Noname manuscript No.
(will be inserted by the editor)

Dictionary learning for fast classification based on soft-thresholding

Alhussein Fawzi · Mike Davies · Pascal Frossard

Received: date / Accepted: date

Abstract Classifiers based on sparse representations have recently been shown to provide excellent results in many visual recognition and classification tasks. However, the high cost of computing sparse representations at test time is a major obstacle that limits the applicability of these methods in large-scale problems, or in scenarios where computational power is restricted. We consider in this paper a simple yet efficient alternative to sparse coding for feature extraction. We study a classification scheme that applies the *soft-thresholding* nonlinear mapping in a dictionary, followed by a linear classifier. A novel supervised dictionary learning algorithm tailored for this low complexity classification architecture is proposed. The dictionary learning problem, which *jointly* learns the dictionary and linear classifier, is cast as a *difference of convex* (DC) program and solved efficiently with an iterative DC solver. We conduct experiments on several datasets, and show that our learning algorithm that leverages the structure of the classification problem outperforms generic learning procedures. Our simple classifier based on soft-thresholding also competes with the recent sparse coding classifiers, when the dictionary is learned appropriately. The adopted classification scheme further requires less computational time at the testing stage, compared to other classifiers. The proposed scheme shows the potential of the adequately trained soft-thresholding mapping for classification and paves the way towards the development of very efficient classification methods for vision problems.

Keywords Dictionary learning · Soft-thresholding · Sparse coding · Rectifier linear units · Neural networks

1 Introduction

The recent decade has witnessed the emergence of huge volumes of high dimensional information produced by all sorts of sensors. For instance, a massive amount of high-resolution images are uploaded on the Internet every minute. In this context, one of the key challenges is to develop techniques to process these large amounts of data in a computationally efficient way. We focus in this paper on the *image classification* problem, which is one of the most challenging tasks in image analysis and computer vision. Given training examples from multiple classes, the goal is to find a rule that permits to predict the class of test samples. *Linear classification* is a computationally efficient way to categorize test samples. It consists in finding a linear separator between two classes.

Linear classification has been the focus of much research in statistics and machine learning for decades and the resulting algorithms are well understood. However, many datasets cannot be separated linearly and require complex nonlinear classifiers. A popular nonlinear scheme, which leverages the efficency and simplicity of linear classifiers, embeds the data into a high dimensional feature space, where a linear classifier is eventually sought. The feature space mapping is chosen to be nonlinear in order to convert nonlinear relations to linear relations. This nonlinear classification framework is at the heart of the popular kernel-based methods

Alhussein Fawzi
Signal Processing Laboratory (LTS4)
E-mail: alhussein.fawzi@epfl.ch

Mike Davies
IDCOM, The University of Edinburgh
E-mail: mike.davies@ed.ac.uk

Pascal Frossard
Signal Processing Laboratory (LTS4)
E-mail: pascal.frossard@epfl.ch

(Shawe-Taylor and Cristianini, 2004) that make use of a computational shortcut to bypass the explicit computation of feature vectors. Despite the popularity of kernel-based classification, its computational complexity at test time strongly depends on the number of training samples (Burges, 1998), which limits its applicability in large scale settings.

A more recent approach for nonlinear classification is based on *sparse coding*, which consists in finding a compact representation of the data in an overcomplete dictionary. Sparse coding is known to be beneficial in signal processing tasks such as denoising (Elad and Aharon, 2006), inpainting (Fadili et al, 2009), coding (Figueras i Ventura et al, 2006), but it has also recently emerged in the context of classification, where it is viewed as a nonlinear feature extraction mapping. It is usually followed by a linear classifier (Raina et al, 2007), but can also be used in conjunction with other classifiers (Wright et al, 2009). Classification architectures based on sparse coding have been shown to work very well in practice and even achieve state-of-the-art results on particular tasks (Mairal et al, 2012; Yang et al, 2009). The crucial drawback of sparse coding classifiers is however the prohibitive cost of computing the sparse representation of a signal or image sample at test time. This limits the relevance of such techniques in large-scale vision problems or when computational power is scarce.

