DiCoDiLe: Distributed Convolutional Dictionary Learning
Thomas Moreau, Alexandre Gramfort

To cite this version:
Thomas Moreau, Alexandre Gramfort. DiCoDiLe: Distributed Convolutional Dictionary Learning, 2019. hal-02371715

HAL Id: hal-02371715
https://hal.archives-ouvertes.fr/hal-02371715
Preprint submitted on 20 Nov 2019

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.
DiCoDiLe: Distributed Convolutional Dictionary Learning

Thomas Moreau and Alexandre Gramfort

Abstract—Convolutional dictionary learning (CDL) estimates shift invariant basis adapted to represent signals or images. CDL has proven useful for image denoising or inpainting, as well as for pattern discovery on multivariate signals. Contrarily to standard patch-based dictionary learning, patterns estimated by CDL can be positioned anywhere in signals or images. Optimization techniques consequently face the difficulty of working with extremely large inputs with millions of pixels or time samples. To address this optimization problem, we propose a distributed and asynchronous algorithm, employing locally greedy coordinate descent and a soft-locking mechanism that does not require a central server. Computation can be distributed on a number of workers which scales linearly with the size of the data. The parallel computation accelerates the parameter estimation and the distributed setting allows our algorithm be used with data that does not fit into a single computer’s RAM. Experiments confirm the theoretical scaling properties of the algorithm, allowing us to learn patterns on images from the Hubble Space Telescope containing tens of millions of pixels.

Index Terms—Convolutional Dictionary Learning, Distributed System, Coordinate Descent.

1 INTRODUCTION

Convolutional sparse linear models consider that signals or images are obtained by linear combination of a few patterns localized in time or space. Patterns, which typically have a finite support, encode local variations such as edges in images or transient events in signals. The benefit of this model over traditional sparse coding technique based on local patches is that it is shift-invariant. While sparse coding encodes every local patch in the data, the convolutional linear model selects and encodes few patches leading to sparser and sharper reconstruction [3]. Over the last ten to fifteen years, these models have been used successfully for various signal and image processing applications. It was first proposed by Grosse et al. [14] for music recognition. It was then used extensively for denoising and inpainting of images [5, 17, 22, 28]. Beyond denoising or inpainting, recent applications used this shift-invariant model as a way to learn and localize relevant patterns in signals or images. Yellin et al. [29] used it to count the number of blood-cells present in holographic lens-free images. Jas et al. [15] and Dupré la Tour et al. [10] learned recurring patterns in univariate and multivariate signals from neuroscience. This model has also been used for denoising astronomical data [26] and localizing predefined patterns from space satellite images [9].

Convolutional dictionary learning (CDL) is the technique commonly employed to estimate the parameters of such shift-invariant models. The dictionary is here formed by the set of patterns (or atoms), whose positions in time or space are encoded by some sparse activation vectors. CDL then boils down to an optimization problem, where the variables are the dictionary as well as the activations. This problem is non-convex and approximate solutions can be found by using alternating minimization over each variable. The estimation of the activation – which is commonly referred to as convolutional sparse coding (CSC) – is the critical bottleneck when it comes to applying CDL to large data.

As CSC is a convex problem, classical optimization tools can be employed. Chalasani et al. [6] used an accelerated proximal gradient method based on Fast Iterative Soft-Thresholding Algorithm (FISTA; [2]). Bristow et al. [5] proposed the Fast Convolutional Sparse Coding (FCSC) method, a splitting algorithm based on Alternating Direction Method of Multipliers (ADMM; [12]). While it is possible to leverage the Fast Fourier Transform to reduce the complexity of these approaches [28], both algorithms require to compute Fourier transforms of the entire data at each iteration. The cost can therefore be prohibitive for large signals or images. To avoid such costly global operations, several works have proposed to use local computations [14, 17, 19, 22, 31]. A first line of work proposed by Kavukcuoglu et al. [17] uses greedy coordinate descent (GCD; [21]) with an efficient mechanism to compute the update value. In their seminal work, Grosse et al. [14] propose to adapt the Feature Sign Search algorithm to only process small window of the signal at a time. Similarly, Papyan et al. [22] proposed to solve the problem locally over smaller continuous chunks of signal and then aggregate these local results by solving a global optimization problem. In our recent work [19], we proposed local greedy coordinate descent (LGCD). Based on updates similar to GCD, this method has a lower per-iteration complexity as it relies on local computation in the data. It does so like a patch-based technique would do, yet solving the non-local problem.

To further reduce the computational burden of CDL, recent studies consider parallelization strategies. Skau and Wohlberg [25] proposed a parallel method based on consensus ADMM. It does so by splitting the computation across the different atoms. This limits the number of cores that can be used to the number of atoms in the dictionary, and it still requires Fourier transforms on the entire data. Concurrently, we proposed an alternative parallel algorithm for CSC, called DICOD [19], that distributes GCD updates when working with one dimensional signals. DICOD could be used with multidimensional data such as images, yet only by splitting the data along one dimension, hence limiting its interest in this setting.

In the present paper, we propose a distributed solver for CDL, which can be used to learn recurring patterns in large multidimensional data. For the CSC step, we extend DICOD to use local updates with lower complexity while still working in the asynchronous setting without a central server. We detail a soft-lock mechanism, necessary to ensure the convergence of the algorithm with a grid of workers while remaining asyn-
chronous. Then, we explain how to also leverage the distributed set of workers to further reduce the cost of the dictionary update. This is done by precomputing in parallel sufficient statistics used in the gradient operator. Numerical experiments confirm the agreement between the theoretical complexity of our algorithm and its running time, allowing us to learn patterns from an astronomical image formed by tens of millions of pixels. Code is made available to replicate our results.

In Section 2, we describe the convolutional linear model and the estimation of its parameters with CDL. Our distributed algorithm DiCoDiLe is introduced in Section 3. Section 4 presents results on simulations, signals and images.

Notation For a vector \( U \in \mathbb{R}^p \) we denote \( U_p \in \mathbb{R} \) its \( p \)th coordinate. For a finite d-dimensional domain \( \Omega = \prod_{i=1}^d [0,T_i] \), where \( [0,T_i] \) are the integers from 0 to \( T_i - 1 \), \( |\Omega| = \prod_{i=1}^d T_i \) is its size. We will denote \( T = \prod_{i=1}^d T_i \) whenever the link between \( T \) and \( T_i \) is unambiguous. We denote \( X_0^p \) the space of observations defined on \( \Omega \) with values in \( \mathbb{R}^p \) (resp. \( \mathbb{R}^P \times Q \)). For instance, \( X_0^d \) with \( d = 2 \) is the space of RGB-images with height \( T_1 \) and width \( T_2 \). The value of an observation \( X \in X_0^p \) at position \( \omega \in \Omega \) is given by \( X[\omega] = X[\omega_1,\ldots,\omega_d] \in \mathbb{R}^p \), and its restriction to the \( p \)th coordinate at each position is denoted \( X_p \in X_0^d \). All signals are 0-padded, i.e. for \( \omega \not\in \Omega \), \( X[\omega] = 0 \). The convolution operator is denoted \(*\) and is defined for \( X \in X_0^p \) and \( Y \in X_0^p \times Q \) as the signal \( X * Y \in X_0^p \) with

\[
(X * Y)[\omega] = \sum_{\tau \in \Omega} X[\omega - \tau] \cdot Y[\tau], \quad \forall \omega \in \Omega . \tag{1}
\]

For any signal \( X \in X_0^d \), the reversed signal is defined as \( X^\tau[\omega] = X[T_1 - \omega_1,\ldots,T_2 - \omega_d] \). Computing the convolution \( X * Y \) is equivalent to computing the cross-correlation with \( X \). The \( p \)-norm is defined as \( \| X \|_p = (\sum_{\omega \in \Omega} \| X[\omega] \|_p)^{1/p} \). We denote \( F \) the Discrete Fourier transform (DFT) of a signal and for a multivariate signal \( X \in X_0^d \), \( \hat{X} = F(X) \) denotes the DFT of each of its coordinates. Finally, we will also denote \( \text{ST} \) the soft-thresholding operator defined as

\[
\text{ST}(u,\lambda) = \text{sign}(u) \max(|u| - \lambda, 0) .
\]

2 Convolutional Dictionary Learning

In this section, we recall the convolutional linear model and the classical way to estimate its parameters by alternating minimization.

