Distributed Convolutional Sparse Coding via Message Passing Interface (MPI)

Thomas Moreau
CMLA, ENS Cachan, CNRS,
Université Paris-Saclay, 94235 Cachan, France
thomas.moreau@cmla.ens-cachan.fr

Laurent Oudre
L2TI, Institut Galilée
Université Paris 13, 93430 Villetaneuse
laurent.oudre@univ-paris13.fr

Nicolas Vayatis
CMLA, ENS Cachan, CNRS,
Université Paris-Saclay, 94235 Cachan, France
nicolas.vayatis@cmla.ens-cachan.fr

1 Introduction

Convolutional sparse coding methods focus on building representations of time signals as sparse and linear combinations of shifted patterns. These techniques have proven to be useful when dealing with signals (such as ECG or images) which are composed of several characteristic patterns ([1, 2, 3]). For this type of signals, the shapes and positions of these templates are crucial for their study (segmentation, classification, etc...). Depending on the context, the dictionary can either be fixed analytically, or learned from the data, using an alternated minimization of the representation cost over the dictionary and the representation.

Several algorithms have been proposed to solve the convolutional sparse coding problem with univariate signals, and most of them can be extended to multivariate signal without major difficulties ([2, 4, 5, 6]). However, these algorithms reach their computational limits for high values of the length of the signal. This is a typical situation, for instance, with physiological signal processing where sensor information can be collected for a few minutes with high sampling rates ([7]). In such context, there is a need for distributed algorithms which would address the problem of convolutional sparse coding. Recent studies have considered the distributed coordinate descent for general ℓ_1-regularized minimization problems, which include the convolutional coding problem ([8, 9, 10, 11, 12]). As those general frameworks do not use the particular structure of the problem, they lead to more complicated distribution strategies, using either locks or synchronizing steps. In the case of convolutional sparse coding, these limitations can be removed with few assumptions as we show in the present paper, leading to a fully parallelized algorithm to solve this task. This algorithm can be implemented via Message Passing Interface (MPI) and thus scales up efficiently on big architectures. It also provides a super-linear speedup.

2 Distributed Convolutional Coordinate Descent

2.1 Convolutional sparse coding and coordinate descent

The convolutional sparse coding problem considers a multivariate signal x in $\mathbb{R}^P$ of length T and a dictionary $\boldsymbol{D}$ of K signals $\boldsymbol{D}_k$ in $\mathbb{R}^P$ of length S. The goal of this method is to find an activation signal z in $\mathbb{R}^K$ of length $T - S$ such that:

$$x[t] = \sum_{k=1}^{K} \sum_{\tau=0}^{S-1} z_k[t-\tau]\boldsymbol{D}_k[\tau] = (z * \boldsymbol{D})[t] \quad \text{for } t \in [0, T\text{-}1], \tag{1}$$

It is also assumed that the weights are sparsely distributed over the signal. The recovery of this sparse activation signal can be achieved by solving an ℓ_1-regularized optimization problem:

$$z^* = \arg\min_z E(z) = \frac{1}{2}\|x - z * \boldsymbol{D}\|_2^2 + \lambda\|z\|_1, \tag{2}$$

where λ is a regularization parameter enforcing the sparsity of z.

In [2], the authors propose to use the coordinate descent (CD) to solve the convolutional sparse coding problem (2). This method replaces at each step one coordinate $z_{k_0}[t_0]$ by its optimal value $z'_{k_0}[t_0]$, which can be computed with a close form:

$$z'_k = \frac{1}{\|\boldsymbol{D}_k\|_2^2}\text{Sh}(\beta_k, \lambda), \tag{3}$$

where $\text{Sh}(y,\lambda)=\text{sign}(y)(|y|-\lambda)_+$ and $\beta_k[t]=\Big((x-\Phi_{k,t}(z)*\boldsymbol{D})*\widetilde{\boldsymbol{D}}_k\Big)[t]$. The replacement operator $\Phi_{k,t}(z)$ is defined as the same signal z except for the value $z_k[t]$ which is replaced by 0 and $\widetilde{D}_k$ denotes the reversed pattern such that $\widetilde{D}_k[t]=D_k[\text{-}t]$. The coordinate to replace is chosen such that $k_0, t_0 = \arg\max_{k,t} |z'_k[t] - z_k[t]|$. This algorithm converges under mild conditions to the optimal solution of (2) ([13]).

