Learning Fast Sparsifying Transforms

Learning Fast Sparsifying Transforms

Cristian Rusu and John Thompson

***Abstract*—Given a dataset, the task of learning a transform that allows sparse representations of the data bears the name of dictionary learning. In many applications, these learned dictionaries represent the data much better than the static well-known transforms (Fourier, Hadamard etc.). The main downside of learned transforms is that they lack structure and therefore they are not computationally efficient, unlike their classical counterparts. These posse several difficulties especially when using power limited hardware such as mobile devices, therefore discouraging the application of sparsity techniques in such scenarios. In this paper we construct orthogonal and non-orthogonal dictionaries that are factorized as a product of a few basic transformations. In the orthogonal case, we solve exactly the dictionary update problem for one basic transformation, which can be viewed as a generalized Givens rotation, and then propose to construct orthogonal dictionaries that are a product of these transformations, guaranteeing their fast manipulation. We also propose a method to construct fast square but non-orthogonal dictionaries that are factorized as a product of few transforms that can be viewed as a further generalization of Givens rotations to the non-orthogonal setting. We show how the proposed transforms can balance very well data representation performance and computational complexity. We also compare with classical fast and learned general and orthogonal transforms.**

I. Introduction

Dictionary learning methods [1] represent a well-known class of algorithms that have seen many applications in signal processing [2], image processing [3], wireless communications [4] and machine learning [5]. The key idea of this approach is not to use an off-the-shelf transform like the Fourier, Hadamard or wavelet but to learn a new transform, often called an overcomplete dictionary, for a particular task (like coding and classification) from the data itself. While the dictionary learning problem is NP-hard [6] in general, it has been extensively studied and several good algorithms to tackle it exist. Alternating minimization methods like the method of optimal directions (MOD) [7], K–SVD [8], [9] and direct optimization [10] have been shown to work well in practice and also enjoy some theoretical performance guarantees. While learning a dictionary we need to construct two objects: the dictionary and the representation of the data in the dictionary.

One problem that arises in general when using learned dictionaries is the fact that they lack any structure. This is to be compared with the previously mentioned off-the-shelf transforms that have a rich structure. This is reflected in their low computational complexity, i.e., they can be applied directly using $O(n \log n)$ computations for example [11]. Our goal in this paper is to provide a solution to the problem of constructing fast transforms, based upon the structure of Givens rotations, learned from training data.

We first choose to study orthogonal structures since sparse reconstruction is computationally cheaper in such a dictionary: we project the data onto the column space of the dictionary and keep the largest s coefficients in magnitude to obtain the provable best s-term approximation. Working in an n dimensional feature space, this operation has complexity $O(n^2)$. In a general non-orthogonal (and even overcomplete) dictionary, special non-linear reconstruction methods such as ℓ_1 minimization [12], greedy approaches like orthogonal matching pursuit (OMP) [13] or variational Bayesian algorithms like approximate message passing (AMP) [14] need to be applied. Aside from the fact that in general these methods cannot guarantee to produce best s-term approximations they are also computationally expensive. For example, the classical OMP has complexity $O(sn^2)$ [15] and, assuming that we are looking for sparse approximations with $s \ll n$, it is in general computationally cheaper than ℓ_1 optimization. Therefore, considering a square orthogonal dictionary is a first step in the direction of constructing a fast transform. For the analysis dictionary, recent work based on transform learning [16], [17] has been proposed. Still, notice that computing sparse representations in such a dictionary has complexity $O(n^2)$ and therefore, our goal of constructing a fast transform cannot be reached with just a general orthogonal dictionary. We make the case that our fundamental goal is to actually build a structured orthogonal dictionary such that matrix-vector multiplications with this dictionary can be achieved with less than $O(n^2)$ operations, preferably $O(n \log n)$. This connects our paper to previous work on approximating orthogonal (and symmetric) matrices [18] such that matrix-vector multiplications are computationally efficient.

When we talk about "learning fast sparsifying transforms" we do not refer to the efficient learning procedures (although the proposed learning methods have polynomial complexity) but we refer to the transforms themselves, i.e., once we have the transform, the computational complexity of using it is low, preferably $O(n \log n)$ to perform matrix-vector multiplication.

Previous work [19], [20], [21], [22], [23], [24], [25] in the literature has already proposed various structured dictionaries to cope with the high computational complexity of learned transforms. Previous work also dealt with the construction of structured orthogonal dictionaries. Specifically, [26] proposed to build an orthogonal dictionary composed of a product of a few Householder reflectors. In this fashion, the computational complexity of the dictionary is controlled and a trade-off between representation performance and computational complexity is shown.

The authors are with the Institute for Digital Communications, School of Engineering, The University of Edinburgh, Scotland. Email: {c.rusu, john.thompson}@ed.ac.uk. Demo source code available online at https://udrc.eng.ed.ac.uk/sites/udrc.eng.ed.ac.uk/files/attachments/demo.zip.

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) Grant number EP/K014277/1 and the MOD University Defence Research Collaboration (UDRC) in Signal Processing.

Learned dictionaries with low computational complexity can bridge the gap between the classical transforms that are preferred especially in power limited hardware (or battery operated devices) and the overcomplete, computationally cumbersome, learned dictionaries that provide state-of-the-art performance in many machine learning tasks. The contribution of this paper is two fold.

First, we consider the problem of constructing an orthogonal dictionary as a product of a given number of generalized Givens rotations. We start by showing the optimum solution to the dictionary learning problem when the dictionary is a single generalized Givens rotation and then move to expand on this result and propose an algorithm that sequentially builds a product of generalized Givens rotations to act as a dictionary for sparse representations. Each step of the algorithm solves exactly the proposed optimization problem and therefore we can guarantee that it monotonically converges to a local minimum. We show numerically that the fast dictionaries proposed in this paper outperform those based on Householder reflectors [26] in terms of representation error, for the same computational complexity.

Second, based on a structure similar to the generalized Givens rotation we then propose a learning method that constructs square, non-orthogonal, computationally efficient dictionaries. In order to construct the dictionary we again solve exactly a series of optimization problems. Unfortunately we cannot prove the monotonic convergence of the algorithm since the sparse reconstruction step, based in this paper on OMP, cannot guarantee in general a monotonic reduction in our objective function. Still, we are able to show that these fast non-orthogonal transforms perform very well, better than their orthogonal counterparts.

In the results section we compare the proposed methods among each other and to previously proposed dictionary learning methods in the literature. We show that the methods proposed in this paper provide a clear trade-off between representation performance and computational complexity. Interestingly, we are able to provide numerical examples where the proposed fast orthogonal dictionaries have higher computational efficiency and provide better representation performance than the well-known discrete cosine transform (DCT), the transform at the heart of the jpeg compression standard [27].

II. A brief description of dictionary learning optimization problems

Given a real dataset $\mathbf{Y} \in \mathbb{R}^{n\times N}$ and sparsity level s, the general dictionary learning problem is to produce the factorization $\mathbf{Y} \approx \mathbf{DX}$ given by the optimization problem:

$$\begin{aligned} \underset{\mathbf{D},\ \mathbf{X}}{\text{minimize}} \quad & \|\mathbf{Y}-\mathbf{DX}\|_F^2 \\ \text{subject to} \quad & \|\mathbf{x}_i\|_0 \le s,\ 1\le i\le N \\ & \|\mathbf{d}_j\|_2 = 1,\ 1\le j\le n, \end{aligned} \tag{1}$$

where the objective function describes the Frobenius norm representation error achieved by the square dictionary $\mathbf{D}\in\mathbb{R}^{n\times n}$ with the sparse representations $\mathbf{X}\in\mathbb{R}^{n\times N}$ whose columns are subject to the ℓ_0 pseudo-norm $\|\mathbf{x}_i\|_0$ (the number of non-zero elements of columns $\mathbf{x}_i$). To avoid trivial solutions, the dimensions obey $s \ll n \ll N$. Several algorithms that work very well in practice exist [7] [8] [15] to solve this factorization problem. Their approach, and the one we also adopt in this paper, is to keep the dictionary fixed and update the representations and then reverse the roles by updating the dictionary with the representations fixed. This alternating minimization approach proves to work very well experimentally [7], [8] and allows some theoretical insights [28].

In this paper we also consider the dictionary learning problem (1) with an orthogonal dictionary $\mathbf{Q}\in\mathbb{R}^{n\times n}$ [29] [30] [31] [32]. The orthogonal dictionary learning problem (which we call in this paper Q–DLA) [33] is formulated as:

$$\begin{aligned} \underset{\mathbf{Q},\ \mathbf{X};\ \mathbf{QQ}^T=\mathbf{Q}^T\mathbf{Q}=\mathbf{I}}{\text{minimize}} \quad & \|\mathbf{Y}-\mathbf{QX}\|_F^2 \\ \text{subject to} \quad & \|\mathbf{x}_i\|_0 \le s,\ 1\le i\le N. \end{aligned} \tag{2}$$

Since the dictionary $\mathbf{Q}$ is orthogonal, the construction of $\mathbf{X}$ no longer involves ℓ_1 [12], OMP [13] or AMP [14] approaches as in (1), but reduces to $\mathbf{X} = \mathcal{T}_s(\mathbf{Q}^T\mathbf{Y})$, where $\mathcal{T}_s()$ is an operator that given an input vector zeros all entries except the largest s in magnitude and given an input matrix applies the same operation on each column in turn. To solve (2) for variable $\mathbf{Q}$ and fixed $\mathbf{X}$, a problem also known as the orthogonal Procrustes problem [34], a closed form solution $\mathbf{Q}=\mathbf{UV}^T$ is given by the singular value decomposition of $\mathbf{YX}^T=\mathbf{U\Sigma V}^T$.

III. A building block for fast transforms

For indices $(i,j),\ j>i$ and variables $p,q,r,t\in\mathbb{R}$ let us define the basic transform, which we call an R-transform:

$$\mathbf{R}_{ij} = \begin{bmatrix} \mathbf{I}_{i-1} & & & & \\ & p & & r & \\ & & \mathbf{I}_{j-i-1} & & \\ & q & & t & \\ & & & & \mathbf{I}_{n-j} \end{bmatrix} \in \mathbb{R}^{n\times n}, \tag{3}$$

where we have denoted $\mathbf{I}_i$ as the identity matrix of size i. For simplicity, we denote the non-zero part of $\mathbf{R}_{ij}$ as

$$\tilde{\mathbf{R}}_{ij} = \begin{bmatrix} p & r \\ q & t \end{bmatrix} \in \mathbb{R}^{2\times 2}. \tag{4}$$

A right side multiplication between a R-transform and a matrix $\mathbf{X}\in\mathbb{R}^{n\times N}$ operates only rows i and j as

$$\begin{aligned} \mathbf{R}_{ij}\mathbf{X} = [\mathbf{x}_1 \quad \dots \quad & p\mathbf{x}_i + r\mathbf{x}_j \quad \dots \\ & \dots \quad q\mathbf{x}_i + t\mathbf{x}_j \quad \dots \quad \mathbf{x}_n]^T, \end{aligned} \tag{5}$$

where $\mathbf{x}_i^T$ is the i^{th} row of $\mathbf{X}$. The number of operations needed for this task is only $6N$. Left and right multiplications with a R-transform (or its transpose) are therefore computationally efficient. We use this matrix structure as a basic building block for the transforms learned in this paper.