To remedy to these large computational requirements, we adopt in the classification a computationally efficient sparsifying transform, the soft thresholding mapping h_α, defined by:

$$h_\alpha(z) = \max(0, z-\alpha) \triangleq (z-\alpha)_+, \tag{1}$$

for $\alpha \in \mathbb{R}_+$ and $(\cdot)_+ = \max(0,\cdot)$. Note that, unlike the usual definition of soft-thresholding given by $\text{sgn}(z)(|z|-\alpha)_+$, we consider here the *one-sided* version of the soft-thresholding map, where the function is equal to zero for negative values (see Fig. 3 (a) vs. Fig 3 (b)). The map h_α is naturally extended to vectors $\mathbf{z}$ by applying the scalar map to each coordinate independently. Given a dictionary $\mathbf{D}$, this map can be applied to a transformed signal $\mathbf{z} = \mathbf{D^T x}$ that represents the coefficients of features in a signal $\mathbf{x}$. Its outcome, which only considers the most important features of $\mathbf{x}$, is used for classification. In more details, we consider in this paper the following simple two-step procedure for classification:

1. **Feature extraction:** Let $\mathbf{D} = [\mathbf{d_1}|\dots|\mathbf{d_N}] \in \mathbb{R}^{n\times N}$ and $\alpha \in \mathbb{R}_+$. Given a test point $\mathbf{x} \in \mathbb{R}^n$, compute $h_\alpha(\mathbf{D^T x})$.

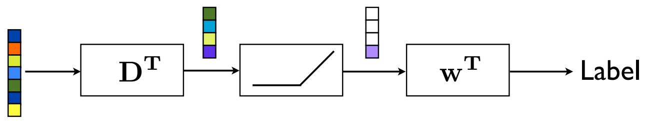


Fig. 1 Soft-thresholding classification scheme. The box in the middle applies the soft-thresholding non-linearity h_α.

2. **Linear classification:** Let $\mathbf{w} \in \mathbb{R}^N$. If $\mathbf{w}^T h_\alpha(\mathbf{D^T x})$ is positive, assign $\mathbf{x}$ to class 1. Otherwise, assign to class -1.

The architecture is illustrated in Fig. 1. The proposed classification scheme has the advantage of being simple, efficient and easy to implement as it involves a single matrix-vector multiplication and a max operation. The soft-thresholding map has been successfully used in (Coates and Ng, 2011), as well as in a number of deep learning architectures (Kavukcuoglu et al, 2010b), which shows the relevance of this efficient feature extraction mapping. The remarkable results in Coates and Ng (2011) show that this simple encoder, when coupled with a standard learning algorithm, can often achieve results comparable to those of sparse coding, provided that the number of labeled samples and the dictionary size are large enough. However, when this is not the case, a proper training of the classifier parameters $(\mathbf{D}, \mathbf{w})$ becomes crucial for reaching good classification performance. This is the objective of this paper.

We propose a novel *supervised dictionary learning* algorithm, which we call LAST (Learning Algorithm for Soft-Thresholding classifier). It *jointly* learns the dictionary $\mathbf{D}$ and the linear classifier $\mathbf{w}$ tailored for the classification architecture based on soft-thresholding. We pose the learning problem as an optimization problem comprising a loss term that controls the classification accuracy and a regularizer that prevents overfitting. This problem is shown to be a *difference-of-convex* (DC) program, which is solved efficiently with an iterative DC solver. We then perform extensive experiments on textures, digits and natural images datasets, and show that the proposed classifier, coupled with our dictionary learning approach, exhibits remarkable performance with respect to numerous competitor methods. In particular, we show that our classifier provides comparable or better classification accuracy than sparse coding schemes.

The rest of this paper is organized as follows. In the next Section, we highlight the related work. In Section 3, we formulate the dictionary learning problem for classifiers based on soft-thresholding. Section 4 then presents our novel learning algorithm, LAST, based on DC optimization. In Section 5, we perform extensive experiments on textures, natural images and digits datasets and Section 6 finally gathers a number of important observations on the dictionary learning algorithm, and the classification scheme.

2 Related work

We first highlight in this section the difference between the proposed approach and existing techniques from the sparse coding and dictionary learning literature. Then, we draw a connection between the considered approach and neural network models on the architecture and optimization aspects.

