrices. Mathematically speaking, let $\mathbf{X} \in \mathbb{R}^{d \times n}$ be our data matrix, each of its n columns \mathbf{x}_i being a training vector, $\mathbf{D} \in \mathbb{R}^{d \times a}$ be our dictionary with a atoms and $\mathbf{\Gamma} \in \mathbb{R}^{a \times n}$ be the corresponding sparse representation matrix such that $\mathbf{X} \approx \mathbf{D}\mathbf{\Gamma}$. In order to meet the requirements and be intrinsically fast, \mathbf{D} must take the form of eq.(2), where the \mathbf{S}_j s are sparse matrices in $\mathbb{R}^{a_j \times a_{j+1}}$ with $a_1 = d$ and $a_{M+1} = a$. Then, denoting $\mathbf{S}_{M+1} = \mathbf{\Gamma}$ for ease of notation, our goal is to find the sparse factors \mathbf{S}_j s such that:

$$\mathbf{X} \approx \prod_{j=1}^{M+1} \mathbf{S}_j. \quad (3)$$

Under this form, our problem amounts to a factorisation of the data matrix into $M+1$ sparse factors, thus it can be cast as a general optimization problem:

$$\underset{\mathbf{S}_1, \dots, \mathbf{S}_{M+1}}{\text{Minimize}} \quad d(\mathbf{X}, \prod_{j=1}^{M+1} \mathbf{S}_j) + \sum_{j=1}^{M+1} g_j(\mathbf{S}_j), \quad (4)$$

where $d(\cdot, \cdot)$ is some distance measure and the g_j s are sparsity-seeking penalties or constraints. This general optimization problem has been studied recently by various authors in several domains.

Related works. For dictionary learning, as mentioned in the introduction, two main works have begun to explore this way. In [4], the authors propose to learn a dictionary which 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 of eq.(2)), so that the whole learned dictionary can be efficiently stored and manipulated. It can be seen as having the $M - 1$ leftmost factors fixed in eq.(3) (let us call it \mathbf{D}_{base}), the M th factor being the sparse representation of the dictionary over the base dictionary ($\mathbf{D} = \mathbf{D}_{\text{base}} \mathbf{S}_M$), and the $M + 1$ th being the sparse representation of the training data over the learned dictionary. The major drawback with this formulation is that the learned dictionary is highly biased toward the base dictionary, so that we do not have full adaptability. In [5], the authors propose to learn a dictionary in which each atom is the composition of several circular convolutions with sparse kernels, so that the dictionary is fast to manipulate. Their model can be seen as eq.(3), with the g_j s 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 and data analysis, some researchers have been interested in statistical models in which the covariance matrix of the data takes the form of eq.(3), so that estimating this covariance matrix amounts to the problem of eq.(4). Recent representative works in this direction are [7] and [8].

Even more recently, similar models were proposed in machine learning. In [9], the authors introduce the sparse multi-factor NMF, that can be seen as modelling the data as in eq.(3), with all \mathbf{S}_j s being non-negative matrices. In [10] and [11], the authors assume that the data come from a deep neural network, assuming that consecutive layers are sparsely connected and neglecting the non-linearities, they provide some strategies to recover the structure of the network. This model can be seen as modelling the data like in eq.(3) with the M leftmost factors representing a layer of the network each (the non-linear part being omitted), and the $M + 1$ th factor being the input of the network.

Note that another concern, 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 problematic have been proposed in [12] and [13].

3 Optimization framework

In this section we explicit the considered optimization problem, and describe an algorithm that is guaranteed to converge to a stationary point of the objective function.

3.1 Objective function

To learn a dictionary that is well adapted to the data while being fast to manipulate and cheap to store, we will minimize an objective function of the form of eq.(4). We will take as distance measure the squared Frobenius norm of the difference $d(\mathbf{X}, \prod_{j=1}^{M+1} \mathbf{S}_j) := \frac{1}{2} \|\mathbf{X} - \prod_{j=1}^{M+1} \mathbf{S}_j\|_F^2$ and as sparsity-seeking penalties some indicator functions of sets of sparse matrices: $g_j := \delta_{\mathcal{E}_j}$, with $\delta_{\mathcal{T}}(\mathbf{S}) = 0$ if $\mathbf{S} \in \mathcal{T}$ and $\delta_{\mathcal{T}}(\mathbf{S}) = \infty$ otherwise. The keen reader might have noticed that this basic formulation of the objective is invariant under relative scalings of the factors if the constraint sets are scale invariant themselves, and we address this issue below.

