Learning Efficient Sparse and Low Rank Models

P. Sprechmann, A. M. Bronstein, and G. Sapiro

Abstract—Parsimony, including sparsity and low rank, has been shown to successfully model data in numerous machine learning and signal processing tasks. Traditionally, such modeling approaches rely on an iterative algorithm that minimizes an objective function with parsimony-promoting terms. The inherently sequential structure and data-dependent complexity and latency of iterative optimization constitute a major limitation in many applications requiring real-time performance or involving large-scale data. Another limitation encountered by these modeling techniques is the difficulty of their inclusion in discriminative learning scenarios. In this work, we propose to move the emphasis from the model to the pursuit algorithm, and develop a *process-centric* view of parsimonious modeling, in which a learned deterministic fixed-complexity pursuit process is used in lieu of iterative optimization. We show a principled way to construct learnable pursuit process architectures for structured sparse and robust low rank models, derived from the iteration of proximal descent algorithms. These architectures learn to approximate the exact parsimonious representation at a fraction of the complexity of the standard optimization methods. We also show that appropriate training regimes allow to naturally extend parsimonious models to discriminative settings. State-of-the-art results are demonstrated on several challenging problems in image and audio processing with several orders of magnitude speedup compared to the exact optimization algorithms.

Index Terms—parsimonious modeling, sparse and low-rank models, NMF, deep learning, real-time implementations, big data, proximal methods.

1 Introduction

Parsimony, preferring a simple explanation to a more complex one, is probably one of the most intuitive principles widely adopted in the modeling of nature. The past two decades of research have shown the power of parsimonious representation in a vast variety of applications from diverse domains of science.

One of the simplest among parsimonious models is *sparsity*, asserting that the signal has many coefficients close or equal to zero when represented in some domain, usually referred to as *dictionary*. The pursuit of sparse representations was shown to be possible using tools from convex optimization, in particular, via ℓ_1 norm minimization [1], [2]. Works [3], [4], followed by many others, introduced efficient computational techniques for dictionary learning and adaptation. Sparse modeling is in the heart of modern approaches to image enhancement such as denoising, demosaicing, impainting, and super-resolution, to mention just a few.

As many classes of data are not described well by the element-wise sparsity model and the ℓ_1 norm inducing it, more elaborate structured sparse models have been developed, in which non-zero elements are no more unrelated, but appear in groups or hierarchies of groups [5]–[9]. Such models have been shown useful in the analysis of functional MRI and genetic data for example.

In the case of matrix-valued data, complexity is naturally measured by the rank, which also induces a notion of parsimony. A recent series of works have elucidated the beautiful relationship between sparsity and low rank representations, showing that rank minimization can be achieved through convex optimization [10], [11]. The combination of low-rank and sparse models paved the path to new robust alternatives of principal component analysis (RPCA) [11], [12] and nonnegative matrix factorization (RNMF) [13], and addressing challenging matrix completion problems [11]. RPCA was also found useful in important applications such as face recognition and modeling, background modeling, and audio source separation. Another relevant low rank modeling scheme is non-negative matrix factorization (NMF) [14], where the input vectors are represented as non-negative linear combination of a non-negative under-complete dictionary. NMF has been particularly successful in applications such as object recognition and audio processing.

1.1 From model-centric to process-centric parsimonious modeling

All parsimonious modeling methods essentially follow the same pattern: First, an objective comprising a data

P. Sprechmann and G. Sapiro are with the Department of Electrical and Computer Engineering, Duke University, Durham 27708, USA. Email: pablo.sprechmann@duke.edu, guillermo.sapiro@duke.edu.

A. M. Bronsteind is with School of Electrical Engineering, Tel Aviv University, Tel Aviv 69978, Israel.Email: bron@eng.tau.ac.il.

Work partially supported by NSF, ONR, NGA, DARPA, AFOSR, ARO, and BSF.

fitting term and parsimony-promoting penalty terms is constructed; next, an iterative optimization algorithm is invoked to minimize the objective, pursuing either the parsimonious representation of the data in a given dictionary, or the dictionary itself. Despite its remarkable achievements, such a *model-centric* approach suffers from critical disadvantages and limitations [15], [16].

The inherently sequential structure and the data-dependent complexity and latency of iterative optimization tools often constitute a major computational bottleneck. The quest for efficiently solving sparse representation pursuit has given rise to a rich family of algorithms, both for sparse coding [17]–[22] , RPCA [11], [16], [23], [24] and NMF [14] problems. Despite the permanent progress reported in the literature, the state-of-the-art algorithms are still impractical in scenarios demanding real-time performance or involving large-scale data [16].

In this paper, we take several steps to depart from the model-centric ideology relying on an iterative solver by shifting the emphasis from the model to the *pursuit process*. Our approach departs from the observation that, despite being highly non-linear and hard to compute, the mapping between a data vector and its parsimonious representation resulting from the optimization procedure is deterministic. The curse of dimensionality precludes the approximation of this mapping on all possible, even modestly sized input vectors; however, since real data tend to have low intrinsic dimensionality, the mapping can be inferred explicitly on the support of the distribution of the input data.

Recently, [25], [26] have proposed to trade off precision in the sparse representation for computational speed-up by learning non-linear regressors capable of producing good approximations of sparse codes in a fixed amount of time. In their inspiring recent paper, [15] showed that a particular network architecture can be derived from the iterative shrinkage-thresholding (ISTA) [17] and proximal coordinate descent (CoD) algorithms [19]. These works were among the first to bridge between the optimization based sparse models and the inherently process-centric neural networks, and in particular auto-encoder networks [27], [28], extensively explored by the deep learning community.

1.2 Contributions

The main contribution of this paper is a comprehensive framework for process-centric parsimonious modeling. The four main contributions are:

First, we begin by proposing a principled way to construct encoders (Section 4) capable of approximating an important family of parsimonious models (Section 2). By extending the original ideas in [15], we propose tailored pursuit architectures derived from first-order proximal descent algorithms, Section 3.

Second, this new approach allows the encoders to be trained in an online manner, making the fast encoders no more restricted to work with a fixed dictionary and a fixed distribution of input vectors known *a priori* (limitation existing, for example, in [15]), and also removing the need to run the exact algorithms at training.

Third we incorporate the parsimonious representation into higher-level optimization problems. In Section 5, we show various alternatives including registration and discriminative scenarios.

Finally, in Secion 6 we demonstrate our framework in several applications, achieving similarly to or better performance compared to the model-centric approaches at a fraction of the complexity of the latter. The present paper generalizes and gives a more rigorous treatment to results previously published by the authors in conferences [29], [30].

2 Parsimonious models

Let $\mathbf{X} \in \mathbb{R}^{m\times n}$ be a give data matrix. In this work, we concentrate our attention on the general *parsimonious modeling* problem that can be posed as the solution of the minimization problem

$$\min \frac{1}{2}\left\|\mathbf{X}-\mathbf{D}\mathbf{Z}\right\|_{\mathrm{F}}^{2}+\psi(\mathbf{Z})+\phi(\mathbf{D}), \tag{1}$$

optimized over $\mathbf{Z} \in \mathbb{R}^{q\times n}$ alone or jointly with $\mathbf{D} \in \mathbb{R}^{m\times q}$. Here $\mathbf{Z}$ is the representation (parsimonious code) of the data in the dictionary, and the penalty terms $\psi(\mathbf{Z})$ and $\phi(\mathbf{D})$ induce a certain structure of the code and the dictionary, respectively.

We will explicitly distinguish between parsimonious *coding* or *representation pursuit* problems (representing data with a given model), and the harder parsimonious *modeling* problem (constructing a model describing given data, e.g., learning a dictionary). In the former, $\mathbf{D}$ is fixed and $\phi(\mathbf{D})$ is constant. Most useful formulations use convex penalty functions. However, the modeling problem is in any case non-convex.

In many relevant applications, the entire data matrix $\mathbf{X}$ is not available *a priori*. The data samples $\{\mathbf{x}_t\}_{t\in\mathbb{N}}$, $\mathbf{x}_t \in \mathbb{R}^m$, arrive sequentially; the index t should be interpreted as time. *Online parsimonious modeling* aims at

estimating and refining the model as the data come in [31]. The need for online schemes also arises when the available training data are simply too large to be handled together. When the regularized function ψ is vector-wise separable, $\psi(\mathbf{Z}) = \sum_{i=1}^{n} \psi(\mathbf{z}_i)$, problem (1) can be solved in an online fashion using an alternating minimization scheme. As a new data vector $\mathbf{x}_t$ is received, we first obtain its representation $\mathbf{z}_t$ given the current model estimate, $\mathbf{D}_{t-1}$. This is achieved by solving the representation pursuit problem

$$\mathbf{z}_t \quad = \quad \underset{\mathbf{z}}{\operatorname{argmin}} \frac{1}{2} \|\mathbf{x}_t - \mathbf{D}_{t-1}\mathbf{z}\|_2^2 + \psi(\mathbf{z}). \tag{2}$$

Then, we update the model using the coefficients, $\{\mathbf{z}_j\}_{j \leq t}$, computed during the previous steps of the algorithm,

$$\mathbf{D}_t \quad = \quad \underset{\mathbf{D}}{\operatorname{argmin}} \sum_{j=1}^{t} \beta_j \frac{1}{2} \|\mathbf{x}_j - \mathbf{D}\mathbf{z}_j\|_2^2 + \phi(\mathbf{D}), \tag{3}$$

where $\beta_j \in [0, 1]$ is a forgetting factor that can be added to rescale older information so that newer estimates have more weight. This can be efficiently solved without remembering all the past codes [31].

In what follows, we detail several important instances of (1), for both modeling or pursuit, and how they can be cast as online learning problems.

2.1 Structured sparsity

The underlying assumption of sparse models is that the input vectors can be reconstructed accurately as a linear combination of the dictionary atoms with a small number of non-zero coefficients. Sparse models are enforced by using sparsity-promoting regularizers $\psi(\mathbf{Z})$. The simplest choice of such a regularizer is $\psi(\mathbf{Z}) = \lambda \sum_{i=1}^{n} \|\mathbf{z}_i\|_1$ (with $\lambda > 0$), for which the pursuit problem can be split into n independent problems on the columns of $\mathbf{Z}$,

$$\min_{\mathbf{z} \in \mathbb{R}^q} \frac{1}{2} \|\mathbf{x} - \mathbf{D}\mathbf{z}\|_2^2 + \lambda \|\mathbf{z}\|_1. \tag{4}$$

This is the classical *unstructured* sparse coding problem, often referred to as Lasso [2] or basis pursuit [1].

Structured sparse models further assume that the pattern of the non-zero coefficients of $\mathbf{Z}$ exhibits a specific structure known *a priori*. Let $A \subseteq \{1, \ldots, q\}$ denote groups of indices of atoms. Then, we define a group structure, $\mathcal{G}$, as a collection of groups of atoms, $\mathcal{G} = \{A_1, \ldots, A_{|\mathcal{G}|}\}$. The regularizer corresponding to the group structure is defined as the column-wise sum

$$\psi_{\mathcal{G}}(\mathbf{Z}) = \sum_{i=1}^{n} \psi_{\mathcal{G}}(\mathbf{z}_i) \quad \text{where} \quad \psi_{\mathcal{G}}(\mathbf{z}) = \sum_{r=1}^{|\mathcal{G}|} \lambda_r \|\mathbf{z}_r\|_2, \tag{5}$$

and $\mathbf{z}_r$ denotes the subvector of $\mathbf{z}$ corresponding to the group of atoms A_r. The regularizer function ψ in the Lasso problem (4) arises from the special case of singleton groups $\mathcal{G} = \{\{1\}, \{2\}, \ldots, \{q\}\}$ and setting $\lambda_r = \lambda$. As such, the effect of $\psi_{\mathcal{G}}$ on the groups of $\mathbf{z}$ is a natural generalization of the one obtained with unstructured sparse coding: it "turns on" and "off" atoms in groups according to the structure imposed by $\mathcal{G}$.