These updates can be done efficiently by keeping a version of β in memory. After an update of z in (k_0,t_0) for the q-th iteration, only $K(2S-1)$ coefficients of β need to be changed with $\beta_k^{(q+1)}[t]=\beta_k^{(q)}[t]-\mathcal{S}_{k,k_0}[t-t_0](z_{k_0}[t_0]-z'_{k_0}[t_0])$, where $\mathcal{S}_{k,k_0}[t] = \sum_{\tau=0}^{S-1} \boldsymbol{D}_k[t+\tau]\boldsymbol{D}_{k_0}^T[\tau]$ is a signal in $\mathbb{R}^{K\times K}$ for $t\in[\text{-}S+1,S-1]$ representing the cross-correlation between the dictionary elements of index k and k_0.

2.2 Distributed Convolutional Coordinate Descent (DICOD)

We propose a new distributed algorihtm for the convolutional sparse coding (DICOD), running asynchronously and providing superlinear speedup. The key point in DICOD is that the solutions on time segments that are not overlapping are weakly dependent. The signal is split in time segments of size $L_p = L/M$ over the M cores $(\mathcal{C}_m)_{1\leq m\leq M}$. Each core $\mathcal{C}_m$ will update independently $\{z_k[t] : k \in [1, K],\ \ t \in [(m-1)L_p, mL_p]\}$using the same mechanism than CD. Then, a core needs to notify its neighbor only when it updates a coefficient at the border of its chunk. When two cores update their border simultaneously, they generate an interference.

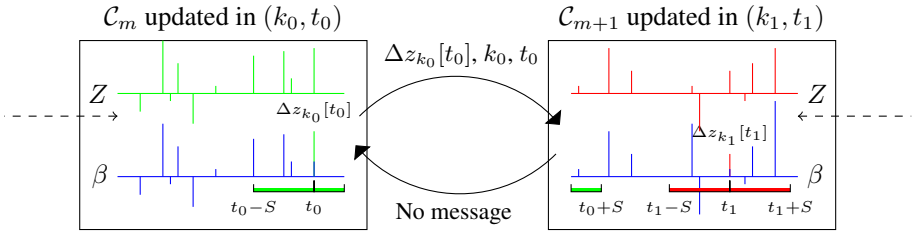

Figure 1: The core $\mathcal{C}_m$ is updating a coordinate in t_0 which is in interference zone $[mL_p-S,mL_p]$. Therefore it needs to notify $\mathcal{C}_{m+1}$ of the update by sending a message composed of the value of the update$\Delta z_{k_0}[t_0] = z_{k_0}[t_0] - z'_{k_0}[t_0]$ and its location (k_0, t_0). The core $\mathcal{C}_{m+1}$ can do its update in t_1 independently as it is located out of interference zones. When the core $\mathcal{C}_{m+1}$ retrieves the pending message from $\mathcal{C}_m$, it will update β to take into account the update of $z_{k_0}[t_0]$.

A significant improvement compared to other parallel implementations of the coordinate descent [8, 9, 10, 11] is that the present version performs the updates asynchronously. The interferences do not change the convergence properties of the algorithm, but they slow down its convergence rate, as it is shown in Section 3. With some interferences, the M updates performed in parallel by DICOD are not as effective as if they had been performed sequentially. However, the interference probability is $\left(\frac{M\cdot S}{T}\right)^2$, and thus if S is small compared to $\frac{T}{M}$, only few interferences will affect the convergence.

Another improvement compared to CD is that each update is computationally cheaper. To evaluate this gain, a sequential DICOD algorithm (SeqDICOD$_M$) was also considered, where the updated coordinates were sequentially chosen from a rolling interval $\left[m\frac{L}{M},(m+1)\frac{L}{M}\right[$.