Choice of the constraints. The choice of the constraint sets is crucial, because they entirely determine the storage and multiplication cost of the learned dictionary. Indeed, storing/multiplying the dictionary in the factorized form will cost $\mathcal{O}(\sum_{j=1}^M \|\text{vec}(\mathbf{S}_j)\|_0^0)$, 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) of the learned dictionary:

$$\text{RC} := \frac{\sum_{j=1}^M \|\text{vec}(\mathbf{S}_j)\|_0^0}{da}. \quad (5)$$

This quantity is clearly positive and should be smaller than 1 in order to make complexity savings. In practice, we will usually choose \mathcal{E}_i s that are subsets of " ℓ_0 balls", namely they will take the form: $\mathcal{E}_j = \mathcal{N}_j \cap \{\mathbf{A} \in \mathbb{R}^{a_j \times a_{j+1}} : \|\text{vec}(\mathbf{A})\|_0^0 \leq p_j\}$, where \mathcal{N}_j is an arbitrary set imposing additional constraints. These constraints will give us: $\text{RC} \leq \sum_{j=1}^M p_j / da$.

Coping with the scaling ambiguity. In order to avoid scaling ambiguities, it is common [5, 9] to normalize the factors and introduce a multiplicative scalar λ in the data fidelity term. Doing so, the actual problem that we consider is the following:

$$\underset{\lambda, \mathbf{S}_1, \dots, \mathbf{S}_{M+1}}{\text{Minimize}} \quad \Psi(\mathbf{S}_1, \dots, \mathbf{S}_{M+1}, \lambda) := \frac{1}{2} \left\| \mathbf{X} - \lambda \prod_{j=1}^{M+1} \mathbf{S}_j \right\|_F^2 + \sum_{j=1}^{M+1} \delta_{\mathcal{E}_j}(\mathbf{S}_j), \quad (6)$$

with a new canonical form for the \mathcal{E}_j s namely $\mathcal{E}_j = \mathcal{N}_j \cap \{\mathbf{A} \in \mathbb{R}^{a_j \times a_{j+1}} : \|\text{vec}(\mathbf{A})\|_0^0 \leq p_j, \|\mathbf{A}\|_F = 1\}$, so that the factors are normalized.

3.2 Algorithm overview

The formulation of the problem in eq.(6) is unfortunately highly non-convex, and the sparsity enforcing part is non-smooth. Stemming on recent advances in non-convex optimization, we propose next an algorithm with convergence guarantees to a stationary point of the problem. In [6], the authors consider cost functions depending on N blocks of variables of the form:

$$\Psi(\mathbf{x}_1, \dots, \mathbf{x}_N) := H(\mathbf{x}_1, \dots, \mathbf{x}_N) + \sum_{j=1}^N f_j(\mathbf{x}_j), \quad (7)$$

where the function H is smooth, and the f_j s are proper and lower semi-continuous (the exact assumptions are given below). It is to be stressed that *no convexity* of any kind is assumed. Here, we assume for simplicity that the f_j s are indicator functions of constraint sets \mathcal{T}_j . To handle this objective function, the authors propose an algorithm called Proximal Alternating Linearized Minimization (PALM)[6], that updates alternatively each block of variable by a proximal (or projected in our case) gradient step. The structure of the PALM algorithm is given in Algorithm 1, where $P_{\mathcal{T}_j}(\cdot)$ is the projection operator onto the set \mathcal{T}_j and c_j^i defines the step size and depends on the Lipschitz constant of the gradient of H (we give its expression in the next subsection). The following conditions are sufficient (not necessary) to ensure that each bounded sequence generated by PALM converges to a stationary point of its objective:

- (i) The f_j s are proper and lower semi-continuous.
- (ii) H is smooth.
- (iii) Ψ is semi-algebraic.
- (iv) $\nabla_{\mathbf{x}_j} H$ is globally Lipschitz for all j , with Lipschitz moduli $L_j(\mathbf{x}_1 \dots \mathbf{x}_{j-1}, \mathbf{x}_{j+1} \dots \mathbf{x}_N)$.
- (v) $\forall i, c_j^i > L_j(\mathbf{x}_1^{i+1} \dots \mathbf{x}_{j-1}^{i+1}, \mathbf{x}_{j+1}^i \dots \mathbf{x}_N^i)$ (the inequality need not be strict for convex f_j).