Remark 1. Every matrix $\mathbf{D}\in\mathbb{R}^{n\times n}$ can be written as a product of at most $\lceil n^2 - \frac{n}{2}\rceil$ R-transforms. Therefore, we can consider the R-transforms as fundamental building blocks for all square transforms $\mathbf{D}$.

Proof. Consider the singular value decomposition $\mathbf{D}=\mathbf{U\Sigma V}^T$. Each $\mathbf{U}$ and $\mathbf{V}$ can be factored as a product of $\binom{n}{2}$ Givens rotations [35] which are all in fact constrained

R-transforms (with $p = t = c$ and $r = -q = d$ for some given c and d such that $c^2 + d^2 = 1$) and a diagonal matrix containing only $\{\pm 1\}$ entries. While the diagonal $\mathbf{\Sigma}$ can be factored as a product of $\lceil \frac{n}{2} \rceil$ diagonal R-transforms. ■

In this paper we will be interested to use $\mathbf{R}_{ij}$ in least squares problems with the objective function as:

$$\|\mathbf{Y} - \mathbf{R}_{ij}\mathbf{X}\|_F^2 = \|\mathbf{Y}\|_F^2 + \|\mathbf{X}\|_F^2 - 2\text{tr}(\mathbf{Z}) - \left\| \begin{bmatrix} \mathbf{y}_i^T \\ \mathbf{y}_j^T \end{bmatrix} \right\|_F^2 - \left\| \begin{bmatrix} \mathbf{x}_i^T \\ \mathbf{x}_j^T \end{bmatrix} \right\|_F^2 + 2\text{tr}(\mathbf{Z}_{\{i,j\}}) + \left\| \begin{bmatrix} \mathbf{y}_i^T \\ \mathbf{y}_j^T \end{bmatrix} - \tilde{\mathbf{R}}_{ij} \begin{bmatrix} \mathbf{x}_i^T \\ \mathbf{x}_j^T \end{bmatrix} \right\|_F^2. \quad (6)$$

For simplicity of exposition we have defined

$$\mathbf{Z} = \mathbf{Y}\mathbf{X}^T, \mathbf{Z}_{\{i,j\}} = \begin{bmatrix} Z_{ii} & Z_{ij} \\ Z_{ji} & Z_{jj} \end{bmatrix} \in \mathbb{R}^{2\times 2}, Z_{ij} = \mathbf{y}_i^T \mathbf{x}_j, \quad (7)$$

where $\mathbf{y}_i^T$ and $\mathbf{x}_i^T$ are the i^{th} rows of $\mathbf{Y}$ and $\mathbf{X}$, respectively.

We now introduce learning methods to create computationally efficient orthogonal and non-orthogonal dictionaries.

IV. A method for designing fast orthogonal transforms: G_m–DLA

In this section we propose a method called G_m–DLA to learn orthogonal dictionaries that are factorized as a product of m G-transforms (constrained R-transforms).

A. An overview of G-transforms

We call $\mathbf{G}_{ij}$ a G-transform, an orthogonal constrained R-transform (3) parameterized only by $c, d \in \mathbb{R}$ with $c^2+d^2 = 1$, and the indices (i, j), $i \neq j$ such that the non-zero part of $\mathbf{G}_{ij}$, corresponding to (4), is given by

$$\tilde{\mathbf{G}}_{ij} \in \left\{ \begin{bmatrix} c & d \\ -d & c \end{bmatrix}, \begin{bmatrix} c & d \\ d & -c \end{bmatrix} \right\}. \quad (8)$$

Classically, a Givens rotation is a matrix as in (3) with $\tilde{\mathbf{G}}_{ij} = \begin{bmatrix} c & d \\ -d & c \end{bmatrix}$ such that $\det(\mathbf{G}_{ij}) = 1$, i.e., proper rotation matrices are orthogonal matrices with determinant one. These rotations are important since any orthogonal dictionary of size $n \times n$ can be factorized in a product of $\binom{n}{2}$ Givens rotations [35]. In this paper, since we are interested in the computational complexity of these structures, we allow both options in (8) that fully characterize all 2×2 real orthogonal matrices – these structures are discussed in [36, Chapter 2.1]. With $\tilde{\mathbf{G}}_{ij} = \begin{bmatrix} c & d \\ d & -c \end{bmatrix}$ the G-transform in (3) is in fact a Householder reflector $\mathbf{G}_{ij} = \mathbf{I} - 2\mathbf{g}_{ij}\mathbf{g}_{ij}^T$ where $\mathbf{g}_{ij} \in \mathbb{R}^n$, $\|\mathbf{g}_{ij}\|_2 = 1$, has all entries equal to zero except for the i^{th} and j^{th} entries that are $\sqrt{0.5(1-c)}$ and $-\text{sign}(d)\sqrt{0.5(1+c)}$, respectively – one might call this a "Givens reflector" to highlight its distinguishing sparse structure. Givens rotations have been previously used in matrix factorization applications [37], [38].

B. One G-transform as a dictionary

Consider now the dictionary learning problem in (2). Let us keep the sparse representations $\mathbf{X}$ fixed and consider a single G-transform as a dictionary. We reach the following

$$\underset{(i,j),\ \tilde{\mathbf{G}}_{ij}}{\text{minimize}} \ \|\mathbf{Y} - \mathbf{G}_{ij}\mathbf{X}\|_F^2. \quad (9)$$

When indices (i, j) are fixed, the problem reduces to constructing $\tilde{\mathbf{G}}_{ij}$, a constrained two dimensional optimization problem. To select the indices (i, j), among the $\binom{n}{2}$ possibilities, an appropriate strategy needs to be defined. We detail next how to deal with these two problems to provide an overall solution for (9).

To solve (9) **for the fixed coordinates** (i, j) we reach the optimization problem

$$\underset{\tilde{\mathbf{G}}_{ij};\ \tilde{\mathbf{G}}_{ij}^T\tilde{\mathbf{G}}_{ij}=\tilde{\mathbf{G}}_{ij}\tilde{\mathbf{G}}_{ij}^T=\mathbf{I}}{\text{minimize}} \left\| \begin{bmatrix} \mathbf{y}_i^T \\ \mathbf{y}_j^T \end{bmatrix} - \tilde{\mathbf{G}}_{ij} \begin{bmatrix} \mathbf{x}_i^T \\ \mathbf{x}_j^T \end{bmatrix} \right\|_F^2. \quad (10)$$

This is a two dimensional Procrustes problem [34] whose optimum solution is $\tilde{\mathbf{G}}_{ij} = \mathbf{U}\mathbf{V}^T$ where $\mathbf{Z}_{\{i,j\}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$. It has been shown in [26] that the reduction in the objective function of (10) when considering an orthogonal dictionary $\mathbf{G}_{ij}$ given by the Procrustes solution is

$$\begin{aligned} &\left\| \begin{bmatrix} \mathbf{y}_i^T \\ \mathbf{y}_j^T \end{bmatrix} - \tilde{\mathbf{G}}_{ij} \begin{bmatrix} \mathbf{x}_i^T \\ \mathbf{x}_j^T \end{bmatrix} \right\|_F^2 \\ &= \left\| \begin{bmatrix} \mathbf{y}_i^T \\ \mathbf{y}_j^T \end{bmatrix} \right\|_F^2 + \left\| \begin{bmatrix} \mathbf{x}_i^T \\ \mathbf{x}_j^T \end{bmatrix} \right\|_F^2 - 2\text{tr}\left(\mathbf{G}_{ij}^T \begin{bmatrix} \mathbf{y}_i^T \\ \mathbf{y}_j^T \end{bmatrix} \begin{bmatrix} \mathbf{x}_i^T \\ \mathbf{x}_j^T \end{bmatrix}^T \right) \quad (11) \\ &= \left\| \begin{bmatrix} \mathbf{y}_i^T \\ \mathbf{y}_j^T \end{bmatrix} \right\|_F^2 + \left\| \begin{bmatrix} \mathbf{x}_i^T \\ \mathbf{x}_j^T \end{bmatrix} \right\|_F^2 - 2\|\mathbf{Z}_{\{i,j\}}\|_*, \end{aligned}$$

where $\|\mathbf{Z}_{\{i,j\}}\|_*$ is the nuclear norm of $\mathbf{Z}_{\{i,j\}}$, i.e., the sum of its singular values.

Choosing (i, j) **in** (9) requires a closer look at its objective function (6) for $\tilde{\mathbf{R}}_{ij} = \tilde{\mathbf{G}}_{ij}$, the constrained G-transform structure. Using (11) we can state a result in the special case of a G-transform. We need both because for any indices (i, j) the reduction in the objective function invokes the nuclear norm, while for the other indices the reduction invokes the trace. We can analyze the two objective function values separately because the Frobenius norm is elementwise and as such also blockwise. Therefore, the objective of (9) is

$$\begin{aligned} &\|\mathbf{Y} - \mathbf{G}_{ij}\mathbf{X}\|_F^2 = \|\mathbf{Y}\|_F^2 + \|\mathbf{X}\|_F^2 - 2\text{tr}(\mathbf{Z}) - 2C_{ij}, \\ &\text{where } C_{ij} = \|\mathbf{Z}_{\{i,j\}}\|_* - \text{tr}(\mathbf{Z}_{\{i,j\}}). \end{aligned} \quad (12)$$

Since we want to minimize this quantity, the choice of indices needs to be made as follows

$$(i^\star, j^\star) = \underset{(i,j),\ j>i}{\arg\max}\ C_{ij}, \quad (13)$$

and then solve a Procrustes problem [34] to construct $\tilde{\mathbf{G}}_{i^\star j^\star}$.