Algorithm 1 Distributed Convolutional Coordinate Descent (DICOD)

Require: $\boldsymbol{D}, x$, and parameters $\epsilon \in \mathbb{R}_+^*$ and $M \in \mathbb{N}^*$

$\beta_k[t] \leftarrow \left(\widetilde{\boldsymbol{D}}_k * x\right)[t]$

for all core $m = 1 \dots M$ in parallel **do**

Retrieve pending messages and update β accordingly

$z'_k[t] \leftarrow \frac{1}{\|\boldsymbol{D}_k\|_2^2} \text{Sh}(\beta_k[t], \lambda), \quad \forall (k,t) \in \mathcal{C}_m$

$(k_0, t_0) \leftarrow \arg\max_{(k,t) \in \mathcal{C}_m} |z_k[t] - z'_k[t]|$

$\beta_k[t] \leftarrow \beta_k[t] - \mathcal{S}_{k,k_0}[t - t_0](z_{k_0}[t_0] - z'_{k_0}[t_0])$

$z_{k_0}[t_0] \leftarrow z'_{k_0}[t_0]$

if $t_0 + S > mL_p$ or $t_0 - S < (m-1)L_p$ **then** Notify neighbor processes

Until $|z_{k_0}[t_0] - z'_{k_0}[t_0]| < \epsilon$

end for

3 Properties of DICOD

3.1 Convergence Analysis

The coordinate descent for the problem (2) can be distributed and run without core synchronization. The key point is that the synchronization is only here to control the interference part. But such interference cannot break the convergence property of the algorithm, except for highly correlated dictionary elements. The interference magnitude is closely related to the value of the cross-correlation between dictionaries

$$C_{k_0 k_1}[t_0 - t_1] = \frac{\mathcal{S}_{k_0,k_1}[t_0 - t_1]}{\|\boldsymbol{D}_{k_0}\|_2 \|\boldsymbol{D}_{k_1}\|_2}. \tag{4}$$

Theorem 1. *We consider the following assumptions:*

H 1. *If for all k_0, k_1, t_0, t_1 such that $t_0 - t_1 \neq 0$, $|C_{k_0 k_1}[t_0 - t_1]| < 1$.*

H 2. *If there exist $A \in \mathbb{N}^*$ such that for all $m \in \{1, \dots M\}$ and $q \in \mathbb{N}$, $\mathcal{C}_m$ is updated at least once between iteration q and $q + A$ if the solution is not optimal for all coefficients assigned to $\mathcal{C}_m$.*

Under these assumptions, the DICOD algorithm converges to the optimal solution z^* of (2).

H 1 is satisfied as long as a dictionary elements are not replicated in shifted positions in the dictionary. This assumption ensures that at each step, the cost is updated in the right direction. **H** 2 guarantees that all coefficients are updated regularly if they are not optimal.

3.2 Computational Complexity Analysis and Speedup through DICOD

The complexity of each iteration for CD is linear with the length of the input signal T as the dominant operation is to compute $(k_0, t_0) = \arg\max_{(k,t)} |z_k[t] - z'_k[t]|$. As in DICOD, each node runs similar updates over subsignals of length T/M, the iteration complexity reduces linearly in the number of cores. The convergence rate of CD is exponential in a certain constant $\gamma < 1$ ([14]). With an analysis of the interference probability, the convergence rate of DICOD with M cores can be bounded by:

$$\mathbb{E}\left[E(z^{(q+1)}) - E(z^*)\right] \leq \gamma^M \left[1 + \alpha^2 \frac{1 - \gamma^2}{\gamma}\right]^{M-1} \mathbb{E}\left[E(z^{(q)}) - E(z^*)\right], \tag{5}$$

with $\alpha^2 = \left(\frac{SM}{T}\right)^2$ the probability of interference. For α close to 0, the convergence rate is γ^M, corresponding to an overall quadratic speedup. An analysis of this rate shows that the speedup is superlinear, as long as $M \leq \frac{T}{S}\sqrt{\frac{1-\gamma}{1-\gamma^2}}$.