2.1 Convolutional Linear Model

For an observed \( X \in X_0^d \), signal or image, the convolutional linear model reads

\[
X = Z * D + \xi , \tag{2}
\]

with \( D \in X_0^K \times P \) a dictionary of \( K \) atoms \( D_k \) in dimension \( P \) defined on a sub-domain \( \Theta = \prod_{i=1}^d [0,L_i] \subset \Omega \), also called support of the atom. The activations are denoted \( Z \in X_0^d \) and \( \xi \in X_0^d \) is an additive noise term which is typically considered white and Gaussian. The dictionary elements \( D_k \) are prototypical patterns present in the data. They typically have a much smaller support \( \Theta \) than the full domain \( \Omega \) (i.e. \( L = |\Theta| \ll T \)). The activations \( Z_k \) encode the localization of the pattern \( k \) in the data. A non-zero coefficient \( Z_k[\omega] \) indicates the presence of the pattern \( D_k \) around the position \( \omega \). As patterns typically occur at few locations in each observation, the coding signal \( Z \) is considered to be sparse. The aim of this model is truly to separate the shape of the local variation in the observation – encoded in the dictionary – from their localization in space or time. Figure 1 illustrates the decomposition of a temporal signal with one pattern.

The parameters of this model are estimated by solving the following CDL optimization problem:

\[
\min_{Z,D} \frac{1}{2} \| X - Z * D \|_2^2 + \lambda \| Z \|_1 , \quad \Omega \subset \mathbb{R}^d , \quad D \in \mathbb{R}^{K \times P} , \quad \lambda > 0 . \tag{3}
\]

where the first term \( F(Z,D) \) measures how well the model fits the data and the second term \( G(Z) \) promotes sparse activations. The regularization parameter \( \lambda > 0 \) balances these two terms. The constraint \( \| D_k \|_2 \leq 1 \) is used to avoid the scale ambiguity in our model due to the \( \ell_1 \) constraint. Indeed, the \( \ell_1 \)-norm of \( Z \) can be made arbitrarily small by rescaling \( D \) by \( \alpha \in \mathbb{R}^d \) and \( Z \) by \( \frac{1}{\alpha} \). This problem is bi-convex, i.e. it is not jointly convex in \((Z,D)\) but it is convex in each of its coordinates. It is usually solved with an alternating minimization algorithm, updating at each iteration one of the block of coordinates, \( Z \) or \( D \). As each of the sub-problems are convex, the alternated descent is guaranteed to decrease the cost function, converging to a critical point of (3). It is however not guaranteed to be a global minimizer.

2.2 Convolutional Sparse Coding (CSC)

Given a dictionary of patterns \( D \), CSC aims to retrieve the sparse decomposition \( Z^* \) associated to the signal \( X \). When minimizing only over \( Z \), (3) becomes

\[
Z^* = \underset{Z}{\text{argmin}} \frac{1}{2} \| X - Z * D \|_2^2 + \lambda \| Z \|_1 . \tag{4}
\]

This boils down to a Lasso problem with a Toeplitz design matrix. Therefore, classical optimization techniques for the Lasso can be applied with the same convergence guarantees.

In practice, these methods are adapted to leverage the convolutional structure of the problem, reducing the computational complexity to solve (4).

2.2.1 Locally Greedy Coordinate Descent (LGCD)

Coordinate descent (CD) is an algorithm which updates a single coefficient at each iteration. Following Moreau et al. [19], we use locally greedy coordinate selection to solve efficiently the CSC (4). Algorithm 1 summarizes the computations for LGCD.

Coordinate update For (4), it is possible to compute efficiently the optimal update for a given coordinate when all the others are fixed. Denoting \( q \) the iteration number, and \( Z_{q+1}^0 \) the iteration number, and \( Z_{q+1} \) (atom

![Fig. 1. Decomposition of a noiseless univariate signal X (blue) as the convolution Z*D between a temporal pattern D (orange) and a sparse activation signal Z (green).](image-url)
k_0 at position \( \omega_0 \) the optimal value of the activation \( Z_{k_0}^{(q)}[\omega_0] \), the update reads
\[
\hat{Z}_{k_0}^{(q)}[\omega_0] = \frac{1}{\|D_{k_0}^{(q)}\|^2} \text{ST}(\beta_{k_0}^{(q)}[\omega_0], \lambda), \tag{5}
\]
where \( \beta_{k_0}^{(q)} \) is an auxiliary variable in \( \mathcal{X}_0^K \) defined for \((k, \omega) \in [1, K] \times \Omega\) as
\[
\beta_{k_0}^{(q)}[\omega] = \left( (X - Z^{(q)} * D + Z_k^{(q)}[\omega]c_\omega * D_k^*) \right)[\omega], \tag{6}
\]
and where \( c_\omega \) is a dirac (canonical vector) with value 1 in \( \omega \) and 0 elsewhere. The value of the auxiliary variable in \((k_0, \omega_0)\) corresponds to the gradient of \( F \) for a point \( Z^{(q)} \) with the coordinate \((k_0, \omega_0)\) set to 0. To enable efficient coordinate update, an important property is that \( \hat{Z}_{k_0}^{(q)}[\omega] \) only depends on a small neighborhood around \( \omega \).

**Definition 2.1** (\( \Theta \)-neighbourhood). Let us consider a support \( \Theta = \prod_{i=1}^d \{0, L_i\} \). For \( \Omega_0 \in \Theta \), the \( \Theta \)-neighbourhood \( V_{\Theta_0}(\omega_0) \) is defined as
\[
V_{\Theta_0}(\omega_0) = \bigcup_{i=1}^d \{0, L_i\} \cup \{0, \omega_0, 1, \omega_0 + L_i\}.
\]

**Proposition 2.2** (Weak dependence). If the coordinate \((k_0, \omega_0)\) is updated in \( Z^{(q)} \), then the value of \( Z^{(q+1)} \) differs from \( Z^{(q)} \) only for \( \omega \) in a \( \Theta \)-neighbourhood \( V_{\Theta_0}(\omega_0) \).

**Proof.** If the coordinate \( Z_{k_0}[\omega_0] \) is updated with an additive update \( \Delta Z_{k_0}[\omega_0] \), then the value of \( Z^{(q+1)} \) differs from \( Z^{(q)} \) only for \( \omega \) in a \( \Theta \)-neighbourhood \( V_{\Theta_0}(\omega_0) \), Kavukcuoglu et al. [17] showed that \( \beta_{k_0}^{(q+1)} \) obtained using the relation
\[
\beta_{k_0}^{(q+1)}[\omega] = \beta_{k_0}^{(q)}[\omega] - (D_{k_0} * D_k^*)[\omega_0] \Delta Z_{k_0}^{(q)}[\omega_0], \tag{7}
\]
for all \((k, \omega) \neq (k_0, \omega_0)\). As the support of \((D_{k_0} * D_k^*)[\omega] \) is \( V_{\Theta_0}(0) \), this implies that the value \( \beta_{k_0}^{(q+1)} \) only changes in the neighborhood \( V_{\Theta_0}(\omega_0) = \omega_0 + V_{\Theta_0}(0) \).