These $(i^\star, j^\star)$ values are the optimum indices that lead to the maximal reduction in the objective function of (9). The expression in (13) is computationally cheap given that $\mathbf{Z}_{\{i,j\}}$ is a 2×2 real matrix. Its trace is trivial to compute $\text{tr}(\mathbf{Z}_{\{i,j\}}) = Z_{ii} + Z_{jj}$ (one addition operation) while the singular values of $\mathbf{Z}_{\{i,j\}}$ can be explicitly computed as

$$\sigma_{1,2} = \sqrt{\frac{1}{2}\left(\|\mathbf{Z}_{\{i,j\}}\|_F^2 \pm \sqrt{\|\mathbf{Z}_{\{i,j\}}\|_F^4 - 4\det(\mathbf{Z}_{\{i,j\}})^2} \right)}. \quad (14)$$

Therefore, the full singular value decomposition can be avoided and the sum of the singular values from (14) can be computed in only 23 operations (three of which are taking square roots). The cost of computing C_{ij} for all indices

(i,j), $j > i$, is $25\frac{n(n-1)}{2}$ operations. The computational burden is still dominated by constructing $\mathbf{Z} = \mathbf{Y}\mathbf{X}^T$ which takes $2snN$ operations.

Remark 2. Notice that $C_{ij} \geq 0$ always. In general, this is because the sum of the singular values of any matrix $\mathbf{Z}$ of size $n \times n$ is always greater than the sum of its eigenvalues. To see this, use the singular value decomposition of $\mathbf{Z} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T, \mathbf{\Sigma} = \text{diag}(\sigma)$, and develop:

$$\text{tr}(\mathbf{Z}) = \text{tr}(\mathbf{\Sigma}\mathbf{V}^T\mathbf{U}) = \sum_{k=1}^{n} \sigma_k \Delta_{kk} \leq \sum_{k=1}^{n} \sigma_k = \|\mathbf{Z}\|_*, \quad (15)$$

where we have use the circular property of the trace and $\mathbf{\Delta} = \mathbf{V}^T\mathbf{U}$ where Δ_{kk} are its diagonal entries which obey $|\Delta_{kk}| \leq 1$ since both $\mathbf{U}$ and $\mathbf{V}$ are orthogonal and their entries are sub-unitary in magnitude. Therefore, in our particular case, we have that $C_{ij} = 0$ when $\mathbf{Z}_{\{i,j\}}$ is symmetric and positive semidefinite (we have that $\mathbf{\Delta} = \mathbf{I}$ in (15) and therefore $\text{tr}(\mathbf{Z}_{\{i,j\}}) = \|\mathbf{Z}_{\{i,j\}}\|_*$). If we have that $C_{ij} = 0$ for all i and j then no G-transform can reduce the objective function in (9) and therefore the solution is $\mathbf{G}_{ij} = \mathbf{I}$. ■

Remark 3. We can extend the G-transform to multiple indices. For example, if we consider three coordinates then $\mathbf{G}_{ijk} \in \mathbb{R}^{n\times n}$ has the non-zero orthogonal block $\tilde{\mathbf{G}}_{ijk} \in \mathbb{R}^{3\times 3}$. For a transform over q indices there are $\binom{n}{q}$ such blocks and its matrix-vector multiplication takes $(2q-1)q$ operations. ■

Remark 4. There are some connections between the Householder [26] and the G-transform approaches. As previously explained, when $\tilde{\mathbf{G}}_{ij} = \begin{bmatrix} c & d \\ d & -c \end{bmatrix}$ the G-transform can also be viewed as a Householder reflector, i.e., $\mathbf{G}_{ij} = \mathbf{I} - 2\mathbf{g}_{ij}\mathbf{g}_{ij}^T$ where $\mathbf{g}_{ij} \in \mathbb{R}^n$ is a 2-sparse vector. Following results from [26] we can also write

$$\begin{aligned} \|\mathbf{Y} - \mathbf{G}_{ij}\mathbf{X}\|_F^2 =& \|\mathbf{Y}\|_F^2 + \|\mathbf{X}\|_F^2 - 2\text{tr}(\mathbf{Y}\mathbf{X}^T) \\ &+ 2\mathbf{g}_{ij}^T(\mathbf{Y}\mathbf{X}^T + \mathbf{X}\mathbf{Y}^T)\mathbf{g}_{ij}, \end{aligned} \quad (16)$$

which, together with (12), leads to $\mathbf{g}_{ij}^T(\mathbf{Y}\mathbf{X}^T + \mathbf{X}\mathbf{Y}^T)\mathbf{g}_{ij} = -C_{ij}$. This means that choosing to maximize C_{ij} in (12) is equivalent to computing an eigenvector of $\mathbf{Y}\mathbf{X}^T + \mathbf{X}\mathbf{Y}^T$ of sparsity two associated with a negative eigenvalue.

There are also some differences between the two approaches. For example, matrix-vector multiplication with a G-transform $\mathbf{G}_{ij}\mathbf{x}$ takes 6 operations but when using the Householder structure $\mathbf{G}_{ij}\mathbf{x} = (\mathbf{I} - 2\mathbf{g}_{ij}\mathbf{g}_{ij}^T)\mathbf{x} = \mathbf{x} - 2(\mathbf{g}_{ij}^T\mathbf{x})\mathbf{g}_{ij}$ takes 8 operations (4 operations to compute the constant $C = 2\mathbf{g}_{ij}^T\mathbf{x}$, 2 operations to compute the 2-sparse vector $\mathbf{z} = C\mathbf{g}_{ij}$ and 2 operations to compute the final result $\mathbf{x} - \mathbf{z}$). Therefore, the G-transform structure is computationally preferable to the Householder structure. Each Householder reflector has $n-1$ (because of the orthogonality constraint) degrees of freedom while each G-transform has only 1 (the angle $\theta \in [0, 2\pi]$ for which $c = \cos\theta$ and $d = \sin\theta$) plus 1 bit (the choice of the rotation or reflector in (8)). ■

This concludes our discussion for the single G-transform case. Notice that the solution outlined in this section solves (9) exactly, i.e., it finds the optimum G-transform.

C. A method for designing fast orthogonal transforms: G_m–DLA

In this paper we propose to construct an orthogonal transform $\mathbf{U} \in \mathbb{R}^{n\times n}$ with the following structure:

$$\mathbf{U} = \mathbf{G}_{i_m j_m} \dots \mathbf{G}_{i_2 j_2} \mathbf{G}_{i_1 j_1}. \quad (17)$$

The value of m is a choice of the user. For example, if we choose m to be $O(n \log n)$ the transform $\mathbf{U}$ can be computed in $O(n \log n)$ computational complexity – similar to the classical fast transforms. The goal of this section is to propose a learning method that constructs such a transform.

We fix the representations $\mathbf{X}$ and all G-transforms in (17) except for the k^{th}, denoted as $\mathbf{G}_{i_k j_k}$. To optimize the dictionary $\mathbf{U}$ only for this transform we reach the objective function

$$\begin{aligned} \|\mathbf{Y} - \mathbf{U}\mathbf{X}\|_F^2 &= \|\mathbf{Y} - \mathbf{G}_{i_m j_m} \dots \mathbf{G}_{i_1 j_1}\mathbf{X}\|_F^2 \\ &= \|\mathbf{G}_{i_{k+1} j_{k+1}}^T \dots \mathbf{G}_{i_m j_m}^T \mathbf{Y} - \mathbf{G}_{i_k j_k} \dots \mathbf{G}_{i_1 j_1}\mathbf{X}\|_F^2 \\ &= \|\mathbf{Y}_k - \mathbf{G}_{i_k j_k}\mathbf{X}_k\|_F^2, \end{aligned} \quad (18)$$

where we have used the fact that multiplication by any orthogonal transform preserves the Frobenius norm. For simplicity we have denoted $\mathbf{Y}_k$ and $\mathbf{X}_k$ the known quantities in (18) and therefore $\mathbf{Z}_k = \mathbf{Y}_k\mathbf{X}_k^T$.

Notice that we have reduced the problem to the formulation in (9) whose full solution is outlined in the previous section. We can apply this procedure for all G-transforms in the product of $\mathbf{U}$ and therefore a full update procedure presents itself: we will sequentially update each transform and then the sparse representations until convergence. The full procedure we propose, called G_m–DLA, is detailed in Algorithm 1.

The initialization of G_m–DLA uses a known construction. It has been shown experimentally in the past [39], that a good initial orthogonal dictionary is to choose $\mathbf{U}$ from the singular value decomposition of the dataset $\mathbf{Y} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$. We can also provide a theoretical argument for this choice. Consider that

$$\mathbf{X} = \mathcal{T}_s(\mathbf{U}^T\mathbf{Y}) = \mathcal{T}_s(\mathbf{U}^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T) = \mathcal{T}_s(\mathbf{\Sigma}\mathbf{V}^T). \quad (19)$$

A sub-optimal choice is to assume that the operator $\mathcal{T}_s$ keeps only the first s rows of $\mathbf{\Sigma}\mathbf{V}^T$, i.e., $\mathbf{X} = \mathbf{\Sigma}_s\mathbf{V}^T$ where $\mathbf{\Sigma}_s$ is the $\mathbf{\Sigma}$ matrix where we keep only the leading principal submatrix of size $s \times s$ and set to zero everything else. This is a good choice since the positive diagonal elements of $\mathbf{\Sigma}$ are sorted in decreasing order of their values and therefore we expect $\mathbf{X}$ to keep entries with large magnitude. In fact, $\|\mathbf{X}\|_F^2 = \sum_{k=1}^{s} \sigma_k^2$, where the σ_k's are the diagonal elements of $\mathbf{\Sigma}$, due to the fact that the rows of $\mathbf{V}^T$ have unit magnitude. Furthermore, with the same $\mathbf{X} = \mathbf{\Sigma}_s\mathbf{V}^T$ we have $\|\mathbf{Y} - \mathbf{U}\mathbf{X}\|_F^2 = \|\mathbf{U}(\mathbf{\Sigma} - \mathbf{\Sigma}_s)\mathbf{V}^T\|_F^2 = \sum_{k=s+1}^{n} \sigma_k^2$. We expect this error term to be relatively small since we sum over the smallest squared singular values of $\mathbf{Y}$. Therefore, with this choice of $\mathbf{U}$ and the optimal $\mathbf{X} = \mathcal{T}_s(\mathbf{U}^T\mathbf{Y})$ we have that $\|\mathbf{Y} - \mathbf{U}\mathbf{X}\|_F^2 \leq \sum_{k=s+1}^{n} \sigma_k^2$, i.e., the representation error is always smaller than the error given by the best s-rank approximation of $\mathbf{Y}$.