2.1 Sparse coding

The classification scheme adopted in this paper shares similarities with the now popular architectures that use sparse coding at the feature extraction stage. We recall that the sparse coding mapping, applied to a datapoint $\mathbf{x}$ in a dictionary $\mathbf{D}$ consists in solving the optimization problem

$$\operatorname*{argmin}_{\mathbf{c}\in\mathbb{R}^N} \|\mathbf{x}-\mathbf{D}\mathbf{c}\|_2^2 + \lambda\|\mathbf{c}\|_1. \tag{2}$$

It is now known that, when the parameters of the sparse coding classifier are trained in a discriminative way, excellent classification results are obtained in many vision tasks (Mairal et al, 2012, 2008; Ramirez et al, 2010). In particular, significant gains over the standard reconstructive dictionary learning approaches are obtained when the dictionary is optimized for classification. Several dictionary learning methods also consider an additional structure (e.g., low-rankness) on the dictionary, in order to incorporate a task-specific prior knowledge (Zhang et al, 2013; Chen et al, 2012; Ma et al, 2012). This line of research is especially popular in face recognition applications, where a mixture of subspace model is known to hold (Wright et al, 2009). Up to our knowledge, all the discriminative dictionary learning methods optimize the dictionary in regards to the sparse coding map in Eq. (2), or a variant that still requires to solve a non trivial optimization problem. In our work however, we introduce a discriminative dictionary learning method *specific to the efficient soft-thresholding map.* Interestingly, soft-thresholding can be viewed as a coarse approximation to non-negative sparse coding, as we show in Appendix A. This further motivates the use of soft-thresholding for feature extraction, as the merits of sparse coding for classification are now well-established.

Closer to our work, several approaches have been introduced to approximate sparse coding with a more efficient feed-forward predictor (Kavukcuoglu et al, 2010a; Gregor and LeCun, 2010), whose parameters are learned in order to minimize the approximation error with respect to sparse codes. These works are however different from ours in several aspects. First, our approach does not require the result of the soft-thresholding mapping to be close to that of sparse coding. We rather require solely a good classification accuracy on the training samples. Moreover, our dictionary learning approach is purely supervised, unlike Kavukcuoglu et al (2010a,b). Finally, these methods often use nonlinear maps (e.g., hyperbolic tangent in Kavukcuoglu et al (2010a), multi-layer soft-thresholding in Gregor and LeCun (2010)) that are different from the one considered in this paper. The single soft-thresholding mapping considered here has the advantage of being simple, very efficient and easy to implement in practice. It is also strongly tied to sparse coding (see Appendix A).

2.2 Neural networks

The classification architecture considered in our work is also quite strongly related to artificial neural network models (Bishop, 1995). Neural network models are multi-layer architectures, where each layer consists of a set of neurons. The neurons compute a linear combination of the activation values of the preceding layer, and an *activation function* is then used to convert the neurons' weighted input to its activation value. Popular choices of activation functions are logistic sigmoid and hyperbolic tangent nonlinearities. Our classification architecture can be seen as a neural network with one hidden layer and h_α as the hidden units' activation function, and zero bias (Fig. 2). Equivalently, the activation function can be set to $\max(0, x)$ with a constant bias $-\alpha$ across all hidden units. The dictionary $\mathbf{D}$ defines the connections between the input and hidden layer, while $\mathbf{w}$ represents the weights that connect the hidden layer to the output.

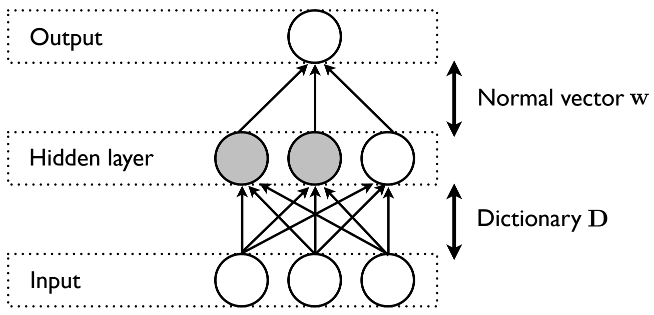


Fig. 2 Neural network representation of our classification architecture. Greyed neurons have zero activation value.