Several important structured sparsity settings can be cast as particular cases of (5): *Group sparse coding*, a generalization of the standard sparse coding to the cases in which the dictionary is sub-divided into groups [5], in this case $\mathcal{G}$ is a partition of $\{1, \ldots, q\}$; *Hierarchical sparse coding*, assuming a hierarchical structure of the non-zero coefficients [7]–[9]. The groups in $\mathcal{G}$ form a hierarchy with respect to the inclusion relation (a tree structure), that is, if two groups overlap, then one is completely included in the other one; *Overlapping group sparse coding*, relaxing the hierarchy assumption so that groups of atoms A_r are allowed to overlap [6].

2.2 Low-rank models and robust PCA

Another significant manifestation of parsimony typical to many classes of data is low rank. The classical low rank model is *principal component analysis* (PCA), in which the data matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$ (each column of $\mathbf{X}$ is an m-dimensional data vector), is decomposed into $\mathbf{X} = \mathbf{L} + \mathbf{E}$, where $\mathbf{L}$ is a low rank matrix and $\mathbf{E}$ is a perturbation matrix.

PCA is known to produce very good results when the perturbation is small [32]. However, its performance is highly sensitive to the presence of samples not following the model. [11], [12] proposed to robustify the model by adding a new term to the decomposition to account for the presence of outliers, $\mathbf{X} = \mathbf{L} + \mathbf{O} + \mathbf{E}$, where $\mathbf{O}$ is an outlier matrix with a sparse number of non-zero coefficients of arbitrarily large magnitude. In one of its formulations, the *robust principal component analysis* can be pursued by solving the convex program

$$\min_{\mathbf{L}, \mathbf{O} \in \mathbb{R}^{m \times n}} \frac{1}{2} \|\mathbf{X} - \mathbf{L} - \mathbf{O}\|_{\mathrm{F}}^2 + \lambda_* \|\mathbf{L}\|_* + \lambda \|\mathbf{O}\|_1. \tag{6}$$

The same way the ℓ_1 norm is the convex surrogate of the ℓ_0 norm (i.e., the convex norm closest to ℓ_0), the nuclear norm, denoted as $||\cdot||_*$, is the convex surrogate of matrix rank. The parameter λ_* controls the tradeoff between the data fitting error and the rank of the approximation.

In [10] it was shown that the nuclear norm of a matrix of $\mathbf{L}$ can be reformulated as a penalty over all possible

factorizations

$$\|\mathbf{L}\|_* = \min_{\mathbf{A},\mathbf{B}} \frac{1}{2}\|\mathbf{A}\|_{\mathrm{F}}^2 + \frac{1}{2}\|\mathbf{B}\|_{\mathrm{F}}^2 \qquad \text{s.t.} \quad \mathbf{AB} = \mathbf{L}, \quad (7)$$

The minimum is achieved through the SVD of $\mathbf{L} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$: the minimizer of (7) is $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}^{\frac{1}{2}}$ and $\mathbf{B} = \mathbf{\Sigma}^{\frac{1}{2}}\mathbf{V}$.

In (6), neither the rank of $\mathbf{L}$ nor the level of sparsity in $\mathbf{O}$ are assumed known *a priori*. However, in many applications, it is a reasonable to have a rough upper bound of the rank, say $\mathrm{rank}(\mathbf{L}) \leq q$. Combining this with (7), it was proposed in [33] to reformulate (6) as

$$\min_{\mathbf{D}_0,\mathbf{S},\mathbf{O}} \frac{1}{2}\|\mathbf{X} - \mathbf{D}_0\mathbf{S} - \mathbf{O}\|_{\mathrm{F}}^2 + \frac{\lambda_*}{2}(\|\mathbf{D}_0\|_{\mathrm{F}}^2 + \|\mathbf{S}\|_{\mathrm{F}}^2) + \lambda\|\mathbf{O}\|_1, \quad (8)$$

with $\mathbf{D}_0 \in \mathbb{R}^{m\times q}$, $\mathbf{S} \in \mathbb{R}^{q\times n}$, and $\mathbf{O} \in \mathbb{R}^{m\times n}$. The low rank component can now be thought of as an undercomplete dictionary $\mathbf{D}_0$, with q atoms, multiplied by a matrix $\mathbf{S}$ containing in its columns the corresponding coefficients for each data vector in $\mathbf{X}$. This interpretation allows to write problem (6) in the form of our general parsimonious models (1), with $\mathbf{Z} = (\mathbf{S};\mathbf{O})$, $\psi(\mathbf{Z}) = \frac{\lambda_*}{2}\|\mathbf{S}\|_{\mathrm{F}}^2 + \lambda\|\mathbf{O}\|_1$, $\mathbf{D} = (\mathbf{D}_0, \mathbf{I}_{m\times m})$, and $\phi(\mathbf{D}) = \frac{\lambda_*}{2}\|\mathbf{D}_0\|_{\mathrm{F}}^2$. Furthermore, unlike the nuclear norm, this new formulation of the rank-reducing regularization is differentiable and vector-wise separable (well suited for the online setting (2)). However, problem (8) is no longer convex. Fortunately, it can be shown that any stationary point of (8), $\{\mathbf{D}_0, \mathbf{S}, \mathbf{O}\}$, satisfying $||\mathbf{X} - \mathbf{D}_0\mathbf{S} - \mathbf{O}||_2 \leq \lambda_*$ is an globally optimal solution of (8) [34]. Thus, problem (8) can be solved using an alternating minimization, as in our online setting, without the risk of falling into a stationary point that is not globally optimal.

2.3 Non-negative matrix factorization

Another popular low rank model is non-negative matrix factorization (NMF). Given a non-negative data matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$, NMF aims at finding a factorization $\mathbf{X} \approx \mathbf{D}\mathbf{Z}$ into non-negative matrices $\mathbf{D} \in \mathbb{R}^{m\times q}$ and $\mathbf{Z} \in \mathbb{R}^{q\times n}$, with $q \leq n$. The smallest possible value of q that allows an exact reconstruction is called *non-negative* rank [35]. The non-negative rank is always greater or equal than the actual rank. In general, this factorization is obtained by manually selecting a value of q and solving the highly non-convex problem

$$\min_{\mathbf{D},\mathbf{Z}\geq\mathbf{0}} \|\mathbf{X} - \mathbf{D}\mathbf{Z}\|_{\mathrm{F}}^2. \quad (9)$$

Problem (9) can be stated as particular instance of (1) by setting ψ and ϕ to be the sum of element-wise indicator functions of the non-negative orthant.

Similarly to PCA, NMF is sensitive to outliers in the data matrix. A robust variant can be obtained by adding a sparse outlier term to the decomposition, $\mathbf{X} \approx \mathbf{D}_0\mathbf{S}+\mathbf{O}$, as done in the RPCA model [13]. Again here the problem can be cast as a particular instance of (1) defining $\mathbf{Z} = (\mathbf{S};\mathbf{O})$ and $\mathbf{D} = (\mathbf{D}_0, \mathbf{I}_{m\times m})$. We can obtain a robust low-rank NMF problem solving [30]

$$\min_{\mathbf{D}_0,\mathbf{S},\mathbf{O}\geq\mathbf{0}} \frac{1}{2}\|\mathbf{X} - \mathbf{D}_0\mathbf{S} - \mathbf{O}\|_{\mathrm{F}}^2 + \frac{\lambda_*}{2}(\|\mathbf{D}_0\|_{\mathrm{F}}^2 + \|\mathbf{S}\|_{\mathrm{F}}^2) + \lambda\|\mathbf{O}\|_1. \quad (10)$$

The sum of the Frobenius norms of the non-negative matrices $\mathbf{D}$ and $\mathbf{S}$ gives an upper bound on the nuclear norm of their product [30]. While not being fully equivalent to a robust low-rank NMF problem, the objective in problem (10) still achieves both robustness to outliers and rank regularization. With some abuse of terminology, we will refer to problem (10) as to RNMF. Again, (10) is a particular instance of (1) well suited for the online setting. Refer to [30] for details on this model.

3 Proximal Methods

Proximal splitting is a powerful optimization technique that has been extensively used by the machine learning and signal processing communities for their simplicity, convergence guarantees, and the fact that they are well suited for tackling sparse and structured sparse coding problems that can be written as (1) (refer to [22] for recent reviews). In Section 4 we will use these algorithms to construct efficient learnable pursuit processes.

The proximal splitting method with fixed constant step, applied to pursuit problems (2), defines a series of iterates, $\{\mathbf{z}^k\}_{k\in\mathbb{N}}$,

$$\mathbf{z}^{k+1} = \boldsymbol{\pi}_{\alpha\psi}(\mathbf{z}^k - \frac{1}{\alpha}\mathbf{D}^{\mathrm{T}}(\mathbf{D}\mathbf{z}^k - \mathbf{x})), \quad (11)$$

$$\text{where} \quad \boldsymbol{\pi}_{\alpha\psi}(\mathbf{z}) = \operatorname*{argmin}_{\mathbf{u}\in\mathbb{R}^m} \frac{1}{2}||\mathbf{u} - \mathbf{z}||_2^2 + \alpha\psi(\mathbf{u}) \quad (12)$$

denotes the proximal operator of ψ. Fixed-step algorithms have been shown to have relatively slow sublinear convergence, $O(1/k)$, and many alternatives have been studied in the literature to improve the convergence rate [18], [20]. Accelerated versions can reach convergence rates of $O(1/k^2)$ which are the best possible for the class of first order methods in this type of problems. The discussion of theses methods is beyond of the scope of this paper.