This proposition shows that the optimal value for a coordinate only depends on the current values of the activations in a neighborhood of size \( |V_{\Theta_0}(0)| = 2^d L \). This corresponds to twice the size of the dictionary in each dimension. Using this property, the computational complexity of updating a coordinate is \( O(2^d KL) \), as it is only necessary to update \( \beta \) and \( \hat{Z}^{(q)} \) on \( V_{\Theta_0}(\omega_0) \).

**Coordinate selection** The selection of the updated coordinate \((k_0^{(q)}, \omega_0^{(q)})\) can follow different strategies. Following our work in [10, 19], we propose to select the coordinates with a locally greedy rule (LGCD).

**Definition 2.3** (Disjoint and contiguous partition). Let us consider a domain \( \Omega \). For \( M \in \mathbb{N} \), a \( M \) disjoint and contiguous partition \( \left\{ C_m \right\}_{m=1}^M \) is a set of subsets \( \Omega \) such that
\[
\bigcup_{i=1}^d \{0, \omega_i\} \subset C_m \quad \text{for} \quad \omega, \omega' \in C_m, \\
\bigcup_m C_m = \Omega, \quad C_m \cap C_{m'} = \emptyset \quad \text{for} \quad m \neq m'.
\]

At iteration \( q \), the coordinate to update is selected greedily among the \( m \)-th sub-domain \( C_m \), with
\[
(k_0^{(q)}, \omega_0^{(q)}) = \arg \max_{(k, \omega) \in [1, K] \times C_m} \left| \frac{\hat{Z}_{k_0}^{(q)}[\omega] - Z_k^{(q)}[\omega]}{\Delta Z_k^{(q)}[\omega]} \right|, \tag{8}
\]
with \( m = q \mod M \). This strategy cycles through each of the \( M \) sub-domains and select the coordinate in \( C_m \) which is the farthest away from its optimal value when all the other coordinates are fixed. The quantity \( \Delta Z_k[\omega] \) acts as a proxy for the reduction of the cost function obtained with the update of coordinate \((k, \omega)\) to the value \( \hat{Z}_{k_0}^{(q)}[\omega] \) [21]. The complexity of the locally greedy selection rule is linear with the size of the sub-segments \( O(K|C_m|) \). In the case where \( M = KT \), this rule boils down to the cyclic CD [11] with a coordinate selection complexity in \( O(1) \), while setting \( M = 1 \) boils down to greedy CD [21] where the complexity of the selection is linear with the size of the signal \( O(KT) \). The selection of \( M \) in LGCD acts as a tradeoff between the computational complexity and the convergence speed of these two methods.

As it was shown in recent studies [16, 20], updating in priority the important coordinates by greedy selection leads to faster convergence rates, although it increases the cost of one iteration. The LGCD strategy proposes to use sub-domains \( C_m \) such that the computational complexities of selecting and updating the coordinate match. By using sub-domains of the \( V_{\Theta_0}(0) \)’s size \( |C_m| = 2^d L \), the global iteration complexity of LGCD scales with \( 2^d \) times the size of the dictionary \( D \) i.e. \( O(2^d KL) \). By doing so LGCD achieves faster convergence rates than non-greedy coordinate selection rule while limiting the per-iteration cost.

**Stopping criterion** The stopping criterion used for this algorithm follows [21]. The algorithm is stopped once the magnitude of all possible coordinate updates are below a threshold \( \varepsilon > 0 \) specified by the user, i.e.
\[
\max_{(k, \omega) \in [1, K] \times \Omega} |\Delta Z_k^{(q)}[\omega]| < \varepsilon. \tag{9}
\]

**2.2.2 Global solvers**

Chalasani et al. [6] propose to solve (4) by proximal gradient methods exploiting the convolution in the gradient computation to save time. The core idea behind this method is to use a proximal splitting of the cost function between its smooth part \( F(\cdot, D) \) and a proximable part \( G \). Proximal gradient descent (ISTA) reads
\[
Z^{(q+1)} = \text{prox}_{\gamma G} \left( Z^{(q+1)} - \gamma \nabla_F F(Z^{(q+1)}, D) \right), \tag{10}
\]
where \( \gamma \) is the step size of the algorithm, usually chosen as the inverse of the smoothness constant of \( F \), and \( \text{prox}_{\gamma G} \) is the proximal operator of \( G \). For \( G(\cdot) = \lambda \cdot \|\cdot\|_1 \), this corresponds to the soft-thresholding operator \( \text{ST}(\cdot, \gamma \lambda) \). To make this algorithm
efficient the key is to rely on the FFT for the gradient computations, as described in [28]. Doing the computation in the Fourier domain, the computational complexity of each iteration of this algorithm is $O(KT \log T)$. It is also possible here to use an accelerated version FISTA to solve the problem [2, 6].

An alternative global algorithm proposed by Bristow et al. [5] is based on the ADMM algorithm [12]. The core idea in this line of work is to introduce an auxiliary variable and split the two part of the objective with an equivalent problem

$$\min_{\tilde{Y}, Z} \frac{1}{2} \| \tilde{X} - \tilde{Y} D \|^2_F + \lambda \| Z \|_1, \text{ s.t. } F(Z) = \tilde{Y}. \quad (11)$$

This problem is then solved using ADMM in its scaled form [4, Section 3.1.1]. The critical part in this algorithm is the minimization of the augmented Lagrangian relatively to Section 3.1.1. This problem is then solved using ADMM in its scaled form [4, 28]. However, ADMM algorithms are slow to reach a convergence, yielding sub-optimal reconstruction performances as the feasible point is not directly optimized but results from a consensus. Yellin et al. [29] proposed to adapt the K-SVD dictionary update method [1] to the convolutional setting but this technique is adapted to an $\ell_0$ penalty for the activation $Z$. It also requires to compute costly singular value decompositions (SVD).

2.3 Dictionary Update

Given an activation signal $Z^{(q)}$, the dictionary is updated to improve how the model reconstructs the signal. The problem reads

$$D^* = \arg \min_{\|D\|_2 \leq 1} \frac{1}{2} \| X - Z^{(q)} \ast D \|^2_2. \quad (12)$$

This problem is smooth and convex and can be solved using algorithms designed for smooth constrained optimization.

2.3.1 Projected Gradient Descent

The projected gradient descent (PGD; [7]) and its accelerated version (APGD) are possible strategies. These two algorithms are equivalent to ISTA and FISTA iterations where the proximal operator is a simple projection [7]. The efficiency of these two methods relies on pre-computations of sufficient statistics for fast evaluation of the gradient of the objective $\nabla_D F$ [10]. The computations can be factorized as

$$\nabla_D F(Z, D) = Z^\dagger (X - Z \ast D) = \Psi - \Phi \ast D, \quad (13)$$

where $\Phi \in \mathcal{A}^{K \times K}$ and $\Psi \in \mathcal{A}^{K \times P}$ are respectively defined as the convolution $Z^\dagger \ast Z$ restricted to $\Theta$ and the convolution $Z^\dagger \ast X$ restricted to $\Theta$, i.e.

$$\Phi[\omega] = Z^\dagger \ast Z = \sum_{\tau \in \Theta} Z[\omega + \tau] Z^\dagger[\tau] \quad \text{for } \omega \in \Theta, \quad (14)$$

$$\Psi[\omega] = Z^\dagger \ast X = \sum_{\tau \in \Theta} Z[\omega + \tau] X^\dagger[\tau] \quad \text{for } \omega \in \Theta. \quad (15)$$

The complexity to obtain the two constants $\Phi$ and $\Psi$ scales as $O(K(K + P)L T)$. This is comparable with the cost of computing the gradient directly $O(KPT \log T)$. However, once the two constants are pre-computed, the gradient of $F$ is obtained using (13) with complexity $O(2^2 K^2 P L \log(L))$. This is independent of the data size $T$, making each iteration efficient.