In an important recent contribution, Glorot et al (2011) showed that using the rectifier activation function $\max(0, x)$ results in better performance for deep networks than the more classical hyperbolic tangent function. On top of that, the rectifier nonlinearity is more biologically plausible, and leads to sparse networks; a property that is highly desirable in representation learning (Bengio et al, 2013). While the architecture considered in this paper is close to that of Glorot et al (2011), it differs in several important aspects. First, our architecture assumes that hidden units have a bias equal to $-\alpha < 0$, shared across all the hidden units, while it is unclear whether any constraint on the bias is set in the existing rectifier networks. The parameter α is intimately related to the sparsity of the features. This can be justified by the fact that h_α is an approximant to the non-negative sparse coding map with sparsity penalty α (see Appendix A). Without imposing any restriction on the neurons' bias (e.g., negativity) in rectifier networks, the representation might however not be sparse. This potentially explains the necessity to use an additional ℓ_1 sparsifying regularizer on the activation values in Glorot et al (2011) to enforce the sparsity of the network, while sparsity is achieved implicitly in our scheme. Second, unlike the work of (Glorot et al, 2011) that employs a biological argument to introduce the rectifier function, we choose the soft-thresholding nonlinearity due to its strong relation to sparse coding. Our work therefore provides an independent motivation for considering the rectifier activation function, while the biological motivation in (Glorot et al, 2011) in turn gives us another motivation for considering soft-thresholding. Third, rectified linear units are very often used in the context of deep networks (Maas et al, 2013; Zeiler et al, 2013), and seldom used with only one hidden layer. In that sense, the classification scheme considered in this paper has a simpler description, and can be seen as a particular instance of the general neural network models.

From an optimization perspective, our learning algorithm leverages the simplicity of our classification architecture and is very different from the generic techniques used to train neural networks. In particular, while neural networks are generally trained with stochastic gradient descent, we adopt an optimization based on the DC framework that directly exploits the structure of the learning problem.

3 Problem formulation

We present below the learning problem, that estimates jointly the dictionary $\mathbf{D} \in \mathbb{R}^{n \times N}$ and linear classifier $\mathbf{w} \in \mathbb{R}^N$ in our fast classification scheme described in Section 1. We consider the binary classification task where $\mathbf{X} = [\mathbf{x_1}| \dots |\mathbf{x_m}] \in \mathbb{R}^{n \times m}$ and $\mathbf{y} = [y_1| \dots |y_m] \in \{-1, 1\}^m$ denote respectively the set of training points and their associated labels. We consider the following supervised learning formulation

$$\underset{\mathbf{D},\mathbf{w}}{\text{argmin}} \sum_{i=1}^{m} L(y_i \mathbf{w^T} h_\alpha(\mathbf{D^T x_i})) + \frac{\nu}{2} \|\mathbf{w}\|_2^2, \tag{3}$$

where L denotes a convex loss function that penalizes incorrect classification of a training sample and ν is a regularization parameter that prevents overfitting. The soft-thresholding map h_α has been defined in Eq. (1). Typical loss functions that can be used in Eq. (3) are the hinge loss ($L(x) = \max(0, 1 - x)$), which we adopt in this paper, or its smooth approximation, the logistic loss ($L(x) = \log(1 + e^{-x})$). The above optimization problem attempts to find a dictionary $\mathbf{D}$ and a linear separator $\mathbf{w}$ such that $\mathbf{w^T}(\mathbf{D^T x_i} - \alpha)_+$ has the same sign as y_i on the training set, which leads to correct classification. At the same time, it keeps $\|\mathbf{w}\|_2$ small in order to prevent overfitting. Note that to simplify the exposition, the bias term in the linear classifier is dropped. However, our study extends straightforwardly to include nonzero bias.

The problem formulation in Eq. (3) is reminiscent of the popular support vector machine (SVM) training procedure, where only a linear classifier $\mathbf{w}$ is learned. Instead, we embed the nonlinearity directly in the problem formulation, and learn jointly the dictionary $\mathbf{D}$ and the linear classifier $\mathbf{w}$. This significantly broadens the applicability of the learned classifier to important nonlinear classification tasks. Note however that adding a nonlinear mapping raises an important optimization challenge, as the learning problem is no more convex.

When we look closer at the optimization problem in Eq. (3), we note that, for any $\alpha > 0$, the objective function is equal to:

$$\begin{aligned} &\sum_{i=1}^{m} L(y_i \alpha \mathbf{w^T} h_1(\mathbf{D^T x_i}/\alpha)) + \frac{\nu}{2} \|\mathbf{w}\|_2^2 \\ =&\sum_{i=1}^{m} L(y_i \tilde{\mathbf{w}}^\mathbf{T} h_1(\tilde{\mathbf{D}}^\mathbf{T} \mathbf{x_i})) + \frac{\nu'}{2} \|\tilde{\mathbf{w}}\|_2^2, \end{aligned}$$

where $\tilde{\mathbf{w}} = \alpha \mathbf{w}$, $\tilde{\mathbf{D}} = \mathbf{D}/\alpha$ and $\nu' = \nu/\alpha^2$. Therefore, without loss of generality, we set the sparsity parameter α to 1 in the rest of this paper. This is in contrast with traditional dictionary learning approaches based on ℓ_0 or ℓ_1 minimization problems, where a sparsity parameter needs to be set manually beforehand. Fixing $\alpha = 1$ and unconstraining the norms of the dictionary atoms essentially permits to adapt the sparsity to the problem at hand. This represents an important advantage, as setting the sparsity parameter is in general a difficult task. A sample $\mathbf{x}$ is then assigned to class '+1' if $\mathbf{w^T} h_1(\mathbf{D^T x}) > 0$, and class '$-1$' otherwise.