Algorithm 1 PALM (summary)

```

for  $i \in \{1 \dots N_{iter}\}$  do
  for  $j \in \{1 \dots N\}$  do
    Set  $\mathbf{x}_j^{i+1} = P_{\mathcal{T}_j} \left( \mathbf{x}_j^i - \frac{1}{c_j^i} \nabla_{\mathbf{x}_j} H(\mathbf{x}_1^{i+1} \dots \mathbf{x}_j^i \dots \mathbf{x}_N^i) \right)$ 
  end for
end for

```

3.3 Algorithm details

Let us now instantiate PALM for our purpose, namely to handle the objective of eq.(6). It is quite straightforward to see that there is a match between eq.(6) and eq.(7) taking $N = M + 2$, $\mathbf{x}_j = \mathbf{S}_j$ for $j \in \{1 \dots M + 1\}$, $\mathbf{x}_{M+2} = \lambda$, H is the data fidelity term, $f_j(\cdot) = \delta_{\mathcal{E}_j}(\cdot)$ for $j \in \{1 \dots M + 1\}$ and $f_{M+2}(\cdot) = \delta_{\mathcal{E}_{M+2}}(\cdot) = \delta_{\mathbb{R}}(\cdot) = 0$ (there is no constraint on λ). With this particular instance of the problem, conditions (i), (ii) and (iii) are trivially fulfilled provided the \mathcal{E}_j s are semi-algebraic sets, which is indeed the case for all the sets considered in this work.

Projection operator. In the case where the \mathcal{E}_j s are defined like in Section 3.1 with no additional constraints, namely $\mathcal{E}_j = \{\mathbf{A} \in \mathbb{R}^{a_j \times a_{j+1}} : \|\text{vec}(\mathbf{A})\|_0^0 \leq p_j, \|\mathbf{A}\|_F = 1\}$ for $j \in \{1 \dots M + 1\}$, then 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 normalize its argument so that it has unit norm (see proof in appendix). Regarding $\mathcal{E}_{M+2} = \mathbb{R}$, the projection operator is the identity mapping.

Gradient and Lipschitz moduli. Let us now analyse more precisely the iterations of PALM specialized to our problem. For that we fix the iteration i and the factor j . We also need to introduce new notations. First we will call $\mathbf{S}^i := \mathbf{S}_j^i$ the factor that we are updating, $\mathbf{L} := \prod_{k=1}^{j-1} \mathbf{S}_k^{i+1}$ what is on the left of the factor we are updating and $\mathbf{R} := \prod_{k=j+1}^{M+1} \mathbf{S}_k^i$ what is on the right (with the convention $\prod_{k \in \emptyset} \mathbf{S}_k = \mathbf{Id}$). Moreover, and to simplify the notation when we update λ , let us introduce $\hat{\mathbf{X}} = \prod_{k=1}^{M+1} \mathbf{S}_k^{i+1}$. With these new notations we have when updating the j th factor: $H(\mathbf{S}_1^{i+1} \dots \mathbf{S}_j^i \dots \mathbf{S}_{M+1}^i, \lambda^i) = \frac{1}{2} \|\mathbf{X} - \lambda^i \mathbf{L} \mathbf{S}_j^i \mathbf{R}\|_F^2$. Or equivalently when updating λ : $H(\mathbf{S}_1^{i+1} \dots \mathbf{S}_{M+1}^{i+1}, \lambda^i) = \frac{1}{2} \|\mathbf{X} - \lambda^i \hat{\mathbf{X}}\|_F^2$.

The gradient of this smooth part of the objective with respect to the j th factor reads:

$$\nabla_{\mathbf{S}_j^i} H(\mathbf{S}_1^{i+1} \dots \mathbf{S}_j^i \dots \mathbf{S}_{M+1}^i, \lambda^i) = \lambda^i \mathbf{L}^T (\lambda^i \mathbf{L} \mathbf{S}_j^i \mathbf{R} - \mathbf{X}) \mathbf{R}^T,$$

which allows us to verify condition (iv) with $L_j(\mathbf{L}, \mathbf{R}, \lambda^i) = (\lambda^i)^2 \|\mathbf{R}\|_2^2 \cdot \|\mathbf{L}\|_2^2$ (see proof in appendix). Fixing a step size c_j^i so as to verify the condition (v), The update of \mathbf{S}_j^i can be rewritten:

$$\mathbf{S}_j^{i+1} = P_{\mathcal{E}_j} \left(\mathbf{S}_j^i - \frac{1}{c_j^i} \lambda^i \mathbf{L}^T (\lambda^i \mathbf{L} \mathbf{S}_j^i \mathbf{R} - \mathbf{X}) \mathbf{R}^T \right).$$