In G_m–DLA, with the sparse representations $\mathbf{X} = \mathcal{T}_s(\mathbf{U}^T\mathbf{Y})$ we proceed to iteratively construct each G-transform. At step k, the problem to be solved is similar to (18)

Algorithm 1 – G_m–DLA.
Fast Orthonormal Transform Learning.
Input: The dataset $\mathbf{Y} \in \mathbb{R}^{n \times N}$, the number of G-transforms m, the target sparsity s and the number of iterations K.
Output: The sparsifying square orthogonal transform $\mathbf{U} = \mathbf{G}_{i_m j_m} \dots \mathbf{G}_{i_2 j_2} \mathbf{G}_{i_1 j_1}$ and sparse representations $\mathbf{X}$ such that $\|\mathbf{Y} - \mathbf{U}\mathbf{X}\|_F^2$ is reduced.

Initialization:

1) Perform the economy size singular value decomposition of the dataset $\mathbf{Y} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$.

2) Compute sparse representations $\mathbf{X} = \mathcal{T}_s(\mathbf{U}^T\mathbf{Y})$.

3) For $k = 1, \dots, m$: with $\mathbf{X}$ and all previous $k-1$ G-transforms fixed and $\mathbf{G}_{i_t j_t} = \mathbf{I},\ t = k+1, \dots, m$, construct the new $\mathbf{G}_{i_k j_k}$ where indices (i_k, j_k) are given by (13) and $\tilde{\mathbf{G}}_{i_k j_k} = \mathbf{U}\mathbf{V}^T$ by the singular value decomposition $\mathbf{J}_{k\{i_k, j_k\}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ such that we minimize $\|\mathbf{Y} - \mathbf{G}_{i_k j_k}\mathbf{G}_{i_{k-1} j_{k-1}} \dots \mathbf{G}_{i_1 j_1}\mathbf{X}\|_F^2 = \|\mathbf{Y} - \mathbf{G}_{i_k j_k}\mathbf{X}_k\|_F^2$, as in (18) for $\mathbf{Y}_k = \mathbf{Y}$ and $\mathbf{J}_k = \mathbf{Y}\mathbf{X}_k^T$.

Iterations $1, \dots, K$:

1) For $k = 1, \dots, m$: two-step update of $\mathbf{G}_{i_k j_k}$, with $\mathbf{X}$ and all other transforms $\mathbf{G}_{i_t j_t},\ t \neq k$ fixed, such that (18) is minimized:

a) Update best indices (i_k, j_k) by (13).

b) With new indices, update the transform $\tilde{\mathbf{G}}_{i_k j_k} = \mathbf{U}\mathbf{V}^T$ by the singular value decomposition $\mathbf{Z}_{k\{i_k, j_k\}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, where $\mathbf{Z}_k = \mathbf{Y}_k\mathbf{X}_k^T$ as in (18).

2) Compute sparse representations $\mathbf{X} = \mathcal{T}_s(\mathbf{U}^T\mathbf{Y})$, where $\mathbf{U}$ is given by (17).

but all transforms indexed above k are currently the identity (not initialized) and will be computed in the following steps.

Notice that $\mathbf{Z} = \mathbf{Y}\mathbf{X}^T$, necessary to compute all the values C_{ij}, is computed fully only once before the iterative process. At each iteration of the algorithms only two columns of $\mathbf{Z}$ need to be recomputed. Therefore, the update of $\mathbf{Z}$ is trivial since it involves the linear combinations of two columns according to a G-transform multiplication (5).

The iterations of G_m–DLA update each G-transform sequentially, keeping all other constant, in order to minimize the current error term.

The algorithm is fast since the matrices involved in all computations can be updated from previous iterations. For example, at step $k+1$, notice from (18) that $\mathbf{Y}_{k+1} = \mathbf{G}_{i_{k+1} j_{k+1}}\mathbf{Y}_k$ and $\mathbf{X}_{k+1} = \mathbf{G}_{i_k j_k}\mathbf{X}_k$. The same observation holds for $\mathbf{Z}_{k+1} = \mathbf{G}_{i_{k+1} j_{k+1}}\mathbf{Y}_k\mathbf{X}_k^T\mathbf{G}_{i_k j_k}^T = \mathbf{G}_{i_{k+1} j_{k+1}}\mathbf{Z}_k\mathbf{G}_{i_k j_k}^T$. Of course, $\mathbf{Y}_1 = \mathbf{G}_{i_2 j_2}^T \dots \mathbf{G}_{i_m j_m}^T\mathbf{Y}$ and $\mathbf{X}_1 = \mathbf{X}$. We always need to construct $\mathbf{Z}_1$ from scratch since $\mathbf{X}$ has been fully updated in the sparse reconstruction step.

After all transforms are computed, the dictionary $\mathbf{U}$ is never explicitly constructed. We always remember its factorization (17) and apply it (directly or inversely) by sequentially applying the G-transforms in its composition. The total computational complexity of applying this dictionary for $\mathbf{U}^T\mathbf{Y}$ is $O(mN)$ which is $O(n\log(n)N)$ for sufficiently large m (of order $n \log n$). This is to be compared with the $O(n^2N)$ of a general orthogonal dictionary. Additionally, when consecutive G-transforms operate on different indices they can be applied in parallel, reducing the running time of G_m–DLA and that of manipulating the resulting dictionary.

The number of transforms m could be decided during the runtime of G_m–DLA based on the magnitude of the largest value C_{ij}. Since this magnitude decides the reduction in the objective function of our problem, a threshold can be introduced to decide on the fly if a new transform is worth adding to the factorization.

These observations are important from a computational perspective since the number of transforms is relatively high, $O(n \log n)$, and therefore their manipulation should be performed efficiently when learning the dictionary to keep the running time of G_m–DLA low.

Since each G-transform computed in our method maximally reduces the objective function and because the sparse reconstruction step is exact when using an orthogonal dictionary, we can guarantee that the proposed method monotonically converges to a local optimum point.

Remark 5. At each iteration of the proposed algorithm we update a single G-transform according to the maximum value C_{ij}. We have in fact the opportunity to update a maximum of $\lfloor n/2 \rfloor$ transforms simultaneously. We could for example partition the set $\{1, 2, \dots, n\}$ in pairs of two and construct the corresponding G-transforms such that the sum of their C_{ij} is maximized. With such a strategy fewer iterations are necessary but the problem of partitioning the indices such that the error is maximally reduced can be computationally demanding (all possible unique combinations of indices associations need to be generated). We expect G_m–DLA, as it is, to produce better results (lower representation error) due to the one-by-one transform update mechanism. Compared to the Householder approach [26] we again expect G_m–DLA to performs better since the optimization is made over two coordinates at a time.

Even so, there are several options regarding the ordering. We can process the G-transforms in the order of their indices or in a random order for example in an effort to try to avoid local minimum points. ■

Remark 6. After indices (i, j) are selected we have that $C_{ij} = 0$ and therefore this pair cannot be selected again until either index i or j participates in the construction of a future G-transform. This is because after constructing $\mathbf{G}_{ij}$ to minimize (10) we have that $\mathbf{Z}_{\{i,j\}}$ is updated to $\mathbf{Z}_{\{i,j\}}\tilde{\mathbf{G}}_{ij}^T = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\mathbf{V}\mathbf{U}^T = \mathbf{U}\mathbf{\Sigma}\mathbf{U}^T$ which is symmetric and positive definite due to the solution $\tilde{\mathbf{G}}_{ij} = \mathbf{U}\mathbf{V}^T$. ■

Remark 7. As previously discussed, the Procrustes solution $\mathbf{Q}$ is the best orthogonal minimizer of (11). It has been shown in [26] that with this $\mathbf{Q}$ we have that $\mathbf{T} = \mathbf{Y}\mathbf{X}^T\mathbf{Q}^T = \mathbf{U}\mathbf{\Sigma}\mathbf{U}^T$ is symmetric positive semidefinite. Since $\mathbf{Q}$ is the global minimizer, there cannot be a G-transform $\mathbf{G}_{ij}$ such that $\mathbf{G}_{ij}\mathbf{Q}$ further reduces the error. This means that all symmetric 2×2 sub-matrices $\mathbf{T}_{\{i,j\}} = \begin{bmatrix} T_{ii} & T_{ij} \\ T_{ij} & T_{jj} \end{bmatrix}$ of $\mathbf{T}$ are positive semidefinite, i.e., $T_{ii} + T_{jj} \geq 0$ and $T_{ii}T_{jj} \geq T_{ij}^2$ for all pairs (i, j). This observation needs to hold for any symmetric positive definite matrix $\mathbf{T}$. Unfortunately, the converse is not true in general.

This means that even with an appropriately large $m \sim O(n^2)$, G_m–DLA might not always be able to match the performance of Q–DLA. This is not a major concern since in this paper we explore fast transforms and therefore $m \ll n^2$.■

This concludes the presentation of the proposed G_m–DLA method. Based on similar principles next we provide a learning method for fast square but non-orthogonal dictionaries.

V. A METHOD FOR DESIGNING FAST, GENERAL, NON-ORTHOGONAL TRANSFORMS: R_m–DLA

In the case of orthogonal dictionaries, the fundamental building blocks like Householder reflectors and Givens rotations are readily available. This is not the case for general dictionaries. In this section we propose a building block for non-orthogonal structures in subsection A and then show how this can be used in a similar fashion to the G-transform to learn computationally efficient square non-orthogonal dictionaries by deriving the R_m–DLA method in subsection B.

A. A building block for fast non-orthogonal transforms

We assume no constraints on the variables p, q, r, t (these are four degrees of freedom) and therefore $\mathbf{R}_{ij}$ from (3) is no longer orthogonal in general. We propose to solve the following optimization problem

$$\underset{(i,j),\ \tilde{\mathbf{R}}_{ij}}{\text{minimize}} \ \|\mathbf{Y} - \mathbf{R}_{ij}\mathbf{X}\|_F^2 . \tag{20}$$

As in the G-transform case, we proceed with analyzing how indices (i, j) are selected and then how to solve the optimization problem (20), with the indices fixed. We define

$$\mathbf{Z} = \mathbf{Y}\mathbf{X}^T, \ \mathbf{W} = \mathbf{X}\mathbf{X}^T, \tag{21}$$

with entries Z_{ij} and W_{ij} respectively.