Proximal splitting methods become particularly interesting when the proximal operator of ψ can be computed exactly and efficiently. Many important cases of

input : Data $\mathbf{x}$, dictionary $\mathbf{D}$, weights $\boldsymbol{\lambda}$.
output: Sparse code $\mathbf{z}$.
Define $\mathbf{H} = \mathbf{I} - \frac{1}{\alpha}\mathbf{D}^{\mathrm{T}}\mathbf{D}$, $\mathbf{W} = \frac{1}{\alpha}\mathbf{D}^{\mathrm{T}}$, $\mathbf{t} = \frac{1}{\alpha}\boldsymbol{\lambda}$.
Initialize $\mathbf{z}^0 = \mathbf{0}$ and $\mathbf{b}^0 = \mathbf{W}\mathbf{x}$.
for $k = 1, 2, \ldots$ *until convergence* **do**
  $\mathbf{z}^{k+1} = \boldsymbol{\pi}_{\mathbf{t}}(\mathbf{b}^k)$
  $\mathbf{b}^{k+1} = \mathbf{b}^k + \mathbf{H}(\mathbf{z}^{k+1} - \mathbf{z}^k)$
end

Algorithm 1: Iterative shrinkage-thresholding algorithm (ISTA).

input : Data $\mathbf{x}$, dictionary $\mathbf{D}_0$, weights λ, λ_*.
output: Approximation $\mathbf{l}$, outlier $\mathbf{o}$.
Define $\mathbf{H} = \mathbf{I} - \frac{1}{\alpha}\begin{pmatrix} \mathbf{D}_0^{\mathrm{T}}\mathbf{D}_0 + \lambda_*\mathbf{I} & \mathbf{D}_0^{\mathrm{T}} \\ \mathbf{D}_0 & (1+\lambda_*)\mathbf{I} \end{pmatrix}$,
$\mathbf{W} = \frac{1}{\alpha}\begin{pmatrix} \mathbf{D}_0^{\mathrm{T}} \\ \mathbf{I} \end{pmatrix}$, and $\mathbf{t} = \frac{\lambda}{\alpha}\begin{pmatrix} \mathbf{0} \\ \mathbf{1} \end{pmatrix}$.
Initialize $\mathbf{z}^0 = \mathbf{0}$, $\mathbf{b}^0 = \mathbf{W}\mathbf{x}$.
for $k = 1, 2, \ldots$ *until convergence* **do**
  $\mathbf{z}^{k+1} = \boldsymbol{\pi}_{\mathbf{t}}(\mathbf{b}^k)$
  $\mathbf{b}^{k+1} = \mathbf{b}^k + \mathbf{H}(\mathbf{z}^{k+1} - \mathbf{z}^k)$
end
Split $\mathbf{z}^{k+1} = (\mathbf{s}; \mathbf{o})$ and output $\mathbf{l} = \mathbf{D}_0\mathbf{s}$.

Algorithm 2: Proximal methods for the online RPCA, $\boldsymbol{\pi}_{\mathbf{t}}(\mathbf{b}) = \tau_{\mathbf{t}}(\mathbf{b})$, and online RNMF, $\boldsymbol{\pi}_{\mathbf{t}}(\mathbf{b}) = \tau^+_{\mathbf{t}}(\mathbf{b})$, problems with fixed $\mathbf{D}_0$.

structured sparsity fall into this category [22]. For the simple unstructured sparsity models induced by regularizers of the form $\psi(\mathbf{z}) = \|\boldsymbol{\lambda} \odot \mathbf{z}\|_1 = \sum_{i=1}^m \lambda_i z_i$, with $\odot$ denoting element-wise multiplication, the proximal operator reduces to the scalar soft-thresholding operator, $(\boldsymbol{\pi}_{\boldsymbol{\lambda}}(\mathbf{z}))_i = \tau_{\lambda_i}(z_i)$, with $\tau_\lambda(t) = \mathrm{sign}(t)\max\{0, |t| - \lambda\}$. In this case, the fixed step proximal splitting algorithm corresponds to the ISTA [17] summarized in Algorithm 1.

If $\mathbf{z}$ is furthermore constrained to be non-negative the soft thresholding is replaced by its one-sided counterpart $\tau^+_\lambda(t) = \max\{0, t - \lambda\}$. The proximal operator of the indicator function $i^+(t)$ imposing the non-negativity constraint is simply $\tau^+_0(t) = \max\{0, t\}$. The fixed-step proximal descent algorithm for the online RPCA and RNMF problems is summarized in Algorithm 2.

To generalize the proximal operator to the structured sparsity case, let us first consider an individual term in the sum (5), $\psi_r(\mathbf{z}) = \lambda_r \|\mathbf{z}_r\|_2$. Its proximal operator, henceforth denoted as $\boldsymbol{\pi}_{\lambda_r}$, can be computed as,

$$(\boldsymbol{\pi}_{\lambda_r}(\mathbf{z}))_s = \begin{cases} \frac{\mathbf{z}_r}{\|\mathbf{z}_r\|_2}\tau^+_{\lambda_r}(\|\mathbf{z}_r\|_2) & : \ s = r, \\ \mathbf{z}_r & : \ s \neq r, \end{cases} \tag{13}$$

where sub-indices r and s denote a sub-vector of the q_r-dimensional vector specified by the group of atoms A_r. Note that $\boldsymbol{\pi}_{\lambda_r}$ applies a group soft thresholding to the coefficients belonging to the r-th group and leaves the remaining ones unaffected. For a non-overlapping collection of groups $\mathcal{G}$, the proximal operator of $\psi_{\mathcal{G}}$ is group-separable and can be computed efficiently.

In general, when the groups of $\mathcal{G}$ overlap, there is no way of computing the proximal operator of $\psi_{\mathcal{G}}$ in closed form [6]. An important exception to this is the hierarchical setting with tree-structured groups. Then, the proximal operator of $\psi_{\mathcal{G}}$ can be shown to be given by the composition of the proximal operators of $\psi_{\mathcal{G}_l}$ in ascending order from the leaves to the root [8], [9]. A particular case of the tree-structured hierarchical sparse model is the two-level HiLasso model introduced to simultaneously promote sparsity at both group and coefficient level [9], [36]. Algorithm 1 is straightforward to generalize to the case of hierarchical sparsity by using the appropriate proximal operator [9].

It is worthwhile noting that the update in Algorithm 1 can be applied to a single element (or group) at a time in a (block) coordinate manner. When applied to a group of coefficients, the algorithm is referred as block-coordinate descent (BCoD); the particular case of a single-element update is called coordinate descent (CoD). In some particular cases (such as the HiLasso model), the proximal operator is not element-wise separable. In such situations, it is more appropriate to use a BCoD approach, see [29] for a discussion. Several variants of these algorithms have been proposed in the literature, see for example [19], [37]. Typically, one proceeds as in Algorithm 1, first applying the proximal operator $\mathbf{y} = \boldsymbol{\pi}(\mathbf{b}^k)$. Next, the residual $\mathbf{e} = \mathbf{y} - \mathbf{z}^k$ is evaluated, and a group is selected e.g. according to $r = \arg\max_r \|\mathbf{e}_r\|_2$ (in case of unstructured sparsity, $r = \arg\max_r |\mathbf{e}_r|$). Then, $\mathbf{b}^{k+1}$ is computed by applying $\mathbf{H}$ only to the selected subgroup of $\mathbf{e}$, and $\mathbf{z}^{k+1}$ is computed by replacing the subgroup of $\mathbf{z}^k$ with the corresponding subgroup of $\mathbf{y}$.

4 Learnable pursuit processes

The general parsimonious modeling problem (1) can be alternatively viewed as the minimization problem

$$\min_{\substack{\hat{\boldsymbol{z}}:\mathbb{R}^m \to \mathbb{R}^q \\ \hat{\boldsymbol{x}}:\mathbb{R}^q \to \mathbb{R}^m}} \frac{1}{n}\sum_{i=1}^n L(\mathbf{x}_i, \hat{\boldsymbol{z}}, \hat{\boldsymbol{x}}), \tag{14}$$

with $L(\mathbf{x}, \hat{\boldsymbol{z}}, \hat{\boldsymbol{x}}) = \frac{1}{2}\left\|(\hat{\boldsymbol{\imath}}\boldsymbol{d} - \hat{\boldsymbol{x}} \circ \hat{\boldsymbol{z}})(\mathbf{x})\right\|^2_{\mathrm{F}} + \psi(\hat{\boldsymbol{z}}(\mathbf{x})) + \phi(\hat{\boldsymbol{x}})$. The optimization is now performed over an *encoder* $\mathbf{z} = \hat{\boldsymbol{z}}(\mathbf{x})$ mapping the data vector $\mathbf{x}$ to the representation

vector $\mathbf{z}$, and a *decoder* $\mathbf{x} = \hat{\boldsymbol{x}}(\mathbf{z})$ performing the converse mapping.[1] The encoder/decoder pair is sought to make the composition $\hat{\boldsymbol{x}} \circ \hat{\boldsymbol{z}}$ close to the identity map, with the regularity promoted by the penalties ψ and ϕ.

Existing parsimonious models restrict the decoder to the class of linear functions $\hat{\boldsymbol{x}}_{\mathbf{D}}(\mathbf{z}) = \mathbf{D}\mathbf{z}$ parametrized by the dictionary $\mathbf{D}$. For a fixed dictionary $\mathbf{D}$, the optimal encoder is given by

$$\hat{\boldsymbol{z}}^* \quad = \quad \arg \min_{\hat{\boldsymbol{z}}:\mathbb{R}^m \to \mathbb{R}^q} L(\mathbf{x}, \hat{\boldsymbol{z}}, \hat{\boldsymbol{x}}_{\mathbf{D}}), \tag{15}$$

which is nothing but the solution of the representation pursuit problem obtained through and iterative optimization algorithm, such as the proximal methods described in Section 3. This interpretation is possible since the solution of the pursuit problem implicitly defines a deterministic mapping that assigns to each input vector $\mathbf{x} \in \mathbb{R}^n$ a unique parsimonious code $\mathbf{z} \in \mathbb{R}^m$. Note that the pursuit problem (in general) does not admit a closed form solution and cannot be expressed as a parametric function of fixed complexity.

In contrast, the process-centric approach proposed in this work, aims at formulating a modeling scheme were both the encoder and the decoder can be explicitly stated and efficiently computed. In our proposed framework, the encoders are constructed explicitly as parametric deterministic functions, $\hat{\boldsymbol{z}}_{\boldsymbol{\Theta}} : \mathbb{R}^n \to \mathbb{R}^m$ with a set of parameters collectively denoted as $\boldsymbol{\Theta}$, while the decoders are the exact same simple linear decoders, $\hat{\boldsymbol{x}}_{\mathbf{D}}(\mathbf{z}) = \mathbf{D}\mathbf{z}$, used in model-centric approaches (we relax this assumption in the following section). We denote by $\mathcal{F}_{\boldsymbol{\Theta}}$ the family of the parametric functions $\hat{\boldsymbol{z}}_{\boldsymbol{\Theta}}$. Naturally, two fundamental question arise: how to select the family $\mathcal{F}_{\boldsymbol{\Theta}}$ capable of defining good parsimonious models, and how to efficiently select the best parameters $\boldsymbol{\Theta}$ given a specific family. We will refer to the first problem as to selecting the *architecture* of the pursuit process, while the second will be referred to as *process learning*. We start with the latter, deferring the architecture selection to Section 4.3.

4.1 Process learning

With the process-centric perspective in mind, problem (14) can be stated as

$$\min_{\hat{\boldsymbol{z}}_{\boldsymbol{\Theta}} \in \mathcal{F}_{\boldsymbol{\Theta}}, \mathbf{D} \in \mathbb{R}^{m \times q}} \frac{1}{n} \sum_{i=1}^{n} L(\mathbf{x}_i, \hat{\boldsymbol{z}}_{\boldsymbol{\Theta}}, \hat{\boldsymbol{x}}_{\mathbf{D}}), \tag{16}$$

1. Note the difference in the notation: $\mathbf{z} \in \mathbb{R}^q$ represents a single coefficient vector associated with an input vector $\mathbf{x} \in \mathbb{R}^m$ (as was used throughout Section 3), while $\hat{\boldsymbol{z}} : \mathbb{R}^m \to \mathbb{R}^q$ represents a function.

where the family $\mathcal{F}_{\boldsymbol{\Theta}}$ imposes some desired characteristics on the encoder such as continuity and almost everywhere differentiability, and certain computational complexity. As in (1), this problem can be naturally solved using an alternating minimization scheme, sequentially minimizing for $\hat{\boldsymbol{z}}_{\boldsymbol{\Theta}}$ or $\mathbf{D}$ while leaving the other one fixed. Note that when the encoder $\hat{\boldsymbol{z}}_{\boldsymbol{\Theta}}$ is fixed, the problem in $\mathbf{D}$ remains essentially the same dictionary update problem and can be solved exactly as before. In what follows, we therefore concentrate on solving the process learning problem

$$\min_{\hat{\boldsymbol{z}}_{\boldsymbol{\Theta}} \in \mathcal{F}_{\boldsymbol{\Theta}}} \frac{1}{n} \sum_{i=1}^{n} L(\mathbf{x}_i, \hat{\boldsymbol{z}}_{\boldsymbol{\Theta}}), \tag{17}$$

given a fixed dictionary. To simplify notation, we henceforth omit $\hat{\boldsymbol{x}}_{\mathbf{D}}$ from L whenever $\mathbf{D}$ is fixed.