Now looking at λ we have:

$$\nabla_{\lambda^i} H(\mathbf{S}_1^{i+1} \dots \mathbf{S}_{M+1}^{i+1}, \lambda^i) = \lambda^i \text{Tr}(\hat{\mathbf{X}}^T \hat{\mathbf{X}}) - \text{Tr}(\mathbf{X}^T \hat{\mathbf{X}}).$$

Since $f_{M+2}(\lambda) = 0$ is a convex penalty, it is enough to check (v) with a non-strict inequality [6], this leads to the update rule:

$$\lambda^{i+1} = \frac{\text{Tr}(\mathbf{X}^T \hat{\mathbf{X}})}{\text{Tr}(\hat{\mathbf{X}}^T \hat{\mathbf{X}})}$$

An explicit version of the algorithm is given in Algorithm 2. Note that for simplicity we introduce a new notation for the total number of factors $Q := M + 1$.

Algorithm 2 PALM for learning efficient dictionaries (`pal4LED`)

Input: The data matrix \mathbf{X} , the desired number of factors Q , the constraint sets \mathcal{E}_j , $j \in \{1 \dots Q\}$ and a stopping criterion (e.g., here, a number of iterations N_{iter}).

```

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

Output: The estimated factorization: $\lambda^{N_{iter}}, \{\mathbf{S}_k^{N_{iter}}\}_{k=1}^Q = \text{pal4LED}(\mathbf{X}, Q, \{\mathcal{E}_j\}_{j=1}^Q)$

3.4 Practical strategy

Algorithm 2 presented above factorizes a data matrix into sparse factors and converges to a stationary point of the problem stated in eq.(6). However, while we are primarily interested in stationary points where the data fidelity term of the cost function is small, there is unfortunately no guarantee that the algorithm converges to such a stationary point. This is illustrated by a very simple experiment where Algorithm 2 is applied to a data matrix $\mathbf{X} = \mathbf{D}$ with a known factorization in M factors: $\mathbf{D} = \prod_{j=1}^M \mathbf{S}_j$, such as the Hadamard dictionary. The naive approach consists in taking directly $Q = M$ in Algorithm 2, and setting the constraints so as to reflect the actual sparsity of the true

factors. This simple strategy performs quite poorly in practice, and the attained local minimum is very often not satisfactory (the data fidelity part of the objective function is big).

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 $\mathbf{X} = \prod_{j=1}^{M+1} \mathbf{S}_j$ is the product of $M+1$ sparse factors, it is also the product $\mathbf{X} = \mathbf{T}_1 \mathbf{T}_2$ of 2 factors $\mathbf{T}_1 = \mathbf{S}_1$ and $\mathbf{T}_2 = \prod_{j=2}^{M+1} \mathbf{S}_j$, so that \mathbf{T}_1 is sparser than \mathbf{T}_2 . Our strategy is then to factorize the data matrix \mathbf{X} in 2 factors, one being sparse (corresponding to \mathbf{T}_1), and the other less sparse (corresponding to \mathbf{T}_2). The process can be repeated on the less sparse factor, and so on until we attain the desired number Q of factors. This strategy turns out to be surprisingly effective and the attained local minima are very good, as illustrated in the next section.

The proposed hierarchical strategy is summarized in Algorithm 3, 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 left factor and $\tilde{\mathcal{E}}_k$ the one for the right factor at the k th factorization. The global optimization step (line 5) is done by initializing `pal4mLED` with the current values of $\{\mathbf{S}_j\}_{j=1}^k$ and \mathbf{R} . It is here to keep an attach to the data matrix \mathbf{X} . Roughly we can say that line 3 of the algorithm is here to yield complexity savings, whereas line 5 is here to keep low the data fidelity term of the cost function.