Solving (20) **for fixed** (i, j) leads to a least squares optimization problem as

$$\underset{\tilde{\mathbf{R}}_{ij}}{\text{minimize}} \ \left\| \begin{bmatrix} \mathbf{y}_i^T \\ \mathbf{y}_j^T \end{bmatrix} - \tilde{\mathbf{R}}_{ij} \begin{bmatrix} \mathbf{x}_i^T \\ \mathbf{x}_j^T \end{bmatrix} \right\|_F^2 , \tag{22}$$

where $\mathbf{y}_i^T, \mathbf{x}_i^T$ are i^{th} rows of $\mathbf{Y}$ and $\mathbf{X}$ respectively and whose solution is $\tilde{\mathbf{R}}_{ij} = \begin{bmatrix} Z_{ii} & Z_{ij} \\ Z_{ji} & Z_{jj} \end{bmatrix} \begin{bmatrix} W_{ii} & W_{ij} \\ W_{ji} & W_{jj} \end{bmatrix}^{-1}$.

Choosing (i, j) **in** (20) depends on the objective function value in (22) given by the least squares solution from above:

$$\begin{aligned} \left\| \begin{bmatrix} \mathbf{y}_i^T \\ \mathbf{y}_j^T \end{bmatrix} - \tilde{\mathbf{R}}_{ij} \begin{bmatrix} \mathbf{x}_i^T \\ \mathbf{x}_j^T \end{bmatrix} \right\|_F^2 &= \left\| \begin{bmatrix} \mathbf{y}_i^T \\ \mathbf{y}_j^T \end{bmatrix} \right\|_F^2 \\ - \text{tr} \left(\begin{bmatrix} Z_{ii} & Z_{ij} \\ Z_{ji} & Z_{jj} \end{bmatrix}^T \begin{bmatrix} Z_{ii} & Z_{ij} \\ Z_{ji} & Z_{jj} \end{bmatrix} \begin{bmatrix} W_{ii} & W_{ij} \\ W_{ji} & W_{jj} \end{bmatrix}^{-1} \right) . \end{aligned} \tag{23}$$

This, together with the development in (6), leads to

$$\begin{aligned} &\|\mathbf{Y} - \mathbf{R}_{ij}\mathbf{X}\|_F^2 = \|\mathbf{Y}\|_F^2 + \|\mathbf{X}\|_F^2 - 2\text{tr}(\mathbf{Z}) - C_{ij}, \\ &\text{with } C_{ij} = \left\| \begin{bmatrix} \mathbf{x}_i^T \\ \mathbf{x}_j^T \end{bmatrix} \right\|_F^2 - 2\text{tr} \left(\begin{bmatrix} Z_{ii} & Z_{ij} \\ Z_{ji} & Z_{jj} \end{bmatrix} \right) \\ &+ \text{tr} \left(\begin{bmatrix} Z_{ii} & Z_{ij} \\ Z_{ji} & Z_{jj} \end{bmatrix}^T \begin{bmatrix} Z_{ii} & Z_{ij} \\ Z_{ji} & Z_{jj} \end{bmatrix} \begin{bmatrix} W_{ii} & W_{ij} \\ W_{ji} & W_{jj} \end{bmatrix}^{-1} \right) . \end{aligned} \tag{24}$$

Since the matrices involved in the computation of C_{ij} are 2×2 we can use the trace formula and the inversion of a 2×2 matrix formula to explicitly calculate

$$\begin{aligned} C_{ij} = W_{ii} + W_{jj} - 2(Z_{ii} + Z_{jj}) \\ + \frac{W_{ii}(Z_{ij}^2 + Z_{jj}^2) + W_{jj}(Z_{ii}^2 + Z_{ji}^2)}{W_{ii}W_{jj} - W_{ij}W_{ji}} \\ - \frac{(Z_{ii}Z_{ij} + Z_{ji}Z_{jj})(W_{ij} + W_{ji})}{W_{ii}W_{jj} - W_{ij}W_{ji}} . \end{aligned} \tag{25}$$

Finally, to solve (20) we select the indices as

$$(i^\star, j^\star) = \underset{j>i}{\arg\max} \ C_{ij}, \tag{26}$$

and then solve a least square problem to construct $\tilde{\mathbf{R}}_{i^\star j^\star}$. The C_{ij} are computed only when $W_{ii}W_{jj} - W_{ij}W_{ji} \neq 0$, otherwise $C_{ij} = -\infty$. To compute each C_{ij} in (25) we need 24 operations and there are $\frac{n(n-1)}{2}$ such C_{ij}. The computational burden is dominated by constructing $\mathbf{Z} = \mathbf{Y}\mathbf{X}^T, \mathbf{W} = \mathbf{X}\mathbf{X}^T$ which take $2snN$ and snN operations, respectively.

Remark 8. A necessary condition for a dictionary $\mathbf{D} \in \mathbb{R}^{n\times n}$ to be a local minimum point for the dictionary learning problem is that all $C_{ij} = 0$ for $\mathbf{Z} = \mathbf{Y}\mathbf{X}^T\mathbf{D}^T, \mathbf{W} = \mathbf{D}\mathbf{X}\mathbf{X}^T\mathbf{D}^T$.■

This concludes our discussion for one transform $\mathbf{R}_{ij}$. Notice that just like in the case of one G-transform, the solution given here finds the optimum $\mathbf{R}_{ij}$ to minimize (20).

B. A method for designing fast general transforms: R_m–DLA

Similarly to G_m–DLA, we now propose to construct a general dictionary $\mathbf{D} \in \mathbb{R}^{n\times n}$ with the following structure:

$$\mathbf{D} = \mathbf{R}_{i_m j_m} \dots \mathbf{R}_{i_2 j_2} \mathbf{R}_{i_1 j_1} \mathbf{\Delta}. \tag{27}$$

The value of m is a choice of the user. For example, if we choose m to be $O(n \log n)$ the dictionary $\mathbf{D}$ can be applied in $O(n \log n)$ computational complexity – similar to the classical fast transforms. The goal of this section is to propose a learning method that constructs such a general dictionary. As the transformations $\mathbf{R}_{ij}$ are general, the diagonal matrix $\mathbf{\Delta} \in \mathbf{R}^{n\times n}$ is there to ensure that all columns $\mathbf{d}_j$ of $\mathbf{D}$ are normalized $\|\mathbf{d}_j\|_2 = 1$ (as in the formulation (1)). This normalization does not affect the performance of the method since $\mathbf{D}\mathbf{X}$ is equivalent to $(\mathbf{D}\mathbf{\Delta})(\mathbf{\Delta}^{-1}\mathbf{X})$.

We fix the representations $\mathbf{X}$ and all transforms in (27) except for the k^{th} transform $\mathbf{R}_{i_k j_k}$. Moreover, all transforms $\mathbf{R}_{i_{k+1} j_{k+1}}, \dots, \mathbf{R}_{i_m j_m}$ are set to $\mathbf{I}$. Because the transforms $\mathbf{R}_{ij}$ are not orthogonal we cannot access directly any transform $\mathbf{R}_{i_k j_k}$ in (27), but only the left most one $\mathbf{R}_{i_m j_m}$. In this case, to optimize the dictionary $\mathbf{D}$ only for this $\mathbf{R}_{i_k j_k}$ transform we reach the objective

$$\|\mathbf{Y} - \mathbf{R}_{i_k j_k} \dots \mathbf{R}_{i_2 j_2} \mathbf{R}_{i_1 j_1} \mathbf{X}\|_F^2 = \|\mathbf{Y} - \mathbf{R}_{i_k j_k} \mathbf{X}_k\|_F^2. \tag{28}$$

Therefore, our goal is to solve

$$\underset{\mathbf{R}_{i_k j_k}}{\text{minimize}} \ \|\mathbf{Y} - \mathbf{R}_{i_k j_k} \mathbf{X}_k\|_F^2. \tag{29}$$

Notice that we have reduced the problem to the formulation in (20) whose full solution is outlined in the previous section. We can apply this procedure for all G-transforms in the

Algorithm 2 – $\mathbf{R}_m$–DLA.
Fast Non-orthogonal Transform Learning.
Input: The dataset $\mathbf{Y} \in \mathbb{R}^{n \times N}$, the number of $\mathbf{R}_{ij}$ transforms m, the target sparsity s and the number of iterations K.
Output: The sparsifying square non-orthogonal transform $\mathbf{D} = \mathbf{R}_{i_m j_m} \dots \mathbf{R}_{i_2 j_2} \mathbf{R}_{i_1 j_1} \mathbf{\Delta}$ and sparse representations $\mathbf{X}$ such that $\|\mathbf{Y} - \mathbf{D}\mathbf{X}\|_F^2$ is reduced.

Initialization:

1) Perform the economy size singular value decomposition of the dataset $\mathbf{Y} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$.
2) Compute sparse representations $\mathbf{X} = \mathcal{T}_s(\mathbf{U}^T\mathbf{Y})$.

Iterations $1, \dots, K$:

1) For $k = 1, \dots, m$: with $\mathbf{X}$ and all previous $k-1$ R-transforms fixed and $\mathbf{R}_{i_t j_t} = \mathbf{I},\ t = k+1, \dots, m$, construct the new $\mathbf{R}_{i_k j_k}$ where indices (i_k, j_k) are given by (26) and $\tilde{\mathbf{R}}_{i_k j_k}$ is given by the least squares solution that minimizes (28).
2) Compute $\mathbf{\Delta}$ in (27) such that $\|\mathbf{d}_j\|_2 = 1$.
3) Compute sparse representations $\mathbf{X}$=OMP$(\mathbf{D}, \mathbf{Y}, s)$ where $\mathbf{D}$ is given in (27).

Iterations $1, \dots, K$:

1) For $k = 1, \dots, m$: with $\mathbf{X}$, indices (i_k, j_k) and all transforms except the k^{th} fixed, update only the non-zero part of $\mathbf{R}_{i_k j_k}$, denoted $\tilde{\mathbf{R}}_{i_k j_k}$, such that (30) is minimized.
2) Compute $\mathbf{\Delta}$ in (27) such that $\|\mathbf{d}_j\|_2 = 1$.
3) Compute sparse representations $\mathbf{X}$=OMP$(\mathbf{D}, \mathbf{Y}, s)$ where $\mathbf{D}$ is given in (27).

product of $\mathbf{D}$ and therefore a full update procedure presents itself: we will sequentially update each transform in (27), from the right to the left, and then the sparse representations until convergence or for a total number of iterations K.