Observe that problem (17) attempts to find a process $\hat{\boldsymbol{z}}_{\boldsymbol{\Theta}}$ in the family $\mathcal{F}_{\boldsymbol{\Theta}}$ that minimizes the empirical risk over a finite set of training examples, as an approximation to the expected risk

$$\hat{\mathcal{L}}(\hat{\boldsymbol{z}}_{\boldsymbol{\Theta}}) = \frac{1}{n} \sum_{i=1}^{n} L(\mathbf{x}_i, \hat{\boldsymbol{z}}_{\boldsymbol{\Theta}}) \approx \int L(\mathbf{x}, \hat{\boldsymbol{z}}_{\boldsymbol{\Theta}}) dP(\mathbf{x}) = \mathcal{L}(\hat{\boldsymbol{z}}_{\boldsymbol{\Theta}})$$

over the data distribution P. While the empirical risk measures the encoder performance over the training set, the expected risk measures the expected performance over new data samples following the same distribution, that is, the generalization capabilities of the model. When the family $\mathcal{F}_{\boldsymbol{\Theta}}$ is sufficiently restrictive, the statistical learning theory justifies minimizing the empirical risk instead of the expected risk [38]. We will come back to this issue in Section 4.2, where we address the accuracy of the proposed encoders in approximating $\mathcal{L}(\hat{\boldsymbol{z}}_{\boldsymbol{\Theta}})$.

When the functions belonging to $\mathcal{F}_{\boldsymbol{\Theta}}$ are almost everywhere differentiable with respect to the parameters $\boldsymbol{\Theta}$, stochastic gradient descent (SGD) can be used to optimize (17), with almost sure convergence to a stationary point [39]. At each iteration, a random subset of the training data, $\{\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_r}\}$, is selected and used to produce an estimate of the (sub)-gradient of the objective function. The parameters of $\mathbf{z}_{\boldsymbol{\Theta}}$ are updated as

$$\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta} - \mu_k \frac{1}{r} \sum_{k=1}^{r} \frac{\partial L(\mathbf{x}_{i_k}, \hat{\boldsymbol{z}}_{\boldsymbol{\Theta}})}{\partial \boldsymbol{\Theta}}, \tag{18}$$

where μ_k is a decaying step, repeating the process until convergence. This requires the computation of the (sub)-gradients $\partial L / \partial \boldsymbol{\Theta}$, which is achieved by a back-propagation procedure as detailed in the sequel. SGD algorithms scale well to big data applications where the

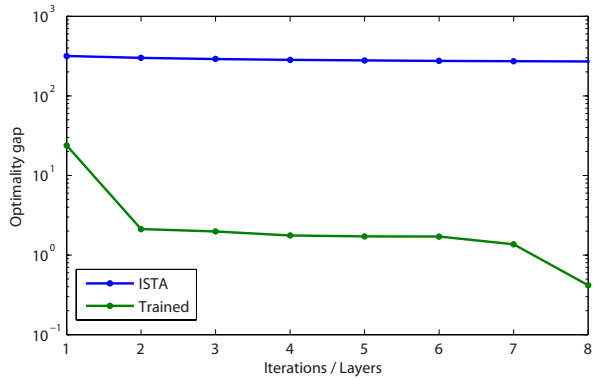

Fig. 1. Comparison between RNMF encoders and proximal methods for music source separation. Optimality gap as a function of the number of iterations/layers with offline trained encoders using the unsupervised regime

limiting factor is the computational time rather than the number of available samples [39]. We use a step size of the form $\mu_k = \min(\mu, \mu k_0/k)$, meaning that a fixed step size is used during the first k_0 iterations, after which it decays according to the $1/k$ annealing strategy. As in any standard SGD scheme, the speed of convergence depends on the choice of μ. We chose this parameter by running a few hundreds of iterations with several values of μ and keeping the one that performs best on a small validation set. In all out experiments we used $k_0 = K/10$, where K is the total number of SGD iterations. The close relation between the encoder architectures and the optimization algorithms provides a very good initialization of the parameters, which is non-trivial in most general learning scenarios.

An illustration to the advantage of the trained pursuit process is shown in Figure 1. The figure shows the optimality gap $\hat{\mathcal{L}}(\hat{\boldsymbol{z}}_{T,\boldsymbol{\Theta}}) - \hat{\mathcal{L}}(\hat{\boldsymbol{z}}^*)$ as a function of T for the truncated proximal descent ($\Theta = \Theta^{(0)}$ set as prescribed by Algorithm 2) and for the trained encoder ($\Theta = \Theta^*$). Note that at testing, after offline training of the encoders, for the same number of layers and iterations the computational complexity of the encoder and the truncated algorithms is identical. Since the encoders aim at approximating the optimization algorithm (and the problem is convex), one can always improve performance by increasing T in the truncated algorithm. It takes about 70 iterations of the proximal method to reach the error obtained by our proposed $7-$layer encoder. A further and stronger justification to the process-centric approach advocated in this paper is presented in the next section.

4.2 Approximation accuracy

Following [39], we split the process training approximation error into three terms, $\epsilon = \epsilon_{\text{app}} + \epsilon_{\text{est}} + \epsilon_{\text{opt}}$. The *approximation error* $\epsilon_{\text{app}} = \mathbb{E}\{\mathcal{L}(\hat{\boldsymbol{z}}^*_{\boldsymbol{\Theta}}) - \mathcal{L}(\hat{\boldsymbol{z}}^*)\}$ measures how well the optimal unrestricted pursuit process $\hat{\boldsymbol{z}}^*$ given by (15) is approximated by the optimal pursuit process restricted to $\mathcal{F}_{\boldsymbol{\Theta}}$, $\hat{\boldsymbol{z}}^*_{\boldsymbol{\Theta}} = \arg\min_{\hat{\boldsymbol{z}}_{\boldsymbol{\Theta}} \in \mathcal{F}_{\boldsymbol{\Theta}}} \mathcal{L}(\hat{\boldsymbol{z}}_{\boldsymbol{\Theta}})$. The *estimation error* $\epsilon_{\text{est}} = \mathbb{E}\{\mathcal{L}(\hat{\hat{\boldsymbol{z}}}^*_{\boldsymbol{\Theta}}) - \mathcal{L}(\hat{\boldsymbol{z}}^*_{\boldsymbol{\Theta}})\}$ with $\hat{\hat{\boldsymbol{z}}}^*_{\boldsymbol{\Theta}} = \arg\min_{\hat{\boldsymbol{z}}_{\boldsymbol{\Theta}} \in \mathcal{F}_{\boldsymbol{\Theta}}} \hat{\mathcal{L}}(\hat{\boldsymbol{z}}_{\boldsymbol{\Theta}})$ measures the cost of optimizing the empirical risk instead of the expected risk. Finally, the *optimization error* $\epsilon_{\text{opt}} = \mathbb{E}\{\mathcal{L}(\hat{\hat{\boldsymbol{z}}}_{\boldsymbol{\Theta}}) - \mathcal{L}(\hat{\hat{\boldsymbol{z}}}^*_{\boldsymbol{\Theta}})\}$ measures the effect of having $\hat{\hat{\boldsymbol{z}}}_{\boldsymbol{\Theta}}$ that minimizes the empirical risk only approximately.

The estimation error vanishes asymptotically with the increase of the training set size n. The optimization error can be made negligible (at least, in the offline setting) by simply increasing the number of SGD iterations. Consequently, the quality of the learned pursuit process largely depends on the choice of the family $\mathcal{F}_{\boldsymbol{\Theta}}$, which we will address in the next section. This optimality gap in Figure 1 represents an empirical approximation error, ϵ_{app}, for the corresponding spaces $\mathcal{F}_{T,\boldsymbol{\Theta}}$.

4.3 Process architecture

We extend the ideas in [15] to derive families of trainable pursuit processes from proximal methods. Let us examine a generic fixed-step proximal descent algorithm described in Section 3. Each iteration can be described as a function receiving the current state $(\mathbf{b}_{\text{in}}, \mathbf{z}_{\text{in}})$ and producing the next state $(\mathbf{b}_{\text{out}}, \mathbf{z}_{\text{out}})$ by applying the non-linear transformation $\mathbf{z}_{\text{out}} = \boldsymbol{\pi}_{\mathbf{t}}(\mathbf{b}_{\text{in}})$ (representing the proximal operator), and the linear transformation $\mathbf{b}_{\text{out}} = \mathbf{b}_{\text{in}} + \mathbf{H}(\mathbf{z}_{\text{out}} - \mathbf{z}_{\text{in}})$ (representing the linear part of the gradient). This can be described by the function $(\mathbf{b}_{\text{out}}, \mathbf{z}_{\text{out}}) = \hat{\boldsymbol{f}}_{\mathbf{H},\mathbf{t}}(\mathbf{b}_{\text{in}}, \mathbf{z}_{\text{in}})$ parametrized by the matrix $\mathbf{H}$ describing the linear transformation, and the vector $\mathbf{t}$ describing the parameters of the proximal operator $\boldsymbol{\pi}_{\mathbf{t}}$.

A generic fixed-step proximal descent algorithm can therefore be expressed as a long concatenation of such iterations, $\hat{\boldsymbol{z}}^*(\mathbf{x}) = \cdots \circ \hat{\boldsymbol{f}}_{\mathbf{H},\mathbf{t}} \circ \cdots \circ \hat{\boldsymbol{f}}_{\mathbf{H},\mathbf{t}}(\mathbf{W}\mathbf{x}, \mathbf{0})$, with the initialization $(\mathbf{b}_{\text{in}}, \mathbf{z}_{\text{in}}) = (\mathbf{W}\mathbf{x}, \mathbf{0})$. Note that matrices $\mathbf{W}$ and $\mathbf{H}$ will depend on the specific problem at hand, following the general model (1). For example, Algorithms 1 and 2 follow this structure exactly, for the appropriate choice of the parameters. Block-coordinate proximal descent algorithms operate very similarly except that the result of the application of $\hat{\boldsymbol{f}}$ is substituted to a subset

of the elements of the state vector, selected in a state-dependent manner at each iteration.