2.3.2 Related solvers

The block coordinate descent for dictionary learning [18], which consists in updating one atom at a time, can also be extended to the convolutional case. As PGD for full dictionary updates, this technique benefits from sufficient statistics pre-computations and the complexity is similar. Another possibility is to rely on ADMM to update the dictionary [5, 28]. However, ADMM algorithms can be slow to reach a convergence, yielding sub-optimal reconstruction performances as the feasible point is not directly optimized but results from a consensus. Yellin et al. [29] proposed to adapt the K-SVD dictionary update method [1] to the convolutional setting but this technique is adapted to an $\ell_0$ penalty for the activation $Z$. It also requires to compute costly singular value decompositions (SVD).

2.4 Choice of initialization and regularization parameter

In practice, the quality of the solution obtained by CDL is impacted by the choice of the regularization parameter $\lambda$. Also, due to the iterative nature of the solvers, the initialization of the dictionary $D^0$ can play an important role.

**Regularization parameter selection**

The choice of a regularization parameter $\lambda$ in the context of dictionary learning is often selected by cross-validation on a subsequent task such as denoising or classification. However, in the context of purely unsupervised learning, e.g. matrix factorization, this is still an open problem. This problem has recently received some attention using Bayesian methods [23, 27]. For the specific case of CDL using an $\ell_1$ norm, it is useful to note that there exists a value $\lambda_{\max}$ such that for $\lambda \geq \lambda_{\max}$, 0 is solution of (4). The first order condition for (4) reads

$$\nabla_Z F(0, D^{(0)}) = X \ast D^{(0)} \in \partial G(0) = [-\lambda, \lambda]^{KT}. \quad (16)$$

This condition is always verified if $\lambda > \lambda_{\max} = \| X \ast D^{(0)} \|_\infty$. In the following, the regularization parameter $\lambda$ is chosen as a fraction of $\lambda_{\max}$. Using this data dependent value, the ratio $\lambda / \lambda_{\max}$ is chosen in $[0, 1]$, making the definition of the regularization parameter common to all input signals $X$.

**Dictionary initialization**

The standard strategy to initialize $D^{(0)}$ for CDL methods is to generate random atoms with Gaussian white noise. However, as these atoms generally poorly correlate with the data, the initial value of $\lambda_{\max}$ is low compared to the following ones, leading to dense activation $Z$. For dictionary learning, a classical way to initialize the dictionary is to take samples from the input data [18]. In the convolutional setting, this boils down to taking random patches of the size of the dictionary directly from the data. This can be theoretically motivated by the work of Zhang et al. [30] that demonstrate improvements in pattern recovery under suitable assumptions.

3 Distributed Convolutional Dictionary Learning (DiCoDiLe)

In this section, we introduce DiCoDiLe, a distributed algorithm for convolutional sparse coding. Its steps are summarized in Algorithm 2. The CSC part of this algorithm will be referred to as DiCoDiLe$D$. It extends our previous work DICOD [19] by handling multidimensional data, in particular images. The algorithm proposes to distribute the computation on a grid of workers. To guarantee convergence with multidimensional data in an asynchronous setting, we introduce the notion of
Algorithm 2 DiCoDiLe with $W$ workers

Require: $D^{(0)}$, $X$, stopping criterion $\nu$.
1: repeat
2: Compute $Z^{(q+1)}$ with DiCoDiLe$(X, D^{(0)}, W)$.
3: Compute $\phi$ and $\psi$ with (14) and $W$ workers.
4: Update $D^{(q+1)}$ with PGD and Armijo backtracking line-search.
5: until Cost variation is smaller than $\nu$

soft-locks described in Subsection 3.1. Details on how the dictionary updates can also benefit from the distributed setting are presented in Subsection 3.2.

3.1 DiCoDiLe$_Z$: Distributed CSC with coordinate descent

In the convolutional setting, coordinates that are far enough – relatively to the size of the dictionary – are only weakly dependent (Proposition 2.2). It is thus natural to distribute the computation of CD by splitting the data in continuous sub-domains [19].

The DiCoDiLe$_Z$ algorithm, described in Algorithm 3, is distributed over $W$ workers which update asynchronously the coordinates of $Z$. We consider $\{S_\omega \}_{\omega=1}^W$, a disjoint and contiguous partition of $\Omega$. Each worker $w \in [1, W]$ is in charge of updating the coordinates of the sub-domain $S_w$. These sub-domains are also partitioned with disjoint and contiguous sub-domains $C^\omega_m$ of size $|V_\omega(0)|$. Then, the worker $w$ chooses at each iteration $q$ an update candidate $(k_0, \omega_0)$ which corresponds to a greedy update in one of its sub-domain $C^\omega_m$, i.e.

$$(k_0, \omega_0) = \arg\max_{(k, \omega) \in [1, K] \times C^\omega_m} |\Delta Z^\omega[k]|,$$  \hspace{1cm} (17)

with $m = q \mod M$. The sub-domains $C^\omega_m$ are selected in a cyclic order but empirical test showed no significant differences for random sub-domain selection. Then, this candidate update is accepted or rejected using the soft-lock mechanism described in Subsection 3.1.1. If the candidate is rejected, the worker moves to the next sub-domain $C^\omega_{m+1}$ without updating a coordinate. If the candidate is accepted, the value of the coordinate $Z^\omega_{k_0} [\omega_0]$ is replaced by $Z^\omega_{k_0} [\omega_0]$ and $\beta$ is updated with (7). To ensure the convergence, it is sufficient to notify the neighboring workers if a local update changes the value of $\beta$ outside of $S_w$, i.e. if $V_\beta(\omega_0) \not\subset S_w$.

Definition 3.1 ($\Theta$-Border). Let us consider a sub-domain, $S_w = \prod_{i=1}^d [l_i, u_i]$, where $(l_i, u_i)$ are the lower and upper limits of this sub-domain for dimension $i$. For $\Theta = \prod_{i=1}^d [0, L_i]$, the $\Theta$-border is defined as the set of points $\omega \in S_w$ such that $V_\Theta(\omega) \not\subset S_w$, i.e.

$$B_\Theta(S_w) = \prod_{i=1}^d [l_i, l_i + L_i \cup u_i - L_i, u_i].$$  \hspace{1cm} (18)

In the case where the updated coordinate $(k, \omega_0)$ is in $S_w \setminus B_\Theta(S_w)$, then $V_\Theta(\omega_0)$ is included in the set of coordinates $S_w$ updated by the worker $w$. In this case, the update is not at the border of the sub-domain, so the values of $\beta$ and $Z^{\omega_0}$ need no update for the other workers $w'$. Thus, there is no need to exchange information between the workers. However, for $\omega_0 \in B_\Theta(S_w)$, it is necessary to notify other workers $w'$ whose domain overlap with the $\Theta$-neighborhood of $\omega_0$ i.e. $V_\Theta(\omega_0) \cap S_w' \neq \emptyset$. The notification can be done by sending the triplet $(k_0, \omega_0, \Delta Z_{k_0}[\omega_0])$ to these workers. These information are sufficient to keep $\beta$ up to date for all workers using (7). Figure 2 illustrates this communication process on a 2D example.

With this distributed algorithm, the inter-processes communications are minimal, as only $d + 2$ values need to be exchanged. Moreover, as it does not rely on centralized communication, the necessary bandwidth between the workers is also reduced as a worker only communicates with its direct neighbors. This contrasts with other distributed algorithms for CDL such as [25] that requires centralized communication of full support values between the workers. Finally, the communication only occurs when $\omega_0 \in B_\Theta(S_w)$, which happens rarely especially when working with large data and patterns of small support. This reduces even further the need of bandwidth between workers.