Once these iterations terminate we can refine the result. As previously mentioned, we cannot arbitrarily update a transform $\tilde{\mathbf{R}}_{i_k j_k}$ because this transform is not orthogonal. But we can update its non-zero part $\tilde{\mathbf{R}}_{i_k j_k}$. Consider the development:

$$\begin{aligned}
&\|\mathbf{Y} – \mathbf{R}_{i_m j_m} \dots \mathbf{R}_{i_{k+1} j_{k+1}} \mathbf{R}_{i_k j_k} \mathbf{R}_{i_{k-1} j_{k-1}} \dots \mathbf{R}_{i_1 j_1} \mathbf{X}\|_F^2 \\
=&\|\mathbf{Y} - \mathbf{B}_k \mathbf{R}_{i_k j_k} \mathbf{X}_k\|_F^2 \\
=&\|\text{vec}(\mathbf{Y}) - (\mathbf{X}_k^T \otimes \mathbf{B}_k)\text{vec}(\mathbf{R}_{i_k j_k})\|_F^2 \\
=&\left\| \text{vec}(\mathbf{Y}) - \sum_{t \in \{1,\dots,n\} \setminus \{i_k, j_k\}} ((\mathbf{X}_k^T)_t \otimes (\mathbf{B}_k)_t) - \mathbf{C}\mathbf{x} \right\|_F^2 \\
=&\|\mathbf{w} - \mathbf{C}\mathbf{x}\|_F^2,
\end{aligned} \tag{30}$$

where $\mathbf{x} = \text{vec}(\tilde{\mathbf{R}}_{i_k j_k}) \in \mathbb{R}^4$ and $\mathbf{C} = [(\mathbf{X}_k^T)_{i_k} \otimes (\mathbf{B}_k)_{i_k} \quad (\mathbf{X}_k^T)_{i_k} \otimes (\mathbf{B}_k)_{j_k} \quad (\mathbf{X}_k^T)_{j_k} \otimes (\mathbf{B}_k)_{i_k} \quad (\mathbf{X}_k^T)_{j_k} \otimes (\mathbf{B}_k)_{j_k}] \in \mathbb{R}^{nN \times 4}$. We have denoted by $(\mathbf{B}_k)_{i_k}$ the i_k^{th} column of $\mathbf{B}_k$ and $\otimes$ is the Kronecker product. To develop (30) we have used the fact that the Frobenius norm is an elementwise operator, the structure of $\mathbf{R}_{i_k j_k}$ and the fact that

$$\text{vec}(\mathbf{B}_k \mathbf{R}_{i_k j_k} \mathbf{X}_k) = (\mathbf{X}_k^T \otimes \mathbf{B}_k)\text{vec}(\mathbf{R}_{i_k j_k}). \tag{31}$$

The $\mathbf{x}$ that minimizes (30) is given by the least squares solution $\mathbf{x} = (\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\mathbf{w}$. Therefore, once the product of the m transforms is constructed we can update the non-zero part of any transform to further reduce the objective function. What we cannot do is update the indices (i_k, j_k) on which the calculation takes place, these stay the same.

Therefore, we propose a learning procedure that has two sets of iterations: the first constructs the transforms $\mathbf{R}_{i_k j_k}$ in a rigid manner, ordered from right to left most, and the second only updates the non-zero parts $\tilde{\mathbf{R}}_{i_k j_k}$ of all the transforms without changing the coordinates (i_k, j_k). The full procedure we propose, called R_m–DLA, is detailed in Algorithm 2. This algorithm has two main parts which we will now describe.

The initialization of $\mathbf{R}_m$–DLA has the goal to construct the sparse representation matrix $\mathbf{X} \in \mathbb{R}^{n \times N}$. We have several options in this step. We can construct $\mathbf{X}$ in the same way as for G_m–DLA from the singular value decomposition of the dataset or by running another dictionary learning algorithm (like the K–SVD [8] for example) and use the $\mathbf{X}$ it constructs.

The iterations of $\mathbf{R}_m$–DLA are divided into two sets. The goal of the first set of iterations is to decide upon all the indices (i_k, j_k) while the second set of iterations optimizes over the non-zero components of all the transforms in the factorization with the fixed indices previously decided.

The proposed R_m–DLA is can be itself efficiently implemented. When iteratively solving problems as (28) we have that $\mathbf{X}_{k+1} = \mathbf{R}_{i_k j_k} \mathbf{X}_k$ with $\mathbf{X}_1 = \mathbf{X}$ while when iteratively solving problems as (30) we have that $\mathbf{X}_{k+1} = \mathbf{R}_{i_k j_k} \mathbf{X}_k$ and $\mathbf{B}_{k+1} = \mathbf{B}_k \mathbf{R}_{i_k j_k}^{-1}$ with $\mathbf{X}_1 = \mathbf{X}$ and $\mathbf{B}_1 = \mathbf{R}_{i_m j_m} \dots \mathbf{R}_{i_2 j_2}$. The explicit inverse $\mathbf{R}_{i_k j_k}^{-1}$ is not computed, instead the equivalent linear system for 2 variables can be efficiently solved.

The updates of all the transforms $\mathbf{R}_{i_k j_k}$ monotonically decrease our objective function since we solve exactly the optimization problems in these variables. Unfortunately, normalizing to unit ℓ_2 norm the columns of the transform and constructing the sparse approximations via OMP, which is not an exact optimization step, may cause increases in the objective function. For these reasons, monotonic convergence of R_m–DLA to a local minimum point cannot be guaranteed. For this reason, at all times we keep track of the best solution pair $(\mathbf{D}, \mathbf{X})$ and return it at the end of each iterative process.

This concludes our discussion of R_m–DLA. We now move to discuss the computational complexity of the transforms created by the proposed methods and to show experimentally their representation capabilities.

VI. The computational complexity of using learned transforms

In this section we look at the computational complexity of using the learned dictionaries to create the sparse representations on a dataset $\mathbf{Y}$ of size $n \times N$. We are in a computational regime where we assume dimensions obey

$$s \ll n \ll N. \tag{32}$$

The computational complexity of using a general non-orthogonal dictionary $\mathbf{A}$ of size $n \times n$ in sparse recovery problems with Batch–OMP [15] is

$$N_{\mathbf{A}} \approx (2n^2 + s^2 n + 3sn + s^3)N + n^3. \tag{33}$$

The cost of n^3 is associated with the construction of the Gram matrix of the dictionary and it does not depend on the number of samples N in the data. The total number of operations is dominated by constructing the projections in the dictionary column space which takes $2n^2$ operations per sample. The other operations dependent on the sparsity s and express the cost of iteratively finding the support of the sparse approximation.

The computational complexity of using an orthogonal dictionary $\mathbf{Q}$ designed via Q–DLA is

$$N_{\mathbf{Q}} \approx (2n^2 + ns)N. \tag{34}$$

As in the general case, the cost is dominated by constructing the projections $\mathbf{Q}^T\mathbf{Y}$ which takes $2n^2$ operations for each of the N columns in $\mathbf{Y}$. The cost of ns expresses the approximate work done to identify the largest s entries in magnitude in the representation of each data sample. This can be performed in an efficient manner by keeping the s largest components in magnitude while the projections are computed for each data sample. Compared with (33), the iterative steps of constructing the support of the OMP solution for each sample and the construction of the Gram matrix (which is the identity matrix in this case) is no longer needed.

The same operation with a dictionary $\mathbf{U}$ as (17) computed via G_{m_1}–DLA takes

$$N_{\mathbf{U}} \approx (6m_1 + ns)N. \tag{35}$$

The result is similar to (34) but now the cost of constructing the projections $\mathbf{U}^T\mathbf{Y}$ takes now only $6m_1$ operations per data sample. Here is where the G-transform factorization is used explicitly to reduce the computational complexity.

Finally, with a dictionary $\mathbf{D}$ as (27) computed via R_{m_2}–DLA the sparse approximation step via Batch–OMP [15] takes

$$N_{\mathbf{D}} \approx (6m_2 + n + s^2n + 3sn + s^3)N + 6m_2n. \tag{36}$$

In this case, the cost of building the projections $\mathbf{D}^T\mathbf{Y}$ takes $6m_2$ operations to apply each $\mathbf{R}_{ij}$ transform and then n operations to apply the scaling of the diagonal $\mathbf{\Delta}$. Simplifications occur also for the construction of the symmetric Gram matrix $\mathbf{D}^T\mathbf{D}$ which now takes $6m_2n$ operations, instead of the regular n^3 operations. This later simplification might not be significant since it is not dependent on the size of the dataset N.

A dictionary $\mathbf{U}$ designed via G_{m_1}–DLA has approximately the same computationally complexity as a general orthogonal dictionary $\mathbf{Q}$ designed via Q–DLA when

$$m_1 = \left\lfloor \frac{n^2}{3} \right\rfloor. \tag{37}$$

Because any orthogonal matrix can be factorized as a product of $\frac{n(n-1)}{2}$ G-transforms and because of the upper limit imposed in (37) it is clear that we cannot express any orthogonal dictionary as an efficient transform for sparse representations. In some cases, the full orthogonal dictionary $\mathbf{Q}$ might be more efficient than its factorization with G-transforms. In general, the representation error achieved by general orthogonal dictionaries designed via Q–DLA is a performance limit for G-transform based dictionaries.

A similar comparison can be made between the computational complexity of a general dictionary $\mathbf{A}$ and that of a dictionary $\mathbf{D}$ composed of m_2 transformations $\mathbf{R}_{ij}$. Their complexities approximately match when

$$m_2 = \left\lfloor \frac{(2n^2 + s^2n + 3sn + s^3)N + n^3}{6(N+n)} \right\rfloor \overset{N\to\infty}{\approx} \left\lfloor \frac{n^2}{3} \right\rfloor. \tag{38}$$

This shows that for both G_m–DLA and R_m–DLA the computationally efficient regimes are when $m \sim O(n)$ or in general $m \ll n^2$.

A last comment regards the comparison between dictionaries created with G_{m_1}–DLA and R_{m_2}–DLA. When $m_1 = m_2$ we expect R_{m_2}–DLA to perform better but at a higher computational cost. Assuming large datasets $N \to \infty$ and low sparsity $s \ll n$, computational complexities approximately match when

$$m_1 \approx \left\lfloor m_2 + \frac{(s^2 + 3s + 1)n}{6} \right\rfloor. \tag{39}$$

Due to the use of the OMP procedure for non-orthogonal dictionaries to create the sparse approximations, dictionaries designed via R_m–DLA are much more computationally complex than the orthogonal dictionaries designed via G_m–DLA. Otherwise, as depicted in (39), for the same representation performance the orthogonal dictionaries may contain many more G-transforms in their factorization than $\mathbf{R}_{ij}$ transforms contained in the factorization of a non-orthogonal dictionary. As a consequence, it may be that orthogonal dictionaries are always more computationally efficient than general dictionaries for approximately equal representation capabilities. A definite advantage of R_m–DLA is that it has the potential to create dictionaries that go below representation errors given by orthogonal dictionaries designed via Q–DLA, the performance limit of G_m–DLA.

Using these approximate complexities, we discuss in the results section the representation performance versus the computational complexity trade-off that the dictionaries constructed via the proposed methods display.