Following [15], we consider the family of pursuit processes derived from *truncated* proximal descent algorithms with T iterations, $\mathcal{F}_{T,\boldsymbol{\Theta}} = \{\hat{\boldsymbol{z}}_{T,\boldsymbol{\Theta}}(\mathbf{x}) = \hat{\boldsymbol{f}}_{\mathbf{H},\mathbf{t}} \circ \cdots \circ \hat{\boldsymbol{f}}_{\mathbf{H},\mathbf{t}}(\mathbf{W}\mathbf{x}, \mathbf{0})\}$. For convenience, we collected all the process parameters into a single pseudo-vector $\boldsymbol{\Theta} = \{\mathbf{W}, \mathbf{H}, \mathbf{t}\}$. Also note that at the last iteration, only $\mathbf{z}$ of the state vector $(\mathbf{b}, \mathbf{z})$ is retained; with some abuse of notation, we still denote the last iteration by $\hat{\boldsymbol{f}}$. Finally, we denote by $\mathcal{F}_{\infty,\boldsymbol{\Theta}}$ the family of untruncated processes.

A process $\hat{\boldsymbol{z}}_{T,\boldsymbol{\Theta}} \in \mathcal{F}_{T,\boldsymbol{\Theta}}$ can be thought of as a feed-forward neural network with identical layers $\hat{\boldsymbol{f}}_{\mathbf{H},\mathbf{t}}$. Flow diagrams of processes derived from the proximal descent Algorithms 1 and 2 for the Lasso and RPCA/RNMF problems are depicted in Figure 2. Since the processes are almost everywhere $\mathcal{C}^1$ with respect to the input $\mathbf{x}$ and the parameters $\boldsymbol{\Theta}$, the sub-gradients required in (18) are well-defined. The procedure for computing the sub-gradients of $L(\mathbf{x}, \hat{\boldsymbol{z}}_{\boldsymbol{\Theta}}(\mathbf{x}))$ with respect to $\boldsymbol{\Theta}$ is frequently referred to as back-propagation and is of standard use in the neural network literature. The concatenation of identical layers in the encoders and the chain rule of differentiation allow a simple recursive procedure starting at the output and propagating backward into the network. The procedure is detailed in Algorithm 3. We follow the standard notation from the neural network literature: the δ prefix denotes the gradient of L with respect to the variable following it, $\delta * = \partial L / \partial *$.

input : Sub-gradient $\delta\mathbf{z} = \dfrac{\partial L(\mathbf{z}^{(T)})}{\partial \mathbf{z}}$.
output: Sub-gradients of L with respect to the parameters, $\delta\mathbf{H}$, $\delta\mathbf{W}$, $\delta\mathbf{t}$; and with respect to the input, $\delta\mathbf{x}$.
Initialize $\delta\mathbf{t}^{(T)} = \dfrac{\partial \boldsymbol{\pi}(\mathbf{b}^{(T)})}{\partial \mathbf{t}}\delta\mathbf{z}$, $\delta\mathbf{b}^{(T)} = \dfrac{\partial \boldsymbol{\pi}(\mathbf{b}^{(T)})}{\partial \mathbf{b}}\delta\mathbf{z}$, $\delta\mathbf{H}^{(T)} = \mathbf{0}$, and $\delta\mathbf{z}^{(T)} = \mathbf{0}$
for $k = T-1, T-2, \ldots, 1$ **do**
$\quad \delta\mathbf{H}^{(k-1)} = \delta\mathbf{H}^{k} + \delta\mathbf{b}^{(k)}(\mathbf{z}^{(k+1)} - \mathbf{z}^{(k)})^{\mathrm{T}}$
$\quad \delta\mathbf{t}^{(k-1)} = \delta\mathbf{t}^{(k)} + \dfrac{\partial \boldsymbol{\pi}(\mathbf{b}^{(k)})}{\partial \mathbf{t}}(\mathbf{H}^{\mathrm{T}}\delta\mathbf{b}^{(k)} - \delta\mathbf{z}^{(k)})$
$\quad \delta\mathbf{z}^{(k-1)} = -\mathbf{H}^{\mathrm{T}}\delta\mathbf{b}^{(k)}$
$\quad \delta\mathbf{b}^{(k-1)} = \delta\mathbf{b}^{(k)} + \dfrac{\partial \boldsymbol{\pi}(\mathbf{b}^{(k)})}{\partial \mathbf{t}}\mathbf{H}^{\mathrm{T}}\delta\mathbf{b}^{(k)} + \dfrac{\partial \boldsymbol{\pi}(\mathbf{b}^{(k)})}{\partial \mathbf{b}}\delta\mathbf{z}^{(k)}$
end
Output $\delta\mathbf{H} = \delta\mathbf{H}^{(0)}$, $\delta\mathbf{t} = \delta\mathbf{t}^{(0)}$, $\delta\mathbf{W} = \delta\mathbf{b}^{(0)}\mathbf{x}^{\mathrm{T}}$, and $\delta\mathbf{x} = \mathbf{W}^{\mathrm{T}}\delta\mathbf{b}^{(0)}$

Algorithm 3: Computation of the sub-gradients of $L(\mathbf{x}, \hat{\boldsymbol{z}}_{T,\boldsymbol{\Theta}}(\mathbf{x}))$ for a pursuit process $\hat{\boldsymbol{z}}_{T,\boldsymbol{\Theta}} \in \mathcal{F}_{T,\boldsymbol{\Theta}}$. The reader is warned not to confuse the T-th iteration index, $\mathbf{z}^{(T)}$, with the transpose, $\mathbf{z}^{\mathrm{T}}$.

4.4 Approximation error vs. complexity trade-off

Since the objective minimized via proximal descent is convex, it is guaranteed that there exists some selection of the parameters $\boldsymbol{\Theta}^*$ such that $\lim_{T\to\infty} \hat{\boldsymbol{z}}_{T,\boldsymbol{\Theta}^*} = \hat{\boldsymbol{z}}^*$. This can be obtained simply by setting $\boldsymbol{\Theta}^*$ to match the parameters given by the exact optimization algorithm. Since the number of iterations is infinite, it will reach convergence for any input vector. In other words, the optimal process $\hat{\boldsymbol{z}}^*$ is contained in $\mathcal{F}_{\infty,\boldsymbol{\Theta}}$. Furthermore, reformulating the non-asymptotic convergence analysis from [18] (Theorem 3.1 in [18]) in our notation, the following holds:

Theorem 1 (Beck&Teboulle)**.** *For every* $\mathbf{D}$*, there exists* $\boldsymbol{\Theta}^*$ *and* $C > 0$ *such that for every* $\mathbf{x}$ *and* $T \geq 1$*,* $L(\mathbf{x}, \hat{\boldsymbol{z}}_{T,\boldsymbol{\Theta}^*}) - L(\mathbf{x}, \hat{\boldsymbol{z}}^*) \leq \dfrac{C}{2T}\|\hat{\boldsymbol{z}}^*(\mathbf{x})\|_2^2$.

This result is worst-case, in the sense that it holds for every input vector $\mathbf{x}$. Assuming bounded support of the input distribution and taking expectation with respect to it, we obtain the following:

Corollary 1. *There exist* $\boldsymbol{\Theta}^*$ *and* $C > 0$ *such that for* $T \geq 1$*,* $\epsilon_{\mathrm{app}} = \mathbb{E}\{\mathcal{L}(\hat{\boldsymbol{z}}_{T,\boldsymbol{\Theta}^*}) - \mathcal{L}(\hat{\boldsymbol{z}}^*)\} \leq \dfrac{C}{2T}$.

In other words, the family $\mathcal{F}_{T,\boldsymbol{\Theta}}$ of pursuit processes allows to set the approximation error to an arbitrarily small number by increasing T. There is, however, a trade-off between the approximation error, ϵ_{app}, and the computational complexity of the encoder, which is proportional to T. This is shown in the experiment shown in Figure1. The bound in Theorem 1 is uniform for every input $\mathbf{x}$, independently of the input distribution P, while in practice we would strive for a much faster decrease of the error on more probable inputs. This implies that the bound in Corollary 1 is by no means tight, and there might be other selections of $\boldsymbol{\Theta}$ giving much lower approximation error at a fixed T. Note that these improvements come at the expense of increasing the number of parameters and the offline computational load associated with their training. While pursuit processes optimal in the sense of the expected risk can be found using the process learning approach, there is no simple way for an iterative pursuit algorithm to take the input data distribution into account.

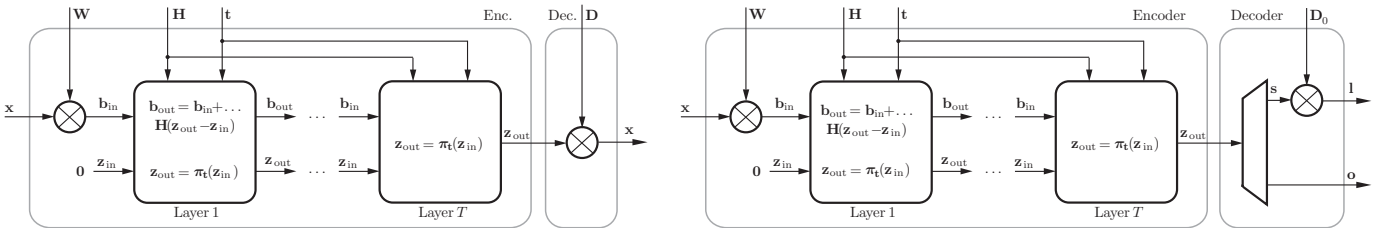

Fig. 2. Encoder/decoder architectures for the unstructured Lasso (left), and the robust PCA/NMF (right).

5 Training Regimes

As it was described in the previous section, parsimonious modeling can be interpreted as training an encoder/decoder pair $(\hat{\boldsymbol{z}}, \hat{\boldsymbol{x}})$ that minimizes the empirical loss $\hat{\mathcal{L}}$ in problem (14). We refer to this setting as *unsupervised*, as no information beside the data samples themselves is provided at training. An important characteristic of this training regime is that it can be performed online, combining online adaptation of $\boldsymbol{\Theta}$ with online dictionary update as described in Section 2. The online adaptation requires some extra computations to update the network parameters.

In many applications, one would like to train the parsimonious model comprising the encoder/decoder pair to optimally perform a specific task, e.g., source separation, object recognition, or classification. This setting can be still viewed as the solution of the modeling problem (14), with the modification of the training objective L to include the task-specific knowledge. However, for most objectives, given a fixed linear decoder $\hat{\boldsymbol{x}}_{\mathbf{D}}(\mathbf{z}) = \mathbf{D}\mathbf{z}$, the encoder of the form

$$\hat{\boldsymbol{z}}(\mathbf{x}) \quad = \quad \arg\min_{\mathbf{z}\in\mathbb{R}^q} \frac{1}{2}\|\mathbf{x} - \mathbf{D}\mathbf{z}\|_{\mathrm{F}}^2 + \psi(\mathbf{z}) \qquad (19)$$

is no more optimal, in the sense that it is not the solution of (14) with the fixed decoder.

Trainable pursuit processes provide an advantageous alternative in these cases. They have fixed complexity and latency controllable through the parameter T, and they are expected to achieve a better complexity-approximation error trade-off than the iterative pursuit processes. These trainable processes have the structure of the iterative pursuit built into the architecture on one hand, while leaving tunable degrees of freedom that can improve their modeling capabilities on the other.

In the sequel, we exemplify various training regimes (i.e., different objectives in (14)) that can be used in combination with the architectures described in Section 4.3, leading to a comprehensive framework of process-centric parsimonious models.