3.1.1 Interferences and Soft-Locks in asynchronous setting

While it now clear how communication should be done between workers, one need to clarify how to proceed to ensure convergence in the case of asynchronous updates. When $W$ coordinates $(k_w, \omega_w)_{w=1}^W$ of $Z$ are updated simultaneously by respectively $\Delta Z_{k_w} [\omega_w]$, the updates might not be independent. The local version of $\beta$ used for the update does not account for the other updates. The cost difference resulting from all these simultaneous updates is denoted $\Delta E$, and the cost reduction induced by only one update $w$ is denoted $\Delta E_{k_w} [\omega_w]$. Simple derivations, detailed in Proposition A.2, show that

$$\Delta E \leq \sum_{w=1}^W \Delta E_{k_w} [\omega_w] \hspace{1cm} \text{(19)}$$

if for all $w$, all other updates $w'$ are such that if $\omega_{w'} \not\in V_\Theta(\omega_w)$, then $(D_{k_w} + D_{k_{w'}})[\omega_{w'} - \omega_w] \Delta Z_{k_w} [\omega_w] \Delta Z_{k_{w'}} [\omega_{w'}] = 0$.

If for all $w$, all other updates $w'$ are such that if $\omega_{w'} \not\in V_\Theta(\omega_w)$, then $(D_{k_w} + D_{k_{w'}})[\omega_{w'} - \omega_w] \Delta Z_{k_w} [\omega_w] \Delta Z_{k_{w'}} [\omega_{w'}] = 0$. The updates can be considered to be sequential as the interference term is zero. This also directly results from Proposition 2.2. When $|\{\omega_w\} \cap V_\Theta(\omega_w)| = I_0 > 1$, the interference term does not vanish. Moreau et al. [19] show that if $I_0 < 3$, then, under mild assumption, the interference term can be controlled and DICOD converges. While this is sufficient for 1D partitioning as $I_0$ cannot be larger than 2 (if $S_w$ is larger than $V_\Theta(0)$), this analysis cannot be extended to $I_0 \geq 3$. This limits the partitioning of $\Omega$ along one single dimension.
To avoid this limitation, we propose the soft-lock, a novel mechanism to avoid interfering updates of higher order. The idea is to avoid updating concurrent coordinates simultaneously. A classical tool in computer science to address such concurrency issue is to rely on a synchronization primitive, called lock. This primitive can ensure that at most one worker enters a protected block of code simultaneously. In our case, it would be possible to use this primitive to restrict updates in $B_0(S_w)$ only when none of its neighbors is updating $B_0(S_w')$. This would avoid interfering updates. However, the drawback of this method is that it prevents the algorithm to be asynchronous, as typical lock implementations rely on centralized communications. Instead, we propose to use an implicit locking mechanism which keeps our algorithm asynchronous.

**Definition 3.2 (θ-extension).** We consider $S_w = \prod_{i=1}^{d} [l_i, u_i]$ a sub-domain of $Ω$. For $θ = \prod_{i=1}^{d} [0, l_i]$, theθ-extension of $S_w$ is defined as $E_θ(S_w) = (S_w + V_θ(0)) \setminus S_w$, i.e.

$$E_θ(S_w) = \prod_{i=1}^{d} [l_i - l_i, u_i + l_i].$$

The idea of the soft-lock is the following. Additionally to the sub-domain $S_w$, each worker also maintains the value of $Z^{(w)}$ and $β$ on the θ-extension $E_θ(S_w)$ (See Figure 2). At each iteration, the worker $w$ considers a candidate coordinate $(k_0, ω_0)$ to update from (17). If $ω_0 \not\in B_0(S_w)$, the coordinate is updated like in LGCD. When $ω_0 \in B_0(S_w)$, the candidate is accepted if there is no better coordinate update in $V_θ(ω_0) \cap E_θ(S_w)$, i.e.

$$|∆Z_{k_0}[ω_0]| > \max_{k, ω \in [1, K] \times V_θ(ω) \cap E_θ(S_w)} |∆Z_{k}[ω]|.$$  

In case of equality, the update in the sub-domain $S_w'$ with the minimal index $w'$ is preferred. Using this mechanism effectively prevents any interfering update. If two workers $w < w'$ have update candidates $ω_0$ and $ω'_0$ such that $ω'_0 \in V_θ(ω_0)$, then with (21), only one of the two updates will be accepted. It will be either the largest one if $|∆Z_{k_0}[ω'|] ≠ |∆Z_{k_0}[ω_0]|$ or the one from $w$ if both updates have the same magnitude. The other update will be rejected and there will be no interference. By doing so this mechanism is fully asynchronous. The updates on the borders are implicitly ordered, preventing interference. The update are guaranteed to be performed as if they were run sequentially, ensuring convergence of the algorithm. This requires to increase slightly the communication between the workers and to double the size of the θ-border in each direction. This is however not a bottleneck as demonstrated in the following experiments.

3.1.2 Speed-up analysis of DiCoDiLe$^2$

In this section, we consider the speed-up provided by this distributed algorithm. As each worker runs LGCD locally, the computational complexity of a local iteration in a worker $w$ is $O(2^d K L)$. This complexity is independent of the number of workers used to run the algorithm. Thus, the speed-up obtained is equal to the number of updates that are performed in parallel. If we consider $W$ update candidates $(k_0, ω_0)$ chosen by the workers, the number of updates that are performed corresponds to the number of updates that are not soft-locked. When the updates are uniformly distributed on $S_w$, the probability that an update candidate located in $ω \in S_w$ is not soft-locked can be lower bounded by

$$P(ω \not\in SL) ≥ \prod_{i=1}^{d} \left(1 - \frac{W_i}{L_i} \right).$$

Computation details are provided in Proposition B.1. When the ratios $\frac{W_i}{L_i}$ are large for all $i$, i.e. when the size of the workers sub-domains $|S_w|$ are large compared to the dictionary size $L_i$, then $P(ω \not\in SL) ≃ 1$. In this case, the expected number of accepted update candidates is close to $W$ and DiCoDiLe$^2$ scales almost linearly. When $W_i$ reaches $\frac{L_i}{\epsilon_i} (2^\epsilon_i d)$, then $P(ω \not\in SL) ≥ \frac{1}{2}$, and the acceleration is reduced to $\frac{1}{2}$.

In comparison, we report in [19] a super-linear speed-up for DICOD on 1D signals. This is due to the fact that we are using GCD locally in each worker. GCD has a very high iteration complexity when $W$ is small, as it scales linearly with the size of the considered sub-domain. Thus, when sub-dividing the signal between more workers, the iteration complexity is decreasing and more updates are performed in parallel. This explains why the speed-up is almost quadratic for low $W$. As the complexity of iterative GCD is worse than LGCD, DiCoDiLe$^2$ has better performance than DICOD in low $W$ regime. Furthermore, in the 1D setting, DICOD becomes similar to DiCoDiLe$^2$ once the size of the workers sub-domain becomes too small. This happens when the local domain of LGCD becomes comparable to the size of the sub-domains $S_w$. This implies that DiCoDiLe$^2$ always outperforms DICOD.

3.1.3 Stopping criterion

In this distributed setting, it is non-trivial to know when the algorithm reaches convergence. To match the sequential stopping criterion of GCD (9), the convergence is considered to be reached once no worker can perform a coordinate update $|ΔZ^{(w)}|$ that is higher than $ε > 0$. However, as a worker can be affected by its neighboring workers, the stopping criterion of CD cannot be applied independently in each worker. Once a worker reaches this state, the potential update can still increase above $ε$ because of updates in $E_θ(S_w)$ by another worker. To make sure the correct criterion is achieved, workers that...
reach the local criterion are paused. They are restarted when a neighboring workers triggers an update on \( E_o(S_w) \). The global convergence can be decided when all workers are paused. While this process is based on centralized communication, it can still be performed asynchronously using message passing to a designated master process that will notify the end of the algorithm to all workers.