VII. Experimental results

In this section we provide experimental results that show how transforms designed via the proposed methods G_m–DLA and R_m–DLA behave on image data.

The input data that we consider are taken from popular test images from the image processing literature (pirate, peppers, boat etc.). The test dataset $\mathbf{Y} \in \mathbb{R}^{n\times N}$ consists of 8×8 non-overlapping patches with their means removed and normalized $\mathbf{Y} \leftarrow \mathbf{Y}/255$. We choose to compare the proposed methods on image data since in this setting fast transforms that perform very well, like the Discrete Cosine Transform (DCT) [41] for example, are available. Our goal is to provide dictionaries based on factorizations like (17) and (27) that perform well in terms of representation error with a small number m of basic transforms in their composition. All algorithms run for $K = 150$ iterations and there are $N = 12288$ image patches in the dataset $\mathbf{Y}$ each of size $n = 64$.

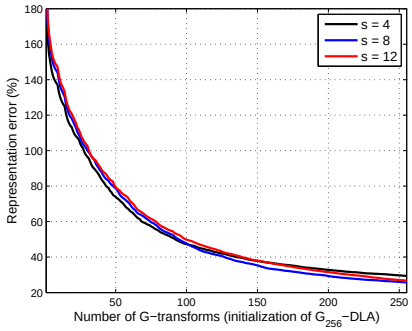

Fig. 1. For the proposed G_{256}–DLA we show the relative representation error (40) in the initialization steps for the dataset $\mathbf{Y}$ created from the patches of the images couple, peppers and boat with sparsity $s \in \{4, 8, 12\}$. Notice that in general the representation error can surpass 100%, for example, for orthogonal dictionaries, the maximum value $\epsilon = 4$ is achieved when $\mathbf{X} = -\mathbf{Y}$ and $\mathbf{D} = \mathbf{I}$ in (40).

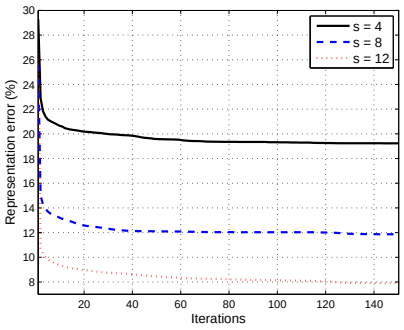

Fig. 2. For the same experimental setup as in Figure 1, we show the representation error for the $K = 150$ regular iterations of G_{256}–DLA.

To measure the quality of a dictionary $\mathbf{D}$ we choose to evaluate the relative representation error

$$\epsilon = \|\mathbf{Y} - \mathbf{D}\mathbf{X}\|_F^2 \|\mathbf{Y}\|_F^{-2} \ (\%). \tag{40}$$

Figures 1 and 2 show the evolution of G_{256}–DLA for $K = 150$ iterations (including the initialization procedure, i.e., the first 256 steps of the algorithm). The figures show how effective the initialization is in reducing the representation error for any sparsity level. Notice that the initialization procedure provides similar results regardless of the sparsity level s. The $K = 150$ iterations of G_{256}–DLA further lower the representation error providing better results with larger sparsity level. As previously discussed, each step of the algorithm monotonically decreases the objective function of the dictionary learning problem until convergence.

Figure 3 shows how G_m–DLA evolves with the number of transforms m and the sparsity s. As expected, increasing the number of transforms $\mathbf{G}_{ij}$ and $\mathbf{R}_{ij}$ in the factorization always lowers the representation error but with diminishing returns as m increases. This figure helps choose the number of transforms m while balancing between the computational complexity and representation performance. Large decreases in the representation error are seen up to $m = 96$ or $m = 128$ while thereafter increasing m brings smaller benefits. Also, with higher sparsity levels the number of transforms m becomes less relevant. We notice that with $s = 12$ the

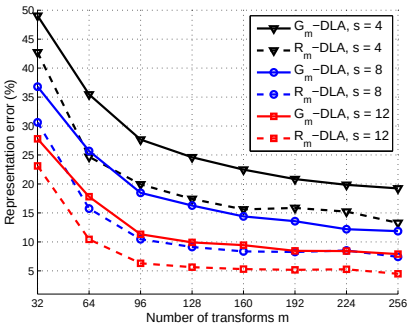

Fig. 3. Performance of G_m–DLA and R_m–DLA in terms of the relative representation error (40) for different sparsity levels $s \in \{4, 8, 12\}$.

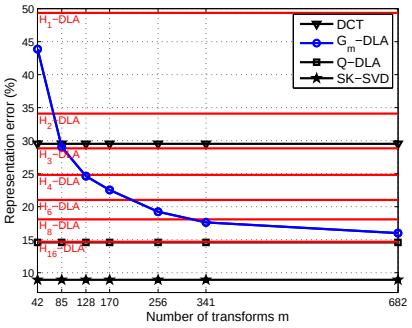

Fig. 4. Comparisons, in terms of relative representation errors (40), of G_m–DLA against the DCT, Q–DLA [33], SK–SVD [40] and Householder based orthogonal dictionaries [26] denoted here H_p–DLA where p is the number of reflectors in the factorization of the dictionary. The number of transforms m is chosen so that computational complexity comparisons against H_p–DLA is possible. Computational complexity approximately match between: H_1–DLA and G_{42}–DLA, H_2–DLA and G_{85}–DLA, H_3–DLA and G_{128}–DLA, H_4–DLA and G_{170}–DLA, H_6–DLA and G_{256}–DLA, H_8–DLA and G_{341}–DLA, H_{16}–DLA and G_{682}–DLA. The sparsity level is set to $s = 4$ for all methods. We use the SK–SVD to build a square, non-orthogonal, dictionary.

representation performance hits a plateau after $m \geq 128$ transforms.

An interesting point of comparison is between the dictionaries constructed via G_m–DLA and H_p–DLA [26]. Figure 4 provides a detailed comparison between the two. A matrix-vector multiplication takes $4n$ operations for a reflector and only 6 operations for a G-transform. If we compare the computational complexities of the dictionaries constructed by the two methods we find approximate equality between H_p–DLA and $G_{\lfloor \frac{2}{3} np \rfloor}$–DLA. Notice from this figure that for a low m the G-transform approach provides better results than the Householder approach while also enjoying lower computational complexity. As the complexity of the dictionaries increases (larger number of G-transforms or reflectors) the gap between the two approaches decreases. The most complex dictionaries are designed via H_{16}–DLA and G_{682}–DLA and they closely match the performance of the general orthogonal dictionary learning approach Q–DLA while still keeping a computational advantage. In this case, the Householder approach keeps a slight edge in representation performance. Since the proposed approach updates the G-transforms sequentially the probability of getting stuck in local minimum points is more likely with large m. The difficulties that G_m–DLA encounters for large m are also discussed in Remark 6.

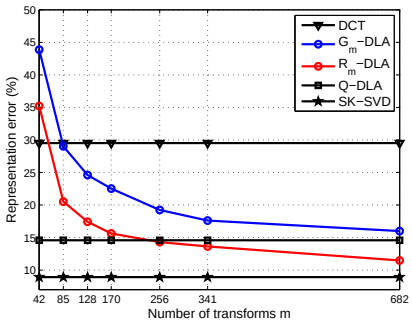

Fig. 5. For the same experimental setup as in Figure 4 we compare G_m–DLA against R_m–DLA.

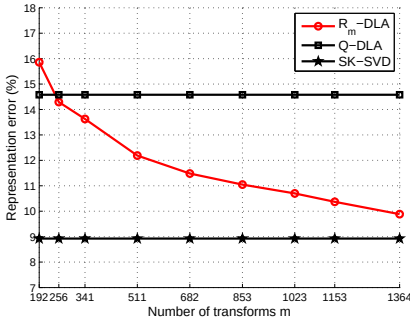

Fig. 6. The evolution of R_m–DLA for m large enough to outperform any orthogonal dictionary.

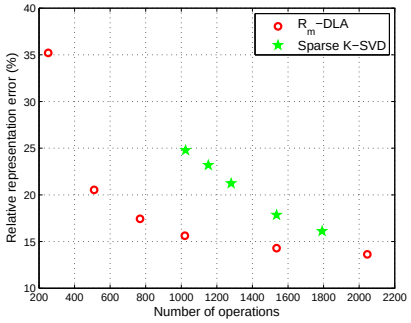

Fig. 7. Pareto curves for R_m–DLA and the Sparse K–SVD approach [42]. We consider the representation error in (40) and the number of operations necessary to perform $\mathbf{D}^T\mathbf{y}$ given a target vector $\mathbf{y} \in \mathbb{R}^n$. We train five square dictionaries $\mathbf{D} = \mathbf{\Phi S}$ with the Sparse K–SVD approach, each with a different sparsity parameter $p \in \{2, 3, 4, 6, 8\}$ in the matrix $\mathbf{S}$. For a transform created with R_m–DLA matrix-vector multiplication takes $6m$ operations. The experimental setup for training the transforms is the same as in Figure 4.

It is also interesting to see that the representation performance of the DCT is matched by H_3–DLA and G_{85}–DLA. The computational complexity of H_3–DLA approximately matches that of the DCT [41] (based on the FFT) while G_{85}–DLA is actually computationally simpler than the DCT. In fact, any dictionary constructed by G_m–DLA for $85 \leq m \leq 128$ is faster and provides better representations than the DCT.

Figure 5 compares the G_m–DLA against the R_m–DLA for the same number of transforms m in their factorizations. R_m–DLA always outperforms G_m–DLA since the $\mathbf{G}_{ij}$ is a constrained version of $\mathbf{R}_{ij}$. Unfortunately, the non-orthogonal transforms also have much higher computational complexity than their orthogonal counterparts in the sparse approximation step. For example, the computational complexity is approximately equivalent between dictionaries designed via R_{42}–DLA and G_{351}–DLA. The main benefit of non-orthogonal transforms is that ultimately, for large enough m, their performance surpasses that of general (computationally inefficient) orthogonal transforms designed via Q–DLA. In our case this happens for $m \geq 256$. The performance of the DCT is approximately matched by R_{50}–DLA. Surprisingly, less than n factors in the product of the transform suffice to match the performance of the classical DCT transform for sparse recovery. This highlights the way dictionaries designed via R_m–DLA balance the computationally efficiency and representation performance trade-off, i.e., one R-transform gives 4 degrees of freedom for the cost of 6 operations.