Note: the hierarchical strategy can also be applied the other way around (starting *from the right*), just by transposing the input. We only present here the version that starts *from the left* because the induced notations are simpler.

Algorithm 3 Hierarchical factorization

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

1: $\mathbf{R} \leftarrow \mathbf{X}$

2: **for** $k = 1$ to $Q - 1$ **do**

3: Factorize the residual \mathbf{R} into 2 factors: $\lambda', \{\mathbf{T}_1, \mathbf{T}_2\} = \text{pal4mLED}(\mathbf{R}, 2, \{\mathcal{E}_k, \tilde{\mathcal{E}}_k\})$

4: $\mathbf{S}_k \leftarrow \lambda' \mathbf{T}_1$ and $\mathbf{R} \leftarrow \mathbf{T}_2$

5: Global optimization: $\lambda, \{\{\mathbf{S}_j\}_{j=1}^k, \mathbf{R}\} = \text{pal4mLED}(\mathbf{X}, k+1, \{\{\mathcal{E}_j\}_{j=1}^k, \tilde{\mathcal{E}}_k\})$

6: **end for**

7: $\mathbf{S}_Q \leftarrow \mathbf{R}$

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

4 Experiments

In all experiments, we consider square dictionaries and square factors.

4.1 Learning a fast implementation of the Hadamard transform

We begin by a dictionary factorization experiment. Consider a data matrix $\mathbf{X} = \mathbf{D}$ with a known factorization in M factors, $\mathbf{D} = \prod_{j=1}^M \mathbf{S}_j$: in Section 3.4, we evoked the failure of Algorithm 2 for this factorization problem. In contrast, Figure 2 illustrates the result of the proposed hierarchical strategy (Algorithm 3) with \mathbf{D} the Hadamard dictionary in dimension $n = 32$. The obtained factorization is exact and *as good as the reference one* shown on Figure 1 in terms of complexity savings. The running time is less than a second. Factorization of the Hadamard matrix in dimension up to $n = 1024$ showed identical performance, with running time $O(n^2)$ up to ten minutes.

4.2 Learning computationally efficient dictionaries

We now test Algorithm 3 in a more realistic framework on a dictionary learning problem with synthetic data, and compare it to classical *learned dictionaries* and *analytic dictionaries*, in terms of approximation quality and relative complexity.

Data. To build the data matrix \mathbf{X} , we generated 500 training samples by selecting uniformly at random 5 atoms in a dictionary $\mathbf{D}_0 \in \mathbb{R}^{32 \times 32}$ with i.i.d. Gaussian coefficients to build each sample. Gathering the coefficients in the matrix $\mathbf{\Gamma}_0 \in \mathbb{R}^{32 \times 500}$ we get our training data matrix $\mathbf{X} = \mathbf{D}_0 \mathbf{\Gamma}_0 \in \mathbb{R}^{32 \times 500}$. Two reference dictionaries \mathbf{D}_0 are considered:

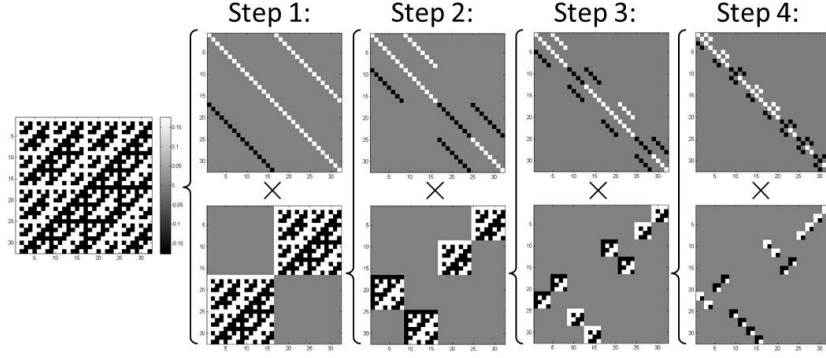


Figure 2: Hierarchical factorization of the Hadamard matrix of size 32×32 . The matrix is iteratively factorized in 2 factors, until we have $Q = 5$ factors, each having $p = 64$ non-zero entries.

- *Factorizable dictionary* (FACT): \mathbf{D}_0 is the product of $M = 5$ sparse matrices: $\mathbf{D}_0 = \prod_{i=1}^5 \mathbf{S}_i^0$, each having a random number p_i^0 of i.i.d. Gaussian non-zero entries with $64 \leq p_i^0 \leq 128$, and being full rank so that the dictionary spans the signal space.
- *Random dictionary* (RAND): \mathbf{D}_0 has i.i.d. Gaussian entries.

Baselines. We compare Algorithm 3 with the following methods. All methods involve a coefficient update step which is performed using Orthogonal Matching Pursuit (OMP) [14]:

- K-SVD [15], one of the most used algorithm that provides a *learned dictionary*. We use the implementation described in [16], running 300 iterations (which proved empirically sufficient to ensure convergence). Note that we also tested the online dictionary learning (ODL) method of [17]. Its performance being almost identical to that of K-SVD in our setting, we decided to consider these two methods as one (K-SVD/ODL) in the interpretation of the results.
- Sparse K-SVD [4], a method that seeks to bridge the gap between *learned dictionaries* and *analytic dictionaries*. The implementation of [4] is used, with \mathbf{D}_{base} the Discrete Cosine Transform (DCT) matrix and 100 iterations, ensuring convergence in practice. The estimated dictionary \mathbf{D} has columns 4-sparse in \mathbf{D}_{base} .
- A fixed *analytic dictionary* with a known fast implementation (either the DCT, the Haar wavelets (HAAR) or the Hadamard matrix (HAD)).

Settings of our algorithm. We tested several configurations for Algorithm 3, and we present here only the best one (PROPOSED). It amounts to Algorithm 3 starting from the right, with two modifications. First, we performed the first factorization ($k = 1$) by K-SVD/ODL to compute $\mathbf{S}_{M+1} = \mathbf{\Gamma}$. Second, we noticed that it was beneficial to update the coefficient matrix $\mathbf{\Gamma}$ with OMP after each global optimization step of Algorithm 3 (line 5). We tested various numbers of factors $Q \in \{3 \dots 6\}$. The considered constraint sets were: $\mathcal{E}_1 = \{\mathbf{A} \in \mathbb{R}^{32 \times 500}, \|\mathbf{a}_n\|_0^0 \leq 5\}$ and for $k \in \{2 \dots Q - 1\}$, $\mathcal{E}_k = \{\mathbf{A} \in \mathbb{R}^{32 \times 32}, \|\text{vec}(\mathbf{A})\|_0^0 \leq p, \|\mathbf{A}\|_F = 1\}$ and $\tilde{\mathcal{E}}_k = \{\mathbf{A} \in \mathbb{R}^{32 \times 32}, \|\text{vec}(\mathbf{A})\|_0^0 \leq \frac{P}{2^{k-2}}, \|\mathbf{A}\|_F = 1\}$. We show the results for sparsity constraints given by $p \in \{2, 3, 4\}$ and $P \in 512 \times \{1, 1.2, 1.4, 1.6\}$. Algorithm 3 was implemented in Matlab, and executed on Intel Core i7-3667U CPU. It typically took between 8 and 9 seconds to converge for each drawn \mathbf{X} . The stopping criterion for palm4LED combined a maximum number of iterations $N_{\text{tier}} = 500$ and a bound $\epsilon = 10^{-6}$ on the variation of the approximation error between consecutive iterations.

Performance measures. The ideal dictionary should approximate well the data at hand, while being at the same time fast to manipulate and cheap to store. The computational efficiency of the dictionary is measured through the *Relative complexity* (RC) quantity introduced in Section 3.1. The quality of approximation is expressed using the Root-Mean-Square Error (RMSE)[4, 15]: $\text{RMSE} := \frac{1}{\sqrt{dn}} \|\mathbf{X} - \mathbf{D}\mathbf{\Gamma}\|_F$.

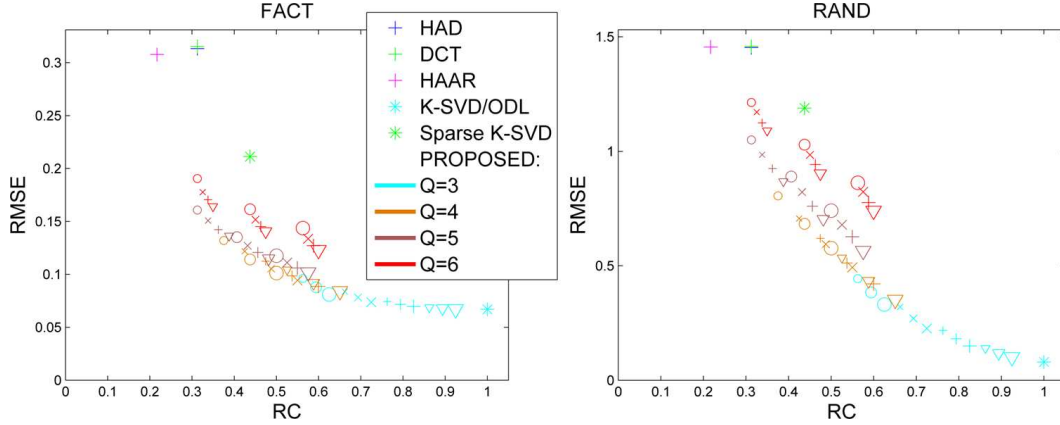


Figure 3: (better seen in colors) Comparison of different dictionary learning methods, with data generated using different dictionaries: factorizable (left), random (right). For the proposed method, the symbol shape indicates the value of P (\circ : $P = 512$, \times : $P = 1.2 \times 512$, $+$: $P = 1.4 \times 512$ and ∇ : $P = 1.6 \times 512$), its color the value of Q (see legend), and its size the value of p (Small: $p = 2$, Medium: $p = 3$, Big: $p = 4$).

Discussion of the results. The experiment has been repeated 100 times with each data generation method. For a given configuration of the algorithm the relative complexity is constant over all trials, and the results shown on Figure 3 display the average RMSE.