### 3.2 Dictionary updates in a distributed setting

For the dictionary, a change in a coordinate in \( Z \) at any point in \( \Omega \) can impact the solution of (12). Moreover, the atoms are interdependent of each other. It is thus complicated to parallelize the dictionary update as it is centralized by nature and there is no natural division of the computation. However, the size of \( D \) is way smaller than \( Z \) and (12) can be solved efficiently when applying the PGD algorithm described in Subsection 2.3. The main computational bottleneck is the computation of the constants \( \Phi \) and \( \Psi \). For these, it is possible to leverage the distributed setting as the computation are separable between the different workers. The computations can be made embarrassingly parallel among the workers as

\[
\Phi[w] = W \sum_{w=1}^{W} \sum_{\tau \in S_w} Z[\tau]Z^T[\tau + \omega] \quad \text{for } \omega \in V_o(0) ,
\]

\[
\Psi[w] = W \sum_{w=1}^{W} \sum_{\tau \in S_w} Z[\tau]X^T[\tau + \omega] \quad \text{for } \omega \in \Theta .
\]

For all \((\omega, \tau) \in V_o(0) \times S_w\), we have \( \omega + \tau \in S_w \cup E_o(S_w) \). Thus, the computations of the inner sum \( \Phi[w] \) can be made independently by each worker. These partial results are gathered and reduced with \( \Phi = \sum_{w=1}^{W} \Phi[w] \). By making use of these distributed computation, \( \Phi \) and \( \Psi \) can be computed with respectively \( O(K^2|S_w| \log(|S_w \cup E_o(S_w)|)) \) and \( O(KP|S_w| \log(|S_w \cup E_o(S_w)|)) \) operations. The values of \( \Psi \) and \( \Phi \) are sufficient statistics to compute \( \nabla_D F \) and \( F \). The dictionary can thus be updated without having to gather the full activation \( Z \) on one machine. This allows to learn a dictionary for a signal \( X \) that does not fit in memory, as each worker only needs to store a part of the full signal and the dictionary updates only require storing these sufficient statistics.

### 4 Numerical Experiments

The numerical experiments are performed on a SLURM cluster with 30 nodes. Each node has 20 physical cores, 40 logical cores and 250 GB of RAM. A worker is typically one logical core on one of the machine. In practice we use up to 400 workers. DiCoDiLe is implemented in Python [24] using mpi4py [8].

Experiments make use of simulated one dimensional signals, generated following the sparse convolutional linear model (2) with \( d = 1 \) and \( P = 7 \). The dictionary is composed of \( K = 25 \) atoms \( D_k \) of length \( L = 250 \). Each atom is sampled from a standard Gaussian distribution and then normalized. The sparse code entries are drawn from a Bernoulli-Gaussian distribution, with Bernoulli parameter \( \rho = 0.007 \), mean 0 and standard variation 10. The noise term \( \xi \) is sampled from a standard Gaussian distribution with variance 1. The length of the signals \( X \) is denoted \( T \).

For experiments with images, we will use the standard colored Mandrill image \( P = 3 \) at full resolution \((512 \times 512)\). Finally, for large scale experiments, we used an image of the Great Observatories Origins Deep Survey (GOODS) South field, acquired by the Hubble Space Telescope [13]. We used the STScI-H-2016-39 image with resolution \( 6000 \times 3600 \).

In all experiments, we used \( \lambda = 0.1\lambda_{\text{max}} \) as a default regularization parameter, unless specified otherwise.

### 4.1 Performance of distributed sparse coding

The performances of the distributed sparse coding step in DiCoDiLe is impacted by the choice of coordinate descent strategy, as well as soft-lock mechanism and the data partitioning scheme. The following experiments discuss their effects on the scalability on DiCoDiLez.

#### 4.1.1 Coordinate Selection Strategy

Using one worker, Figure 3 compares the average running times of three CD strategies to solve (4): Cyclic, Greedy and LGCD. LGCD consistently outperforms the other strategies for both \( T = 150L \) and \( T = 750L \). For LGCD and Cyclic, the iteration complexity is constant compared to the size of the signal, whereas the complexity of each Greedy iteration scales linearly with \( T \). This makes running time of Greedy explode when \( T \) grows. Besides, as LGCD also prioritizes the more important coordinates, it is more efficient than Cyclic that selects the coordinate to update independently of the data. This explains the performances observed in Figure 3 and justifies the use of LGCD as the base of the DiCoDiLez distributed algorithm. In the following, we will not include Cyclic coordinate selection in our distributed algorithm as it is less efficient. We include Greedy as its complexity depends on the size of the signal, meaning that it can be competitive when the workload is distributed on many workers.

#### 4.1.2 Need for soft-lock mechanism

The reconstruction of the Mandrill with and without soft-locks is shown in Figure 4. It uses a dictionary \( D \) consisting of \( K = 25 \) patches of size \( 16 \times 16 \) extracted randomly from the original image. The reconstructed image \( Z * D \) obtained without soft-lock shows artifacts around the edges of some of the sub-domains \( S_w \) as the activations diverge in these locations. This visually demonstrates the need for controlling the interfering updates when more than two workers can interfere. When using the soft-lock mechanism, the algorithm returns the expected result.

#### 4.1.3 Impact of data partitioning

To demonstrate the impact of the domain partitioning strategy on images, Figure 5 shows the average runtime for DiCoDiLez

---


2. The image is publicly available on Hubble website: www.hubblesite.org/image/3920/news/58-hubble-ultra-deep-field
Scaling for 1D signals

Figure 6 investigates the role of CD strategies in the distributed setting with 1D signals. The average runtime of DICOD (with GCD on each worker), and DiCoDiLeZ (with LGCD) are reported in Figure 6. DiCoDiLeZ with LGCD scales sub-linearly with the number of workers used, whereas DICOD scales almost quadratically. However, DiCoDiLeZ outperforms DICOD consistently, as the performance of GCD is poor for low number of workers using large sub-domains. The two coordinate selection strategies become equivalent when $W$ reaches $T/4L$, as in this case DiCoDiLe has only one sub-domain $\Omega_{i}$. This is consistent for larger signals. Indeed, in the case where $T = 750$, DiCoDiLeZ scales linearly with the number of workers. We do not see the drop in performance, as in this experiments $W$ does not reach the scaling limit $T/4L$. Importantly, DiCoDiLeZ still outperforms DICOD for all regimes.

4.2 Comparison with Skau and Wohlberg [25]

Figure 8 displays the evolution of the objective function (3) as a function of time for DiCoDiLe and the Consensus ADMM algorithm proposed by Skau and Wohlberg [25]. To the best of our knowledge, the work of Skau and Wohlberg [25] is the only approach in the literature that proposes a parallel algorithm for CDL. Both algorithms were run on a single node with 36 workers. We used the Hubble Space Telescope GOODS South image as input data. As the algorithm by Skau and Wohlberg [25] raised a memory error with the full image on

We used W from an image of the GOODS South field, acquired by the Hubble Space Telescope [13]. We used LGCD GCD algorithms were run 5 times with 36 workers on a random-patch of size 512 × 512 taken from the Hubble Space Telescope GOODS South image. The solid lines denote the median curves obtained through interpolation.

This unsupervised algorithm is able to highlight structured patterns in the image, such as small stars. Patterns 1 and 7 are very fuzzy. This is probably due to the presence of very large object in the foreground. As this algorithm is not scale invariant, the objects larger than our patterns are encoded using these low frequency atoms.