Finally, we note that, even if our focus primarily goes to the binary classification problem, the extension to multi-class can be easily done through a one-vs-all strategy, for instance.

4 Learning algorithm

The problem in Eq. (3) is non-convex and difficult to solve in general. In this section, we propose to relax the original optimization problem and cast it as a *difference-of-convex* (DC) program. Leveraging this property, we introduce LAST, an efficient algorithm for learning the dictionary and the classifier parameters in our classification scheme based on soft-thresholding.

4.1 Relaxed formulation

We rewrite now the learning problem in an appropriate form for optimization. We start with a simple but crucial change of variables. Specifically, we define $\mathbf{u_j} \leftarrow |w_j|\mathbf{d_j}$, $v_j \leftarrow |w_j|$ and $s_j \leftarrow \text{sgn}(w_j)$. Using this change of variables, we have for any $1 \leq i \leq m$,

$$\begin{aligned} y_i \mathbf{w^T} h_1(\mathbf{D^T x_i}) &= y_i \sum_{j=1}^{N} \text{sgn}(w_j)(|w_j| \mathbf{d_j^T x_i} - |w_j|)_+ \\ &= y_i \sum_{j=1}^{N} s_j (\mathbf{u_j^T x_i} - v_j)_+. \end{aligned}$$

Therefore, the problem in Eq.(3), with $\alpha = 1$, can be rewritten in the following way:

$$\underset{\mathbf{U},\mathbf{v},\mathbf{s}}{\text{argmin}} \sum_{i=1}^{m} L\left(y_i \sum_{j=1}^{N} s_j (\mathbf{u_j^T x_i} - v_j)_+ \right) + \frac{\nu}{2}\|\mathbf{v}\|_2^2, \quad (4)$$
$$\text{subject to } \mathbf{v} > 0.$$

The equivalence between the two problem formulations in Eqs. (3) and (4) only holds when the components of the linear classifier $\mathbf{w}$ are restricted to be all non zero. This is however not a limiting assumption as zero components in the normal vector of the optimal hyperplane of Eq. (3) can be removed, which is equivalent to using a dictionary of smaller size.

The variable $\mathbf{s}$, that is the sign of the components of $\mathbf{w}$, essentially encodes the "classes" of the different atoms. In other words, an atom $\mathbf{d_j}$ for which $s_j = +1$ (i.e., w_j is positive) is most likely to be active for samples of class '1'. Conversely, atoms with $s_j = -1$ are most likely active for class '-1' samples. We assume here that the vector $\mathbf{s}$ is known a priori. In other words, this means that we have a prior knowledge on the proportion of class 1 and class -1 atoms in the desired dictionary. For example, setting half of the entries of the vector $\mathbf{s}$ to be equal to $+1$ and the other half to -1 encodes the prior knowledge that we are searching for a dictionary with a balanced number of class-specific atoms. Note that $\mathbf{s}$ can be estimated from the distribution of the different classes in the training set, assuming that the proportion of class-specific atoms in the dictionary should approximately follow that of the training samples.

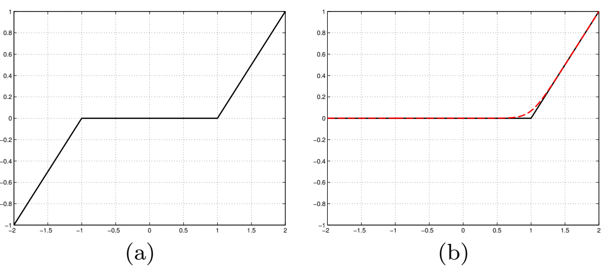


Fig. 3 (a): $\text{sgn}(x)(|x| - \alpha)_+$, (b): h_α (solid), and its smooth approximation $q(x-\alpha)$ (dashed), with $\beta = 10$. We used $\alpha = 1$.