4 Numerical Results

All the numerical experiments are run on five Linux machines with 6 to 24 Intel Xeon 2.70 GHz processors and at least 64 GB of RAM on local network. We use a combination of Python, C++ and the OpenMPI 1.6 for the different algorithms. The DICOD algorithm is tested over artificial

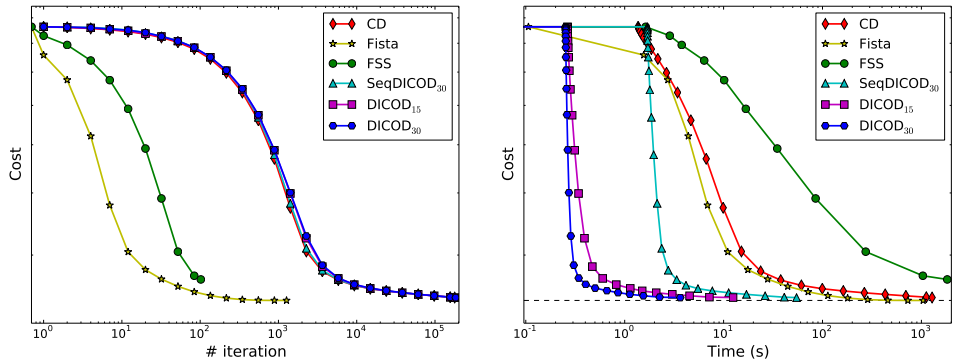

Figure 2: Cost evolution for CD [2], FISTA [4], FSS [6], SeqDICOD$_{30}$ and DICOD$_M$ (DICOD with M cores) relatively to the algorithm iteration *(left)* and runtime *(right)* for an artificial signal generated with default parameters.

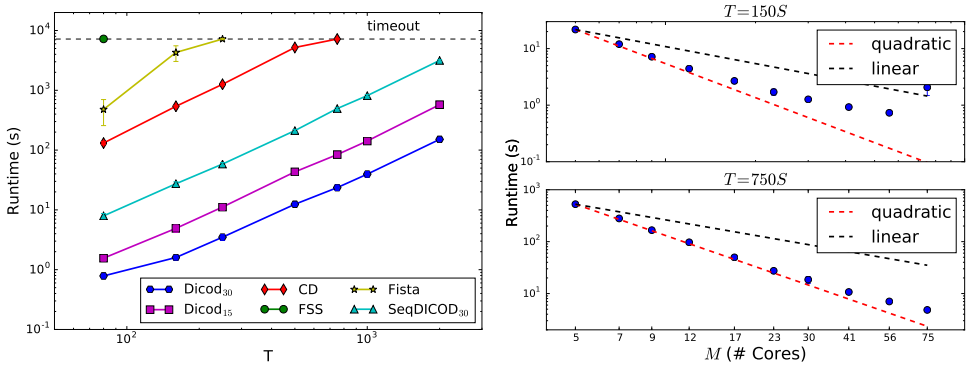

Figure 3: Runtime in seconds averaged over 10 repetitions for different problem sizes T, ranging from $80S$ to $2000S$ *(left)* and for different numbers of cores M, ranging from 5 to 75 cores, for 2 problem sizes *(right top)* $150S$, *(right bottom)* $750S$. A timeout is set to 2h.

signals generated as follows. A dictionary $\boldsymbol{D}$ of K patterns in $\mathbb{R}^P$ of length $S = 200$ is generated as multivariate harmonic signals defined for $p \in [1, P]$ by:

$$\boldsymbol{D}_{k,p}[t] = a_{k,p} \sin(2\pi f_{k,p} t + \phi_{k,p}), \tag{6}$$

with a sample rate of 200Hz. The component frequencies $f_{k,p}$ are drawn independently, uniformly distributed over $[0, 20]$Hz, the intensity $a_{k,p}$ from $[0, 10]$ and the phases $\phi_{k,p}$ from $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$. Then, we generate a sparse code signal z with a percentage ρ of non-null values drawn uniformly from $[0, 10]$. The signal x is defined as the convolution product between the generated dictionary $\boldsymbol{D}$ and the coding weights z such as $x = z * \boldsymbol{D} + \sigma$ with σ a gaussian white noise with power 1. The default values for the parameters are set to $S = 200$, $K = 10$, $P = 7$, $T = 250S$, $\rho = 0.007$ and we used $\lambda = 0.1$ and $\epsilon = 5e^{-2}$.