5 Conclusion
This work introduces DiCoDiLe, an asynchronous distributed algorithm to apply convolutional dictionary learning to very large multidimensional data such as signals and images. The number of workers that can be used to solve the CSC scales linearly with the data size, allowing to adapt the computational resources to the problem dimensions. Using careful distributed pre-computations, the complexity of the dictionary update step is independent of the data size. Experiments show that it has good scaling properties and demonstrate that DiCoDiLe is able to learn patterns in astronomical images for large size which are not handled by other state-of-the-art parallel algorithms.

Appendix A
Computation for the cost updates
When a coordinate \( Z_k[\omega] \) is updated to \( \tilde{Z}_k^{(q)}[\omega] \), the cost update is a simple function of \( Z_k[\omega] \) and \( \tilde{Z}_k^{(q)}[\omega] \).

Proposition A.1. If the coordinate \((k_0, \omega_0)\) is updated from value \( Z_{k_0}[\omega_0] \) to value \( \tilde{Z}_{k_0}^{(q)}[\omega_0] \), then the change in the cost function \( \Delta E_{k_0}[\omega_0] \) is given by:

\[
\Delta E_{k_0}[\omega_0] = \frac{||D_{k_0}||^2}{2} (Z_{k_0}[\omega_0]^2 - \tilde{Z}_{k_0}^{(q)}[\omega_0]^2)
- \beta_{k_0}[\omega_0](Z_{k_0}[\omega_0] - \tilde{Z}_{k_0}^{(q)}[\omega_0])
+ \lambda|Z_{k_0}[\omega_0] - |\tilde{Z}_{k_0}^{(q)}[\omega_0]|).
\]
Proof. We denote $Z_k^{(1)}[ω] = \{Z_k^{(q)}[ω], \text{ if } (k, ω) = (k_0, ω_0) \}
\{Z_k[ω], \text{ elsewhere} \}$. Let $α_0$ be the residual when coordinate $(k_0, ω_0)$ of $Z$ is set to 0. For $ω ∈ Ω$,
\[
α_{k_0}[ω] = (X - Z * D)[ω] + D_{k_0}[ω - ω_0]Z_{k_0}[ω_0] = (X - Z * D)[ω] + Z_{k_0}[ω_0](e_ω + D_{k_0})[ω_0].
\]
We also have $α_{k_0}[ω] = (X - Z^{(1)} * D)[ω] + D_{k_0}[ω - ω_0]Z_{k_0}^{(q)}[ω_0]$. Then
\[
ΔE_{k_0}[ω] = \frac{1}{2} \sum_{ω ∈ Ω} \frac{1}{2} \sum_{ω ∈ Ω} (X - Z * D)^2 [ω] + λ∥Z∥_1
\]
\[
= \frac{1}{2} \sum_{ω ∈ Ω} (α_{k_0}[ω] - D_{k_0}[ω - ω_0]Z_{k_0}[ω_0])^2
\]
\[
= \frac{1}{2} \sum_{ω ∈ Ω} (α_{k_0}[ω] - D_{k_0}[ω - ω_0]Z_{k_0}[ω_0])^2 + λ((Z_{k_0}[ω_0] - Z_{k_0}^{(q)}[ω_0]))
\]
\[
= \frac{1}{2} \sum_{ω ∈ Ω} (D_{k_0}[ω - ω_0]^2)(Z_{k_0}[ω_0] - Z_{k_0}^{(q)}[ω_0])^2
\]
\[
= \frac{1}{2} \sum_{ω ∈ Ω} (D_{k_0}[ω - ω_0]^2)(Z_{k_0}[ω_0] - Z_{k_0}^{(q)}[ω_0])^2 + λ((Z_{k_0}[ω_0] - Z_{k_0}^{(q)}[ω_0]))
\]
\[
γ[ω] = γ[ω] - D_{k_0}[ω - ω_0]Z_{k_0}[ω_0] + γ[ω] - D_{k_0}[ω - ω_0]Z_{k_0}^{(q)}[ω_0]
\]
\[
= γ[ω] - D_{k_0}[ω - ω_0]Z_{k_0}[ω_0] + γ[ω] - D_{k_0}[ω - ω_0]Z_{k_0}^{(q)}[ω_0].
\]
This concludes our proof. □

Using this result, we can derive the optimal value $Z_{k_0}^{(q)}[ω_0]$ to update the coordinate $(k_0, ω_0)$ as the solution of the following optimization problem:
\[
\hat{Z}_{k_0}^{(q)}[ω_0] = \arg \min_{y ∈ \mathbb{R}} \frac{1}{2} \sum_{ω ∈ Ω} (Z_{k_0}[ω] - Z_{k_0}^{(q)}[ω_0])^2 + λ∥u∥_1.
\]
In the case where multiple coordinates $(k_w, ω_w)$ are updated in the same iteration to values $Z_{k_w}^{(q)}[ω_w]$, we obtain the following cost variation.