With a *factorizable dictionary* (left), as expected, the methods that use a fast dictionary (HAD, DCT and HAAR) perform quite poorly in approximation (vertical axis), but very good in relative complexity (horizontal axis) taking advantage of their intrinsic structure. On the other hand, K-SVD exhibits good approximation performance, while the lack of structure of the obtained dictionary does not lead to any complexity savings ($RC = 1$). In between these two extremes, Sparse K-SVD, thanks to its layer of adaptivity, performs better than the analytic dictionaries in approximation at the expense of a slightly higher relative complexity. The proposed method (PROPOSED) has the ability to achieve a flexible tradeoff between complexity and adaptation to the data. More specifically, we can identify several behaviors for the proposed method. With $Q = 3$ factors and $P = 1.4 \times 512$ or $P = 1.6 \times 512$, the proposed method performs almost as good as K-SVD in terms of approximation, with reduced relative complexities between 0.7 and 0.9. On the other hand, with $p = 2$ and $Q = 5$ or $Q = 6$, the proposed method provides dictionaries almost as compact as analytic dictionaries ($RC \approx 0.3$), while being better adapted to the data (RMSE up to twice smaller). The other configurations of p , P and Q all lie between these two behaviors in terms of performance.

With a *random dictionary* (right), the methods exhibit qualitatively the same comparative behavior as with a factorizable dictionary. Notably, the proposed method can learn a dictionary as computationally efficient as the one provided by Sparse K-SVD but with half the approximation error.

5 Conclusion

We proposed a dictionary learning framework that provides a flexible tradeoff between computational efficiency and adaptation to the training data. Stemming on recent advances in non-convex optimization, we derived an algorithm with convergence guarantees to learn efficient dictionaries and demonstrated experimentally its ability to provide complexity/accuracy tradeoffs that state of the art dictionary learning methods could not achieve. Besides the obvious need to further test the approach on real data and with redundant dictionaries, and 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 learning process itself, in order to *efficiently learn* efficient dictionaries.

Acknowledgments

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.

References

- [1] Ron Rubinstein, A.M. Bruckstein, and Michael Elad. Dictionaries for Sparse Representation Modeling. *Proceedings of the IEEE*, 98(6):1045–1057, 2010.
- [2] James Cooley and John Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.
- [3] Stéphane Mallat. A theory for multiresolution signal decomposition : the wavelet representation. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 11:674–693, June 1989.
- [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 Tournieret, and Nicolas Dobigeon. Toward fast transform learning. Technical report, November 2013.
- [6] Jérôme Bolte, Shoham Sabach, and Marc Teboulle. Proximal alternating linearized minimization for nonconvex and nonsmooth problems. *Mathematical Programming*, pages 1–36, 2013.
- [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] S.G. Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415, December 1993.
- [15] 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.
- [16] Ron Rubinstein, Michael Zibulevsky, and Michael Elad. Efficient Implementation of the K-SVD Algorithm using Batch Orthogonal Matching Pursuit. Technical report, 2008.
- [17] 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.