Figure 2 presents the evolution of the cost function value relatively to the number of iterations, here counted as the number of updates for DICOD, and to the computation time for an artificial problem. In the left figure, the variations between the curve of DICOD$_{20}$, DICOD$_{10}$ and CD are due to the interfering steps that slow down the convergence rate as seen in Section 3.2. The right curve displays the speed of the algorithms for a convolutional sparse coding problem. The serial version of the distributed algorithm SeqDICOD achieves a notable speed up compared to CD, suggesting that this idea could also be used in the distributed algorithm to further improve the performance.

Figure 3 displays the running time averaged over 10 repetitions as a function of the problem sizes, and the number of cores. In particular, the plots confirm that DICOD can solve large problems with low time budgets. The right figure also highlights the speed up obtain with the parallelization. We can see that it is close to quadratic for low value of $\alpha = SM/T$ and as M grow, the speedup get lower. The speedup gets sub-linear for $M = \frac{T}{2S}$ as the number of interference gets to high. Empirically, for $M \geq \frac{T}{4S}$, the gain in speed is low.

References

[1] A. Adler, M. Elad, Y. Hel-Or, and E. Rivlin. Sparse Coding with Anomaly Detection. In *IEEE International Workshop on Machine Learning for Signal Processing (MLSP), Southampton, UK*, pages 22 – 25, 2013.

[2] Koray Kavukcuoglu, Pierre Sermanet, Y-lan Boureau, Karol Gregor, and Yann Lecun. Learning Convolutional Feature Hierarchies for Visual Recognition. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1–9, 2013.

[3] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online Learning for Matrix Factorization and Sparse Coding. *Journal of Machine Learning Research*, 11:19–60, 2009.

[4] Rakesh Chalasani, Jose C. Principe, and Naveen Ramakrishnan. A fast proximal method for convolutional sparse coding. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 2013.

[5] Hilton Bristow, Anders Eriksson, and Simon Lucey. Fast convolutional sparse coding. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 391–398, 2013.

[6] Roger Grosse, Rajat Raina, Helen Kwong, Symbolic Systems Program, and Andrew Y Ng. Shift-Invariant Sparse Coding for Audio Classification. *Cortex*, 8:9, 2007.

[7] Charles others Coles, Lisa D and Patterson, Edward E and Sheffield, W Douglas and Mavoori, Jaideep and Higgins, Jason and Michael, Bland and Leyde, Kent and Cloyd, James C and Litt, Brian and Vite. Feasibility study of a caregiver seizure alert system in canine epilepsy. *Epilepsy research*, 106:456–460, 2013.

[8] Chad Scherrer, Mahantesh Halappanavar, Ambuj Tewari, and David Haglin. Scaling Up Coordinate Descent Algorithms for Large 1 Regularization Problems. Technical report, 2012.

[9] Chad Scherrer, Ambuj Tewari, Mahantesh Halappanavar, and David J. Haglin. Feature Clustering for Accelerating Parallel Coordinate Descent. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1–9, 2012.

[10] Joseph K. Bradley, Aapo Kyrola, Danny Bickson, and Carlos Guestrin. Parallel Coordinate Descent for L1-Regularized Loss Minimization. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 321–328, 2011.

[11] Hsiang Fu Yu, Cho Jui Hsieh, Si Si, and Inderjit Dhillon. Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, pages 765–774, 2012.

[12] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M Hellerstein. Distributed GraphLab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, 5:716–727, 2012.

[13] Stanley Osher and Yingying Li. Coordinate descent optimization for l1 minimization with application to compressed sensing; a greedy algorithm. *Inverse Problems and Imaging*, 3:487–503, 2009.

[14] Julie Nutini, Mark Schmidt, Issam H Laradji, Michael Friedlander, and Hoyt Koepke. Coordinate Descent Converges Faster with the Gauss-Southwell Rule Than Random Selection. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1–34, 2015.