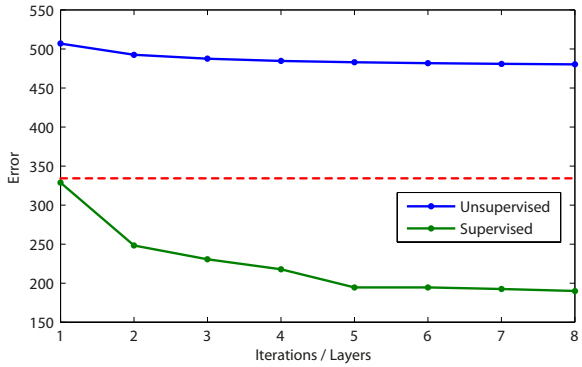

Fig. 3. Comparison between RNMF encoders and proximal methods for music source separation. The ℓ_2 separation error obtained with offline supervised and the unsupervised training. The red dotted line represents the error achieved by Algorithm 2 after convergence.

5.1 Supervised learning

In many applications, parsimonious models are used as a first order approximation to various classes of natural signals. An example is the music separation problem discussed in Section 4.1. While achieving excellent separation results, both the RPCA and RNMF models are merely a crude representation of the reality: the music is never exactly low-rank, as well as the voice does not exactly admit the naïve unstructured sparse model. We propose to fill the gap between the parsimonious models and the more sophisticated (and, hence, prone to errors) domain-specific models, by incorporating learning. This can be done by designing the objective function L of (14) to incorporate the domain-specific information.

Sticking to the example of audio source separation by means of online robust PCA or NMF, let us be given a collection of n data vector, $\mathbf{x}_i$, representing the short-time spectrum of the mixtures, for which the clean ground-truth accompaniment $\mathbf{l}_i^*$ and voice $\mathbf{o}_i^*$ spectra are given. We aim at finding an RPCA/RNMF encoder $(\mathbf{s}; \mathbf{o}) = \hat{\boldsymbol{z}}_{\boldsymbol{\Theta}}(\mathbf{x})$ with the architecture depicted in Figure 2, and the linear decoder $(\mathbf{l}, \mathbf{o}) = \mathbf{D}(\mathbf{s}; \mathbf{o})$ parametrized by

$\mathbf{D} = (\mathbf{D}_0, \mathbf{I}_{m\times m})$, that minimize (14) with the objective

$$\hat{\mathcal{L}}(\mathbf{z}_{\boldsymbol{\Theta}}, \mathbf{D}) = \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{D}_0\hat{\boldsymbol{s}}_{\boldsymbol{\Theta}}(\mathbf{x}_i) - \mathbf{l}_i^*\|_2^2 + \|\hat{\boldsymbol{o}}_{\boldsymbol{\Theta}}(\mathbf{x}_i) - \mathbf{o}_i^*\|_2^2 + \frac{\lambda_*}{2}\left(\|\mathbf{D}_0\|_2^2 + \|\hat{\boldsymbol{s}}_{\boldsymbol{\Theta}}(\mathbf{x}_i)\|_{\mathrm{F}}^2\right) + \lambda\|\hat{\boldsymbol{o}}_{\boldsymbol{\Theta}}(\mathbf{x}_i)\|_1. \quad (20)$$

Note that the architecture and the initial parameters of the encoder are already a very good starting point, yet the supervised training allows it to better model the reality that is not fully captured by the RPCA/RNMF model. Figure 3 shows this phenomenon (see Section 6 for further details on the experimental setting). The encoders trained with the supervised setting outperform not only their unsupervisedly trained counterparts, but also the optimization based RPCA/RNMF algorithm. Note that unlike the experiment in Figure 1, we do not want to approximate the solution of the algorithms but to solve a different task: a source separation problem. This illustrates the generality of the proposed framework.

In a broader perspective, the sound separation example can be viewed as a particular instance of *supervised* encoder/decoder training regimes, in which the loss L is expressed in terms of the composite output of $\hat{\boldsymbol{x}} \circ \hat{\boldsymbol{z}}$, but is supervised by some $\mathbf{y}_i$ different from the input data, e.g.,

$$\hat{\mathcal{L}}(\hat{\boldsymbol{z}}, \hat{\boldsymbol{x}}) = \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{y}_i - \hat{\boldsymbol{x}} \circ \hat{\boldsymbol{z}}(\mathbf{x}_i)\|_2^2 + \psi(\hat{\boldsymbol{z}}(\mathbf{x}_i)) + \phi(\hat{\boldsymbol{x}}). \quad (21)$$

Compare this to the unsupervised setting, where essentially $\mathbf{y}_i = \mathbf{x}_i$, and the performance of the encoder/decoder pair is measured by their ability to *reconstruct* the input.

The idea of [15] to train pursuit processes to approximate the output of iterative pursuit algorithms falls into this category. For each training data vector $\mathbf{x}_i$, let $\mathbf{z}_i^* = \hat{\boldsymbol{z}}^*(\mathbf{x}_i)$ be the output of the optimal encoder (15). Setting the decoder to the identity map, $\hat{\boldsymbol{x}} = \hat{\boldsymbol{id}}$ in (21) reduces the supervision to the encoder outputs, $\hat{\mathcal{L}}(\hat{\boldsymbol{z}}) = \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{z}_i - \hat{\boldsymbol{z}}(\mathbf{x}_i)\|_2^2$. While producing good results in practice, under the limiting factor that the testing data is from the same class as the training data, this training regime requires supervision yet it is unable to improve the modeling capabilities of the encoder beyond those of the underlying parsimonious model (unlike the supervision of the decoder outputs in (21)). On the other hand, we show in Section 6 that similar performance can be achieved by using unsupervised training. We refer to this regime as *Approximation*.

5.2 Discriminative learning

The supervised training setting is also useful to extend parsimonious models beyond the conventional *generative* scenario, in which the data can be approximately recovered from the representation, to the *discriminative* scenario, such as classification problems, where the representation is typically non-invertible and is intended to capture various invariant properties of the data.

As an illustration, we use the activity recognition problem from [40]. A collection of action-specific dictionaries, $\mathbf{D}_1, \ldots, \mathbf{D}_k$, is trained offline, and the models are fit to previously unobserved data. The lowest fitting error is then used to assign the speaker identity. See Section 6.1 for experimental evaluation with real data.

Using the process-centric methodology, we construct k encoders $(\hat{\boldsymbol{s}}_{\boldsymbol{\Theta}_j}, \hat{\boldsymbol{o}}_{\boldsymbol{\Theta}_j})(\mathbf{x})$, and k corresponding decoders $(\mathbf{D}_0\mathbf{s}, \mathbf{D}_j\mathbf{o})$ with the shared noise dictionary $\mathbf{D}_0$. The encoder/decoder pairs are trained by minimizing an empirical loss of the form

$$L(\mathbf{x}, l, \Theta_1, \ldots, \Theta_k, \mathbf{D}_0, \ldots, \mathbf{D}_k) = \|\mathbf{x} - \mathbf{D}_0\hat{\boldsymbol{s}}_{\boldsymbol{\Theta}_l}(\mathbf{x}) - \mathbf{D}_l\hat{\boldsymbol{o}}_{\boldsymbol{\Theta}_l}(\mathbf{x})\|_2^2 + \sum_{j\neq l} \max\left\{0, \epsilon - \|\mathbf{x} - \mathbf{D}_0\hat{\boldsymbol{s}}_{\boldsymbol{\Theta}_j}(\mathbf{x}) - \mathbf{D}_j\hat{\boldsymbol{o}}_{\boldsymbol{\Theta}_j}(\mathbf{x})\|_2^2\right\}, \quad (22)$$

averaged on the training set containing examples of noisy speech, $\mathbf{x}$, and the corresponding labels, $l \in \{1, \ldots, k\}$ indicating which speaker is present in the sample. The loss function (22) promotes low fitting error for the encoder from the class coinciding with the ground-truth class, while favoring high fitting error for the rest of the encoders. The hinge parameter ϵ counters excessive influence of the negatives.

5.3 Data transformations

Parsimonious models rely on the assumption that the input data vectors are "aligned" with respect to each other. This assumption might be violated in many applications. A representative example is face modeling via RPCA, where the low dimensional model only holds if the facial images are pixel-wise aligned [41]. Even small misalignments can break the structure in the data; the representation then quickly degrades as the rank of the low dimensional component increases and the matrix of outliers loses its sparsity. In [41], it was proposed to simultaneously align the input vectors and solve RPCA by including the transformation parameters into the optimization variables. This problem is highly non-convex, yet if a good initialization of the transformation

parameters is available, a solution can be found by solving a sequence of convex optimization problems, each of them being comparable to (6).

Following [41], we propose to incorporate the optimization over geometric transformations of the input data into our modeling framework. We assume that the data are known to be subject to a certain class of parametric transformations $\mathcal{T} = \{\hat{\boldsymbol{g}}_{\boldsymbol{\alpha}} : \mathbb{R}^m \rightarrow \mathbb{R}^m : \boldsymbol{\alpha}\}$. Given a data matrix $\mathbf{X}$, we search for its best alignment together with finding the best model by solving

$$\min_{\substack{\mathbf{z}_{\boldsymbol{\Theta}}, \hat{\mathbf{x}}_{\mathbf{D}} \\ \boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_n}} \frac{1}{n} \sum_{i=1}^{n} L(\hat{\boldsymbol{g}}_{\boldsymbol{\alpha}_i}(\mathbf{x}_i), \hat{\boldsymbol{z}}_{\boldsymbol{\Theta}}, \hat{\boldsymbol{x}}_{\mathbf{D}}), \tag{23}$$

jointly optimizing over the encoder $\hat{\boldsymbol{z}}_{\boldsymbol{\Theta}}$, possibly the decoder $\hat{\boldsymbol{x}}_{\mathbf{D}}$, and the transformation parameters $\boldsymbol{\alpha}_i$ of each of the training vectors. This approach can be used with any of the training objectives described before.

Problem (23) can be carried out as a simple extension of our general process training scheme. Transformation parameters $\boldsymbol{\alpha}$ can be treated on par with the encoder parameters $\boldsymbol{\Theta}$; note that since the encoder functions $\hat{\boldsymbol{z}}_{\boldsymbol{\Theta}} \in \mathcal{F}_{\boldsymbol{\Theta}}$ are by construction almost everywhere differentiable with respect to their input, we can find the subgradients of the composition $\hat{\boldsymbol{z}}_{\boldsymbol{\Theta}}(\hat{\boldsymbol{g}}_{\boldsymbol{\alpha}}(\mathbf{x}))$ with respect to $\boldsymbol{\alpha}$ by simply applying the chain rule. This allows to solve (23) using SGD and the back-propagation in Algorithm 3.

The obtained encoders are conceptually very similar to the ones we had before and can still be trained in an online manner. In this particular setting, our framework presents a significant advantage over existing approaches based on iterative pursuit. In general, as the new data arrive and the model is refined or adapted, the alignment of the previously received input vectors has to be adjusted. In our settings, there is no need to recompute the alignment for the whole dataset from scratch, and it can be adjusted via online SGD. While differently motivated, in this setting, our framework is related to the sparse autoencoders [28].

Unlike the untransformed model, the pursuit is no more given by an explicit function $\hat{\boldsymbol{z}}_{\boldsymbol{\Theta}}$, but requires the solution of a simple optimization problem in $\boldsymbol{\alpha}$, $\boldsymbol{\alpha}^* = \arg\min_{\boldsymbol{\alpha}} L(\mathbf{x}, \hat{\boldsymbol{z}}_{\boldsymbol{\Theta}} \circ \hat{\boldsymbol{g}}_{\boldsymbol{\alpha}}, \hat{\boldsymbol{x}}_{\mathbf{D}})$. A new data vector $\mathbf{x}$ is then encoded as $\mathbf{z} = \hat{\boldsymbol{z}}_{\boldsymbol{\Theta}}(\hat{\boldsymbol{g}}_{\boldsymbol{\alpha}^*}(\mathbf{x}))$.