Proposition A.2. The update of the $W$ coordinates $(k_w, ω_w)_{w=1}^W$ with additive update $ΔZ_{k_w}[ω_w]$ changes the cost by:
\[
ΔE = \sum_{w=1}^W ΔE_{k_w}
\]
\[
− \sum_{w' ≠ w'} (D_{k_w} * D_{k_w})[ω_w' - ω_w]ΔZ_{k_w}[ω_w]ΔZ_{k_w}[ω_w']
\]
Proof. We define $Z_k^{(1)}[ω] = \{Z_k^{(q)}[ω], \text{ if } (k, ω) = (k_w, ω_w) \}
\{Z_k[ω], \text{ otherwise} \}$. Let
\[
γ[ω] = γ[ω] + D_{k_w}[ω - ω_w]Z_{k_w}[ω_w] + γ[ω] + D_{k_w}[ω - ω_w]Z_{k_w}^{(q)}[ω_w].
\]
\[
= γ[ω] + D_{k_w}[ω - ω_w]Z_{k_w}[ω_w] + γ[ω] + D_{k_w}[ω - ω_w]Z_{k_w}^{(q)}[ω_w].
\]
Then
\[
ΔE = \frac{1}{2} \sum_{ω ∈ Ω} (X[ω] - Z * D^[ω])^2 + λ∥Z∥_1
\]
\[
− \sum_{ω ∈ Ω} (X[ω] - Z^{(1)} * D^[ω])^2 - λ∥Z^{(1)}∥_1
\]
\[
= \frac{1}{2} \sum_{ω ∈ Ω} (γ[ω] - D_{k_w}[ω - ω_w]Z_{k_w}[ω_w])^2
\]
\[
= \frac{1}{2} \sum_{ω ∈ Ω} (γ[ω] - D_{k_w}[ω - ω_w]Z_{k_w}[ω_w])^2 + λ((Z_{k_w}[ω_w] - Z_{k_w}^{(q)}[ω_w]))
\]
\[
= \frac{1}{2} \sum_{ω ∈ Ω} (D_{k_w}[ω - ω_w]^2)(Z_{k_w}[ω_w] - Z_{k_w}^{(q)}[ω_w])^2
\]
\[
= \frac{1}{2} \sum_{ω ∈ Ω} (D_{k_w}[ω - ω_w]^2)(Z_{k_w}[ω_w] - Z_{k_w}^{(q)}[ω_w])^2 + λ((Z_{k_w}[ω_w] - Z_{k_w}^{(q)}[ω_w]))
\]
\[
= \frac{1}{2} \sum_{w=1}^W (D_{k_w}[ω - ω_w^2]^2)(Z_{k_w}[ω_w] - Z_{k_w}^{(q)}[ω_w])^2
\]
\[
= \frac{1}{2} \sum_{w=1}^W (D_{k_w}[ω - ω_w^2]^2)(Z_{k_w}[ω_w] - Z_{k_w}^{(q)}[ω_w])^2 + λ((Z_{k_w}[ω_w] - Z_{k_w}^{(q)}[ω_w]))
\]
\[
= \frac{1}{2} \sum_{w=1}^W (D_{k_w}[ω - ω_w^2]^2)(Z_{k_w}[ω_w] - Z_{k_w}^{(q)}[ω_w])^2
\]
\[
= \frac{1}{2} \sum_{w=1}^W (D_{k_w}[ω - ω_w^2]^2)(Z_{k_w}[ω_w] - Z_{k_w}^{(q)}[ω_w])^2 + λ((Z_{k_w}[ω_w] - Z_{k_w}^{(q)}[ω_w]))
\]
\[
= \frac{1}{2} \sum_{w=1}^W (D_{k_w}[ω - ω_w^2]^2)(Z_{k_w}[ω_w] - Z_{k_w}^{(q)}[ω_w])^2
\]
\[
= \frac{1}{2} \sum_{w=1}^W (D_{k_w}[ω - ω_w^2]^2)(Z_{k_w}[ω_w] - Z_{k_w}^{(q)}[ω_w])^2 + λ((Z_{k_w}[ω_w] - Z_{k_w}^{(q)}[ω_w]))
\]
\[
= \frac{1}{2} \sum_{w=1}^W (D_{k_w}[ω - ω_w^2]^2)(Z_{k_w}[ω_w] - Z_{k_w}^{(q)}[ω_w])^2
\]
\[
= \frac{1}{2} \sum_{w=1}^W (D_{k_w}[ω - ω_w^2]^2)(Z_{k_w}[ω_w] - Z_{k_w}^{(q)}[ω_w])^2 + λ((Z_{k_w}[ω_w] - Z_{k_w}^{(q)}[ω_w]))
\]
\[
= \frac{1}{2} \sum_{w=1}^W (D_{k_w}[ω - ω_w^2]^2)(Z_{k_w}[ω_w] - Z_{k_w}^{(q)}[ω_w])^2
\]
\[
= \frac{1}{2} \sum_{w=1}^W (D_{k_w}[ω - ω_w^2]^2)(Z_{k_w}[ω_w] - Z_{k_w}^{(q)}[ω_w])^2 + λ((Z_{k_w}[ω_w] - Z_{k_w}^{(q)}[ω_w]))
\]
\[
= \frac{1}{2} \sum_{w=1}^W (D_{k_w}[ω - ω_w^2]^2)(Z_{k_w}[ω_w] - Z_{k_w}^{(q)}[ω_w])^2
\]
\[
= \frac{1}{2} \sum_{w=1}^W (D_{k_w}[ω - ω_w^2]^2)(Z_{k_w}[ω_w] - Z_{k_w}^{(q)}[ω_w])^2 + λ((Z_{k_w}[ω_w] - Z_{k_w}^{(q)}[ω_w]))
\]
\[
= \frac{1}{2} \sum_{w=1}^W (D_{k_w}[ω - ω_w^2]^2)(Z_{k_w}[ω_w] - Z_{k_w}^{(q)}[ω_w])^2
\]
\[
= \frac{1}{2} \sum_{w=1}^W (D_{k_w}[ω - ω_w^2]^2)(Z_{k_w}[ω_w] - Z_{k_w}^{(q)}[ω_w])^2 + λ((Z_{k_w}[ω_w] - Z_{k_w}^{(q)}[ω_w]))
\]
\[
= \frac{1}{2} \sum_{w=1}^W (D_{k_w}[ω - ω_w^2]^2)(Z_{k_w}[ω_w] - Z_{k_w}^{(q)}[ω_w])^2
\]
\[
= \frac{1}{2} \sum_{w=1}^W (D_{k_w}[ω - ω_w^2]^2)(Z_{k_w}[ω_w] - Z_{k_w}^{(q)}[ω_w])^2 + λ((Z_{k_w}[ω_w] - Z_{k_w}^{(q)}[ω_w]))
\]

□
APPENDIX B

PROBABILITY OF A COORDINATE TO BE SOFT-LOCKED

Proposition B.1. We consider DiCoDiLe with \( W \) workers (\( W_i \) in each direction), run on a signal with \( d \) dimensions. We denote \( T_i \), the length of the signal and \( L_i \), the length of the atoms in each dimension \( i \). Assuming that the updates are spread uniformly on \( \Omega \), the probability that an update candidate at random localization \( \omega \) is accepted is lower bounded by

\[
P(\omega \notin \mathbf{SL}) \geq d \prod_{i=1}^{d} \left( 1 - \frac{W_i L_i}{T_i} \right)
\]

Proof. We denote \( \omega_0 \) the position of an update candidate, and \( W_{\omega_0} = \{ w \in [1, W] \text{ s.t. } S_w \cap V_{\omega_0}(\omega) = \emptyset \} \) the set of workers that are impacted by this update. We denote \( j_{\omega_0} = |W_{\omega_0}| \) the number of workers in this set. With the definition of the soft-lock, there exists at least one worker \( w \in W_{\omega_0} \), which has a coordinate that is not soft-locked. As the updates are uniformly spread across the workers, the probability of an update in \( \omega_0 \) being accepted satisfies \( P(\omega \notin \mathbf{SL} | \omega = \omega_0) \geq \frac{1}{j_{\omega_0}} \).

Since one assumes uniform update distribution, the probability to update a coordinate at location \( \omega_0 \) is \( P(\omega_0) = \frac{1}{W} \). The probability for an update candidate at a random location \( \omega \) to be accepted satisfies

\[
P(\omega \notin \mathbf{SL}) = \sum_{w=1}^{W} \sum_{\omega_0 \in S_w} P(\omega \notin \mathbf{SL} | \omega = \omega_0) P(\omega_0)
\]

\[
\geq \sum_{w=1}^{W} \sum_{\omega_0 \in S_w} \frac{1}{j_{\omega_0}} \prod_{i=1}^{d} \left( 1 - \frac{W_i L_i}{T_i} \right)
\]

\[
\geq \prod_{i=1}^{d} \left( 1 - \frac{W_i L_i}{T_i} \right)
\]

where \( (B.3) \) results from the summation on \( \Omega \). Basically, in each worker \( w \), there is \( \prod_{i=1}^{d} \left( T_i/W_i - 2L_i \right) \) coordinates for which \( j_{\omega_0} = 1 \). This corresponds to the coordinate in the worker that are not on the border i.e. \( S_w \setminus B_0(S_w) \). For a coordinate at the border with \( j_{\omega_0} = k > 1 \), it suffices to notice that it will appear \( k \) times in the sum due to symmetry considerations. This is illustrated in Figure 10.

\[
\square
\]

ACKNOWLEDGMENTS

This work was supported by the ERC Starting Grant SLAB ERC-YSIG-676943.

REFERENCES


Thomas Moreau received the graduate degree from the Ecole Polytechnique, Palaiseau, France, in 2014, and the PhD degree from the Ecole Normale Supérieure, Cachan, France, in 2017 under the supervision of Nicolas Vayatis and Laurent Oudre in the GMLA laboratory. He recently joined the Inria Parietal project team in Saclay, first as a post-doctoral researcher and then as a researcher. His research interests include unsupervised learning, image/signal processing and distributed computing.

Alexandre Gramfort graduated from Ecole Polytechnique in 2005 and Telecom ParisTech in 2006, and he holds a MS in Applied Mathematics from Ecole Normale Supérieure, Cachan, France. He is currently senior researcher in the Parietal Team at INRIA Saclay Research Center and CEA Neurospin since 2017. He was formerly Assistant Professor at Telecom ParisTech, Université Paris-Saclay, in the image and signal processing department. His field of expertise is statistical machine learning, signal processing and scientific computing applied primarily to functional brain imaging data (EEG, MEG, fMRI).