Figure 6 shows the performance of R_m–DLA in a regime close to the results of the SK–SVD dictionary learning method [40]. We use the SK–SVD to construct a square dictionary, i.e., a dictionary with n atoms. The complexity of the dictionary designed via R_{1364}–DLA matches that of the dictionary designed via SK–SVD while there is a small performance gap between the two. We notice experimentally that the iterative procedure of R_m–DLA improves performance always when increasing m but the probability of getting stuck in local minimum points increases. Therefore, just as G_m–DLA has some trouble matching the performance of Q–DLA, R_m–DLA has trouble exactly matching the performance of SK–SVD.

In the last experimental setup we compare our R_m–DLA with the previously proposed Sparse K–SVD approach [42]. We use the Sparse K–SVD to build a square dictionary $\mathbf{D} = \mathbf{\Phi S} \in \mathbb{R}^{n \times n}$ where $\mathbf{\Phi}$ is a well-known classic transform (in our case the DCT) and $\mathbf{S} \in \mathbb{R}^{n \times n}$ is matrix with only p non-zero entries per column. In this fashion, matrix-vector multiplication like $\mathbf{D}^T\mathbf{y} = \mathbf{S}^T\mathbf{\Phi}^T\mathbf{y}$ takes $2pn + C$ operations, where C is the cost of applying the DCT (in our case, this is the same as using a transform designed via G_{128}–DLA or R_{128}–DLA). R_m–DLA performs consistently better than the Sparse K–SVD for very fast transforms while the performance gap closes for very low representation errors. The Sparse K–SVD suffers from the fact that the fast transform $\mathbf{\Phi}$ is fixed and therefore the optimization takes place over only pn degreed of freedom. We restrict ourselves to square transforms and avoid the comparison with overcomplete dictionaries designed via the K–SVD or the Sparse K–SVD. Experimental insights into how the representation performance scales with the number of atoms in the dictionary are given in [40], [43].

When designing a very fast orthogonal transform (whose complexity let us say is order n or $n \log n$) then G_m–DLA provides very good results while achieving the lowest computational complexity. For improved performance, more complex orthogonal transforms perform better when designed via H_p–DLA. If representation capabilities is the only performance metric then the non-orthogonal transforms designed by R_m–DLA are the weapon of choice. For large m both G_m–DLA and R_m–DLA can suffer from long running times. For example, G_{128}–DLA takes several minutes to terminate while R_{128}–DLA's running time is close to ten minutes on a modern Intel i7 computer system. We note that the algorithms are

implemented in Matlab®. A careful implementation in a lower level compiled programming language will drive these running times much lower and reduce the memory footprint.

VIII. Conclusions

In this paper we present practical procedures to learn square orthogonal and non-orthogonal dictionaries already factored into a fixed number of computationally efficient blocks. We show how effective the dictionaries constructed via the proposed methods are on image data where we compare against the fast cosine transform on one side and general non-orthogonal and orthogonal dictionaries on the other. We also show comparisons with a recently proposed method that constructs Householder based orthogonal dictionaries. We show empirically that the proposed methods construct transforms that provide an improved trade-off between computational complexity and representation performance among the methods we consider. We are able to construct transforms that exhibit lower computational efficiency and lower representation error than the fast cosine transform for image data. We expect the current work to extend the use of learned transforms in time critical scenarios and to devices where, due to power limitations, only low complexity algorithms can be deployed.

References

[1] I. Tosic and P. Frossard, "Dictionary learning," *IEEE Signal Processing Magazine*, vol. 28, no. 2, pp. 27–38, 2011.

[2] A. M. Bruckstein, D. L. Donoho, and M. Elad, "From sparse solutions of systems of equations to sparse modeling of signals and images," *SIAM Review*, vol. 51, pp. 34–81, 2009.

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

[4] C. Rusu, R. Mendez-Rial, N. Gonzalez-Prelcic, and R. W. Heath, "Low complexity hybrid sparse precoding and combining in millimeter wave MIMO systems," in *Proc. IEEE ICC*, 2015.

[5] J. Mairal, F. Bach, and J. Ponce, "Task-driven dictionary learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 4, pp. 791–804, 2012.

[6] A. M. Tillmann, "On the computational intractability of exact and approximate dictionary learning," *IEEE Signal Processing Letters*, vol. 22, no. 1, pp. 45–49, 2014.

[7] K. Engan, S. O. Aase, and J. H. Husøy, "Method of optimal directions for frame design," in *Proc. IEEE ICASSP*, 1999, pp. 2443–2446.

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

[9] ——, "On the uniqueness of overcomplete dictionaries, and a practical way to retrieve them," *Linear Algebra and Applications*, vol. 416, pp. 48–67, 2006.

[10] A. Rakotomamonjy, "Direct optimization of the dictionary learning problem," *IEEE Trans. Sig. Proc.*, vol. 61, no. 22, pp. 5495–5506, 2013.

[11] J. Cooley and J. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965.

[12] J. A. Tropp, "Just relax: Convex programming methods for subset selection and sparse approximation," *IEEE Trans. Inf. Theory*, vol. 52, pp. 1030–1051, 2006.

[13] ——, "Greed is good: Algorithmic results for sparse approximation," *IEEE Trans. Inf. Theory*, vol. 50, pp. 2231–2242, 2004.

[14] D. L. Donoho, A. Maleki, and A. Montanari, "Message-passing algorithms for compressed sensing," *Proceedings of the National Academy of Sciences*, vol. 106, no. 45, pp. 18 914–18 919, 2009.

[15] R. Rubinstein, M. Zibulevsky, and M. Elad, "Efficient implementation of the K-SVD algorithm using batch orthogonal matching pursuit," *CS Technion*, 2008.

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

[17] ——, "ℓ_0 sparsifying transform learning with efficient optimal updates and convergence guarantees," *IEEE Trans. Signal Process.*, vol. 63, no. 9, pp. 2389–2404, 2015.

[18] M. Mathieu and Y. LeCun, "Fast approximation of rotations and Hessians matrices," *arXiv:1404.7195*, 2014.

[19] R. Rubinstein, M. Zibulevsky, and M. Elad, "Double sparsity: learning sparse dictionaries for sparse signal approximation," *IEEE Trans. Sig. Proc.*, vol. 58, no. 3, pp. 1553–1564, 2010.

[20] L. L. Magoarou and R. Gribonval, "Chasing butterflies: In search of efficient dictionaries," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, April 2015, pp. 3287–3291.

[21] O. Chabiron, F. Malgouyres, J.-Y. Tourneret, and N. Dobigeon, "Toward fast transform learning," *Technical report*, 2013.

[22] C. Rusu, B. Dumitrescu, and S. A. Tsaftaris, "Explicit shift-invariant dictionary learning," *IEEE Signal Proc. Let.*, vol. 21, no. 1, pp. 6–9, 2014.

[23] C. Rusu and B. Dumitrescu, "Block orthonormal overcomplete dictionary learning," in *21st European Sig. Proc. Conf.*, 2013, pp. 1–5.

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

[25] J. Sulam, B. Ophir, M. Zibulevsky, and M. Elad, "Trainlets: Dictionary learning in high dimensions." *IEEE Trans. Signal Processing*, vol. 64, no. 12, pp. 3180–3193, 2016.

[26] C. Rusu, N. Gonzalez-Prelcic, and R. Heath, "Fast orthonormal sparsifying transforms based on Householder reflectors," *IEEE Trans. Sig. Process.*, vol. 64, no. 24, pp. 6589–6599, 2016.

[27] G. K. Wallace, "The JPEG still picture compression standard," *IEEE Trans. Consumer Electronics*, vol. 38, no. 1, 1992.

[28] A. Agarwal, A. Anandkumar, P. Jain, P. Netrapalli, and R. Tandon, "Learning sparsely used overcomplete dictionaries," in *JMLR Workshop and Conference Proceedings*, vol. 35, 2014, pp. 1–15.

[29] O. G. Sezer, O. Harmanci, and O. G. Guleryuz, "Sparse orthonormal transforms for image compression," in *Proc. IEEE ICIP*, 2008, pp. 149–152.

[30] O. G. Sezer, O. G. Guleryuz, and Y. Altunbasak, "Approximation and compression with sparse orthonormal transforms," *IEEE Trans. Image Proc.*, vol. 24, no. 8, pp. 2328–2343, 2015.

[31] A. Dremeau and C. Herzet, "An EM-algorithm approach for the design of orthonormal bases adapted to sparse representations," in *Proc. IEEE ICASSP*, 2010, pp. 2046–2049.

[32] H. Schutze, E. Barth, and T. Martinetz, "Learning efficient data representations with orthogonal sparse coding," *IEEE Trans. Comp. Imaging*, vol. 2, no. 3, pp. 177–189, 2016.

[33] S. Lesage, R. Gribonval, F. Bimbot, and L. Benaroya, "Learning unions of orthonormal bases with thresholded singular value decompositon," in *Proc. IEEE ICASSP*, 2005, pp. 293–296.

[34] P. Schonemann, "A generalized solution of the orthogonal Procrustes problem," *Psychometrika*, vol. 31, no. 1, pp. 1–10, 1966.

[35] G. H. Golub and C. F. Van Loan, *Matrix Computations*. Johns Hopkins University Press, 1996.

[36] R. A. Horn and C. R. Johnson, *Matrix analysis*. Cambridge University Press, 1985.

[37] C. Guangzhi, L. R. Bachega, and C. A. Bouman, "The sparse matrix transform for covariance estimation and analysis of high dimensional signals," *IEEE Trans. Image Processing*, vol. 20, no. 3, pp. 625–640, 2011.

[38] R. Kondor, N. Teneva, and V. Garg, "Multiresolution matrix factorization," in *Proc. of the 31st International Conference on Machine Learning*, 2014, pp. 1620–1628.

[39] C. Rusu and B. Dumitrescu, "An initialization strategy for the dictionary learning problem," in *Proc. IEEE ICASSP*, 2014, pp. 6731–6735.

[40] ——, "Stagewise K-SVD to design efficient dictionaries for sparse representations," *IEEE Signal Processing Letters*, vol. 19, no. 10, pp. 631–634, 2012.

[41] W.-H. Chen, C. H. Smith, and S. C. Fralick, "A fast computational algorithm for the discrete cosine transform," *IEEE Trans. Communications*, vol. 25, no. 9, pp. 1004–1009, 1977.

[42] R. Rubinstein, M. Zibulevsky, and M. Elad, "Double sparsity: Learning sparse dictionaries for sparse signal approximation," *IEEE Trans. Sig. Process.*, vol. 58, no. 3, pp. 1553–1564, 2010.

[43] C. Rusu, "Fast design of efficient dictionaries for sparse representations," in *Proc. IEEE Machine Learning for Signal Processing*, 2012, pp. 1–5.