6 Experimental results

In what follows, we describe experiments conducted to assess the efficiency of the proposed framework. Fast encoders were implemented in Matlab with built-in GPU acceleration and executed on a multi-Intel Xeon E5620 CPU (total 32 threads) and NVIDIA Tesla C2070 GPU. When referring to a particular encoder, we specify its architecture and the training regime; for example "CoD (Unsupervised)" stands for the CoD network trained in the unsupervised regime. We denote by *Approximation*, *Supervised*, *Unsupervised* and *Discriminative* the corresponding training regime as defined in Section 5. *Untrained* denotes an untrained encoder, that is, with parameters set as in the corresponding fixed-step proximal descent algorithm. The results obtained when running the proximal descent algorithms untill convergence are denoted by the prefix "Opt" (i.e., "Opt RNMF"), since these are optimization-based (model-centric) approaches.

TABLE 1
Action classification rates and average computational costs for USC-sport dataset.

	SM-1	70-iter FISTA	ISTA-Encoders (*Unsup.*)
Accuracy	98%	92%	95.5%
Comp. time	23.86 s	1.52 s	0.26 s

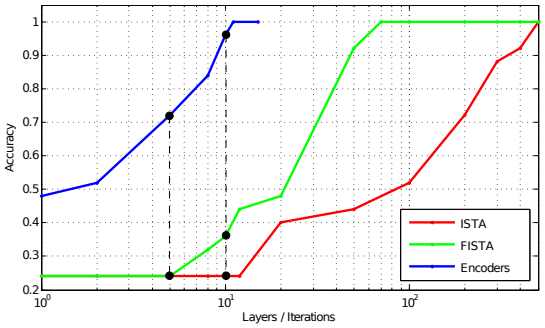

Fig. 4. Comparison between the learned ISTA-encoders and standard ISTA and FISTA on the activity recognition experiment. Depicted is the classification performance as a function of the number of iterations/layers with offline trained encoders in the unsupervised regime. Vertical lines compare indicate fixed computational budget.

6.1 Running time comparison

The objective of this experiment is to compare the computational efficiency of the proposed encoders. As a test case, we use the classification of human actions in video [40]. First, spatio-temporal features characterizing local changes in the frames are extracted. Then, for each action of interest, an individual dictionary is learned representing the associated spatio-temporal features as a non-negative sparse linear combination of the learned atoms. Finally, these dictionaries are combined into a single class-structured dictionary. At testing, the spatio-

temporal features of the videos are coded solving a non-negative Lasso problem. The coefficients associated to each class are summed over all feature vectors, and the video is assigned to the class with the highest score. In this experiment we omit the second stage of the algorithm proposed in [40], as we want to isolate a simple case to asses the computational benefits of using the proposed trainable encoders on a real application.

We used the UCF-Sports dataset, consisting of 150 videos with 10 different action classes [40]. In the first experiment, we select a random representative sample of 25 videos for testing and use the remaining ones for training. Figure 4 depicts how the performance as a function of the number of iterations of ISTA and FISTA and the number of layers in the trained ISTA-encoders. Note that while each ISTA iteration has exactly the same computational cost as a layer in the learned encoders, FISTA iterations are slightly more expensive. We observe that the encodres are about an order of magnitude faster than FISTA and about two orders of magnitude faster than ISTA. More importantly, the encoders significantly outperform the truncated algorithms for a fixed computational budget (marked with black vertical lines in Figure 4). With the budget equivalent to 10 iterations, the encoders, FISTA and ISTA achieve an accuracy of 96%, 35 % and 22% respectively.

The USC dataset is normally evaluated using leave-one-out cross validation [40]. The one-stage algorithm in [40], referred to as SM-1, achieves state-of-the-art performance with 10% improvement over its closest competitor (see Table 4 in [40]). In Table 1 we compare the performance and the execution time of SM-1 solved using the SPAMS package [31], 70 iterations of FISTA, and ISTA (*Unsupervised*) encoders with $T = 10$ learned layers, trained offline with the non-negative Lasso cost function. We observe that the networks achieve similar performance but are significantly faster than the other alternatives. While the encoders and FISTA were implemented in Matlab with GPU acceleration, SPAMS implementing the LARS algorithm is a highly-optimized, multi-thread package implemented in C++. The GPU runtime results do not include the data transfer to the GPU memory, which is an order of magnitude longer than the computation itself.

6.2 Online sparse encoders

To evaluate the performance of the unstructured CoD (*Unsupervised*) encoders in the online learning regime, we

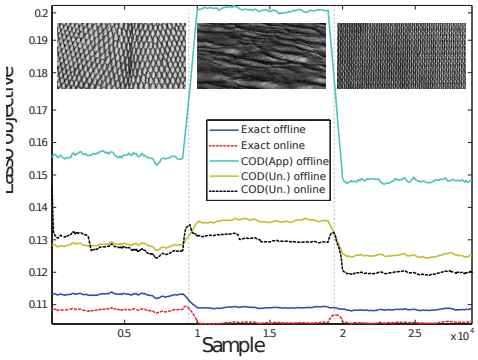

Fig. 5. Performance of CoD sparse encoders, trained under different training regimes, measured using the Lasso objective, as a function of sample number in the online learning experiment. Shown are the three groups of patches corresponding to different texture images from the Brodatz dataset.

used 30×10^4 randomly located 8×8 patches from three images from the Brodatz texture dataset [42]. The patches were ordered in three consecutive blocks of 10^4 patches from each image. Dictionary size was fixed to $q = 64$ atoms, and $T = 4$ layers were used in all CoD encoders. Unsupervised online learning was performed using the Lasso objective with $\lambda = 1$ on overlapping windows of $1{,}000$ vectors with a step of 100 vectors. Online trained encoders were compared to an online version of Lasso with the dictionary adapted on the same data. As reference, we trained offline a CoD (*Approximation*) encoder and another CoD (*Unsupervised*) encoder. Offline training was performed on a distinct set of $6{,}000$ patches extracted from the same images.

Performance measured in terms of the Lasso objective is reported in Figure 5 (high error corresponds to low performance). Initially, the performance of the online trained CoD (*Unsupervised*) encoder is slightly inferior to the offline trained counterpart; however, the online version starts performing better after the network parameters and the dictionary adapt to the current class of data. The CoD (*Approximation*) encoder trained offline exhibits the lowest performance. This experiment shows that, while the drop in performance compared to the "Opt Lasso" is relatively low, the computational complexity of the online CoD (*Unsupervised*) encoder is tremendously lower and fixed.

6.3 Structured sparse encoders

In this section we investigate the relevance of the architecture used in the encoder. The idea is to asses the effect of matching architecture and objective function on

TABLE 2
Speaker identification misclassification rates.

Code	Error rate
Opt HiLasso	2.35%
CoD HiLasso (*Approximation*)	6.08%
BCoD (*Approximation*)	3.53%

the success of the framework. We use a speaker identification task reproduced from [43]. In this application the authors use hierarchical sparse coding to automatically detect the speakers in a given mixed signal. The dataset consists of recordings of five different radio speakers, two females and three males. 25% of the samples were used for training, and the rest for testing. Within the testing data, two sets of waveforms were created: one containing isolated speakers, and another containing all possible combinations of mixtures of two speakers. Signals were decomposed into a set of overlapping time frames of 512 samples with 75% overlap, such that the properties of the signal remain stable within each frame. An 80-dimensional feature vector is obtained for each audio frame as its short-time power spectrum envelope (refer to [43] for details). Five under-complete dictionaries with 50 atoms each were trained on the single speaker set minimizing the Lasso objective with $\lambda = 0.2$ (one dictionary per speaker), and then combined into a single structured dictionary containing 250 atoms. Increasing the dictionary size exhibited negligible performance benefits. Speaker identification was performed by first encoding a test vector in the structured dictionary and measuring the ℓ_2 energy of each of the five groups. Energies were sum-pooled over 500 time samples selecting the labels of the highest two.

To assess the importance of the process architecture, a CoD (*Approximation*) and a BCoD (*Approximation*) encoders with $T = 2$ layers were trained offline to approximate the solution of the Opt HiLasso, with $\lambda_2 = 0.05$ (regularizer parameter in the lower level of the hierarchy). Note that the framework would use the CoD architecture with the Lasso objective instead, to distinguish from that (standard) setting, we denote the method CoD HiLasso (*Approximation*). Table 2 summarizes the obtained misclassification rates. The results show that BCoD encoders following the same structure as the constructed dictionary perform significantly better than encoders using the "correct" objective function but with the CoD architecture, agnostic to the structure in the problem.

6.4 Encoders with geometric optimization

In order to evaluate the performance of robust PCA encoders, we used a face dataset consisting of 800 66×48 images of a female face photographed over the timespan of 4.5 years, roughly pose- and scale-normalized and aligned. The images manifested a significant variety of hair and dressing styles, while keeping similar facial traits. Following [41], we use RPCA to decompose a collection of faces represented as columns of the data matrix $\mathbf{X}$ into $\mathbf{L} + \mathbf{O}$, with the low-rank term $\mathbf{L} = \mathbf{D}_0\mathbf{S}$ approximating the face identity (atoms of $\mathbf{D}_0$ can be thought of as "*eigenfaces*"), while the outlier term $\mathbf{O}$ capturing the appearance variability.

We trained a five layer RPCA (*Unsupervised*) encoder on 600 images from the faces dataset. The dictionary was initialized using standard SVD and parameters were set to $q = 50$, $\lambda_* = 0.1$ and $\lambda = 10^{-2}$. To evaluate the representation capabilities of RPCA encoders in the presence of geometric transformations, as a test set, we used the remaining 200 faces, as well as a collection of geometrically transformed images from the same test set. Sub-pixel planar translations were used for geometric transformations. The encoder was applied to the misaligned set, optimizing the unsupervised objective over the transformation parameters. For reference, the encoder was also applied to the transformed and the untransformed test sets without performing optimization. Examples of the obtained representations are visualized in Figure 6. Note the relatively larger magnitude and the bigger active set of the sparse outlier vector $\mathbf{o}$ produced for the misaligned faces, and how they are re-aligned when optimization over the transformation is allowed. Since the original data are only approximately aligned, performing optimal alignment during encoding frequently yields lower cost compared to the plain encoding of the original data.

6.5 Robust NMF encoders

We now evaluate the source separation problem (singing-voice/background-accompaniment), described in Section 4.4. The separation performance was evaluated on the MIR-1K dataset [44], containing $1{,}000$ Chinese karaoke clips performed by amateur singers. The experimental settings closely followed that of [45], to which the reader is referred for further details. As the evaluation criteria, we used the *normalized source-to-distortion ratio* (NSDR) from the BSS-EVAL metrics [46], averaged over the test set. Encoders with RNMF

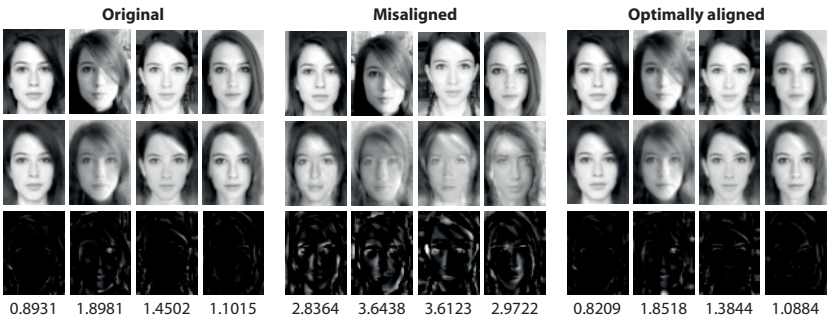

Fig. 6. Robust PCA representation of the faces dataset in the presence of geometric transformations (misalignment). Left group: original faces; middle group: shifted faces; right group: faces optimally realigned during encoding. First row: reconstructed face $\mathbf{Ds}+\mathbf{o}$; middle row: its low-rank approximation $\mathbf{Ds}$; and bottom row: sparse outlier $\mathbf{o}$. RPCA loss is given for each representation.

TABLE 3
Audio separation quality (dB SDR).

Method	[45]	Opt RNMF	RNMF encoders	
			(*Untrained*)	(*Sup.*)
SDR (dB)	3.15	2.93	2.73	4.96

architecture composed by $T = 20$ layers and $q = 25$ were trained using different training regimes. We used $\lambda = \sqrt{2n}\sigma$ and $\lambda_* = \sqrt{2}\sigma$ with $\sigma = 0.3$ set following [11]. Table 3 summarizes the obtained separation performance. While (*Unsupervised*) training regime makes fast RNMF encoders on par with the Opt RNMF (at a fraction of the computational complexity and latency of the latter), significant improvement is achieved by using the (*Supervised*) regime, where the encoders are trained to approximate the ground-truth separation over a reduced training set. For further details refer to [30].

7 Conclusions and Future Work

In this work we have developed a comprehensive framework for process-centric parsimonious modeling. By combining ideas from convex optimization with multilayer neural networks, we have shown how to produce deterministic functions capable of faithfully approximating the optimization-based solution of parsimonious models at a fraction of the computational time. Furthermore, at almost the same computational cost, the framework includes different objective functions that allow the encoders to be trained in a discriminative fashion or solve challenging alignment problems. We conducted empirical experiments in different settings and real applications such as image modeling, robust face modeling, audio sources separation and robust speaker recognition. A simple unoptimized implementation already achieves often several order of magnitude speedups when compared to exact solvers.

While we limited our attention to synthesis models, the proposed framework can be naturally extended to *analysis* cosparse models, in which the signal is known to be sparse in a transformed domain. This is subject of our current work, see [47] and references therein.

References

[1] S. Chen, D. Donoho, and M. Saunders, "Atomic decomposition by basis pursuit," *SIAM J. Scientific Computing*, vol. 20, no. 1, pp. 33–61, 1999.

[2] R. Tibshirani, "Regression shrinkage and selection via the LASSO," *J. Royal Stat. Society: Series B*, vol. 58, no. 1, pp. 267–288, 1996.

[3] B. Olshausen and D. J. Field, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature*, vol. 381, no. 6583, pp. 607–609, 1996.

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

[5] M. Yuan and Y. Lin, "Model selection and estimation in regression with grouped variables," *J. Royal Stat. Society, Series B*, vol. 68, pp. 49–67, 2006.

[6] L. Jacob, G. Obozinski, and J. Vert, "Group lasso with overlap and graph lasso," in *ICML*, 2009, pp. 433–440.

[7] P. Zhao, G. Rocha, and B. Yu, "The composite absolute penalties family for grouped and hierarchical variable selection," *Annals of Statistics*, vol. 37, no. 6A, p. 3468, 2009.

[8] R. Jenatton, J. Mairal, G. Obozinski, and F. Bach, "Proximal methods for hierarchical sparse coding," *Journal of Machine Learning Research*, vol. 12, pp. 2297–2334, 2011.

[9] P. Sprechmann, I. Ramírez, G. Sapiro, and Y. C. Eldar, "C-hilasso: A collaborative hierarchical sparse modeling framework," *IEEE Trans. Signal Process.*, vol. 59, no. 9, pp. 4183–4198, 2011.

[10] N. Srebro and A. Shraibman, "Rank, trace-norm and max-norm," *Proc. COLT*, pp. 599–764, 2005.

[11] E. Candès, X. Li, Y. Ma, and J. Wright, "Robust principal component analysis?" *Journal of the ACM*, vol. 58, no. 3, 2011.

[12] H. Xu, C. Caramanis, and S. Sanghavi, "Robust PCA via outlier pursuit," *IEEE Trans. Inf. Theory*, vol. 58, no. 5, pp. 3047–3064, 2012.

[13] L. Zhang, Z. Chen, M. Zheng, and X. He, "Robust non-negative matrix factorization," *Frontiers of Electrical and Electronic Engineering in China*, vol. 6, no. 2, pp. 192–200, 2011.

[14] D. Lee and H. Seung, "Learning parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, 1999.

[15] K. Gregor and Y. LeCun, "Learning fast approximations of sparse coding," in *ICML*, 2010, pp. 399–406.

[16] B. Recht and C. Ré, "Parallel stochastic gradient algorithms for large-scale matrix completion," *Optimization Online*, 2011.

[17] I. Daubechies, M. Defrise, and C. De Mol, "An iterative thresholding algorithm for linear inverse problems with a sparsity constraint," *Communications on Pure and Applied Mathematics*, vol. 57, no. 11, pp. 1413–1457, 2004.
[18] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM J. Img. Sci.*, vol. 2, pp. 183–202, March 2009.
[19] Y. Li and S. Osher, "Coordinate descent optimization for ℓ_1 minimization with application to compressed sensing; a greedy algorithm," *Inverse Problems and Imaging*, vol. 3, pp. 487–503, 2009.
[20] Y. Nesterov, "Gradient methods for minimizing composite functions," *Mathematical Programming*, pp. 1–37, 2012.
[21] D. Bertsekas, "Nonlinear programming," 1999.
[22] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski, "Convex optimization with sparsity-inducing norms," in *Optimization for Machine Learning*. MIT Press, 2011.
[23] J.-F. Cai, E. J. Candès, and Z. Shen, "A singular value thresholding algorithm for matrix completion," *SIAM J. on Opt.*, vol. 20, no. 4, pp. 1956–1982, 2010.
[24] Y. Mu, J. Dong, X. Yuan, and S. Yan, "Accelerated low-rank visual recovery by random projection," in *CVPR*, 2011, pp. 2609–2616.
[25] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *CVPR*, 2009, pp. 2146–2153.
[26] K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "Fast inference in sparse coding algorithms with applications to object recognition," *arXiv:1010.3467*, 2010.
[27] G. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.
[28] I. Goodfellow, Q. Le, A. Saxe, H. Lee, and A. Y. Ng, "Measuring invariances in deep networks," in *In NIPS*, 2009, pp. 646–654.
[29] P. Sprechmann, A. M. Bronstein, and G. Sapiro, "Learning efficient structured sparse models," in *ICML*, 2012.
[30] ——, "Real-time online singing voice separation from monaural recordings using robust low-rank modeling," in *ISMIR*, 2012.
[31] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online dictionary learning for sparse coding," in *ICML*, 2009, pp. 689–696.
[32] I. T. Jolliffe, *Principal Component Analysis*, 2nd ed. Springer, 2002.
[33] G. Mateos and G. B. Giannakis, "Robust PCA as bilinear decomposition with outlier-sparsity regularization," *arXiv.org:1111.1788*, 2011.
[34] M. Mardani, G. Mateos, and G. B. Giannakis, "Unveiling network anomalies in large-scale networks via sparsity and low rank," in *Proc. of 44th Asilomar Conf. on Signals, Systems, and Computers*, 2011.
[35] H. Fawzi and P. A. Parrilo, "New lower bounds on nonnegative rank using conic programming," *arXiv preprint arXiv:1210.6970*, 2012.
[36] J. Friedman, T. Hastie, and R. Tibshirani, "A note on the group lasso and a sparse group lasso," 2010, preprint.
[37] P. Tseng, "Convergence of a block coordinate descent method for nondifferentiable minimization," *J. Optim. Theory Appl.*, vol. 109, no. 3, pp. 475–494, June 2001.
[38] V. Vapnik and A. Chervonenkis, "On the uniform convergence of relative frequencies of events to their probabilities," *Theory of Probability & Its Applications*, vol. 16, no. 2, pp. 264–280, 1971.
[39] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *COMPSTAT*, August 2010, pp. 177–187.
[40] A. Castrodad and G. Sapiro, "Sparse modeling of human actions from motion imagery," *International journal of computer vision*, vol. 100, no. 1, pp. 1–15, 2012.
[41] Y. Peng, A. Ganesh, J. Wright, W. Xu, and Y. Ma, "RASL: Robust alignment by sparse and low-rank decomposition for linearly correlated images," in *CVPR*, 2010, pp. 763–770.
[42] T. Randen and J. H. Husoy, "Filtering for texture classification: a comparative study," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 21, no. 4, pp. 291–310, 1999.
[43] P. Sprechmann, I. Ramirez, P. Cancela, and G. Sapiro, "Collaborative sources identification in mixed signals via hierarchical sparse modeling," in *Proc. ICASSP*, May 2011.
[44] C. Hsu and J. Jang, "On the improvement of singing voice separation for monaural recordings using the MIR-1K dataset," *IEEE Trans. on Audio, Speech, and Lang. Proc.*, vol. 18, no. 2, pp. 310–319, 2010.
[45] P.-S. Huang, S. Chen, P. Smaragdis, and M. Hasegawa-Johnson, "Singing-voice separation from monaural recordings using robust principal component analysis," in *ICASSP*, 2012.
[46] E. Vincent, R. Gribonval, and C. Févotte, "Performance measurement in blind audio source separation," *IEEE Trans. on Audio, Speech, and Lang. Proc.*, vol. 14, no. 4, pp. 1462–1469, 2006.
[47] P. Sprechmann, R. Littman, T. BenYakar, A. Bronstein, and G. Sapiro, "Supervised sparse analysis and synthesis operators," *accepted to NIPS*, 2013.

Pablo Sprechmann (S'09) received the E.E. and the M.Sc. from the Universidad de la Repblica, Uruguay, in 2007 and 2009 respectively, and the Ph.D. from the University of Minnesota in 2012. He is currently a postdoctoral associate at the ECE department of Duke University. His main research interests lie in the areas of signal processing, machine learning, and their application to computer vision, audio processing and music information retrieval.

Alex Bronstein is a professor in the School of Electrical Engineering at Tel Aviv University. His main research interests are theoretical and computational methods in metric geometry and their application to problems in computer vision, pattern recognition, shape analysis, computer graphics, image processing, and machine learning. In addition to his academic activities, Alex Bronstein is an active technologist. His technology has been in the foundation of several successful startup companies.

Guillermo Sapiro (M94, SM03) graduated from the Technion. After post-doctoral research at MIT and 3 years at HP Labs, he was with the University of Minnesota. Currently he is with Duke University. G. Sapiro was awarded the Office of Naval Research Young Investigator Award, the Presidential Early Career Awards for Scientist and Engineers (PECASE), the National Science Foundation Career Award, and the National Security Science and Engineering Faculty Fellowship. He received the test of time award at ICCV 2011, G. Sapiro is a fellow of IEEE and SIAM, and the founding Editor-in-Chief of the SIAM Journal on Imaging Sciences.