After the above change of variables, we now approximate the term $(\mathbf{u_j^T x_i} - v_j)_+$ in Eq.(4) with a smooth function $q(\mathbf{u_j^T x_i} - v_j)$ where $q(x) = \frac{1}{\beta}\log(1 + \exp(\beta x))$, and β is a parameter that controls the accuracy of the approximation (Fig. 3 (b)). Specifically, as β increases, the quality of the approximation becomes better. The function q with $\beta = 1$ is often referred to as "soft-plus" and plays an important role in the training objective of many classification schemes, such as the classification restricted Boltzmann machines (Larochelle et al, 2012). Note that this approximation is used only to make the optimization easier at the learning stage; at test time, the original soft-thresholding is applied for feature extraction.

Finally, we replace the strict inequality $\mathbf{v} > 0$ in Eq. (4) with $\mathbf{v} \geq \epsilon$, where ϵ is a small positive constant number. The latter constraint is easier to handle in the optimization, yet both constraints are essentially equivalent in practice.

We end up with the following optimization problem:

$$(\text{P}): \underset{\mathbf{U},\mathbf{v}}{\text{argmin}} \sum_{i=1}^{m} L\left(y_i \sum_{j=1}^{N} s_j q(\mathbf{u_j^T x_i} - v_j) \right) + \frac{\nu}{2}\|\mathbf{v}\|_2^2,$$
$$\text{subject to } \mathbf{v} \geq \epsilon,$$

that is a relaxed version of the learning problem in Eq. (4). Once the optimal variables $(\mathbf{U}, \mathbf{v})$ are determined,

D and **w** can be obtained using the above change of variables.

4.2 DC decomposition

The problem (P) is still a nonconvex optimization problem that can be hard to solve using traditional methods, such as gradient descent or Newton-type methods. However, we show in this section that problem (P) can be written as a *difference of convex* (DC) program (Horst, 2000) which leads to efficient solutions.

We first define DC functions. A real-valued function f defined on a convex set $U \subseteq \mathbb{R}^n$ is called DC on U if, for all $\mathbf{x} \in U$, f can be expressed in the form

$$f(\mathbf{x}) = g(\mathbf{x}) - h(\mathbf{x}),$$

where g and h are convex functions on U. A representation of the above form is said to be a DC decomposition of f. Note that DC decompositions are clearly not unique, as $f(\mathbf{x}) = (g(\mathbf{x}) + c(\mathbf{x})) - (h(\mathbf{x}) + c(\mathbf{x}))$ provides other decompositions of f, for any convex function c. Optimization problems of the form $\min_{\mathbf{x}}\{f(\mathbf{x}) : f_i(\mathbf{x}) \leq 0, i = 1, \ldots, p\}$, where f and f_i for $1 \leq i \leq p$ are all DC functions, are called *DC programs.*

The following proposition now states that the problem (P) is DC:

Proposition 1 *For any convex loss function L and any convex function q, the problem (P) is DC.*

While Proposition 1 states that the problem (P) is DC, it does not give an explicit decomposition of the objective function, which is crucial for optimization. The following proposition exhibits a decomposition when L is the hinge loss.

Proposition 2 *When $L(x) = \max(0, 1-x)$, the objective function of problem (P) is equal to $g - h$, where*

$$g = \tfrac{\nu}{2}\|\mathbf{v}\|_2^2 + \sum_{i=1}^{m} \max\Big(\sum_{j:s_j=y_i} q(\mathbf{u_j^T x_i} - v_j), \\ 1 + \sum_{j:s_j \neq y_i} q(\mathbf{u_j^T x_i} - v_j)\Big),$$

$$h = \sum_{i=1}^{m} \sum_{j:s_j=y_i} q(\mathbf{u_j^T x_i} - v_j).$$

The proofs of Propositions 1 and 2 are given in Appendix B. Due to Proposition 2, the problem (P) can be solved efficiently using a DC solver.

4.3 Optimization

DC problems are well studied optimization problems and efficient optimization algorithms have been proposed in (Horst, 2000; Tao and An, 1998) with good performance in practice (see An and Tao (2005) and references therein, Sriperumbudur et al (2007)). While there exists a number of popular approaches that solve *globally* DC programs (e.g., cutting plane and branch-and-bound algorithms (Horst, 2000)), these techniques are often inefficient and limited to very small scale problems. A robust and efficient difference of convex algorithm (DCA) is proposed in Tao and An (1998), which is suited for solving general large scale DC programs. DCA is an iterative algorithm that consists in solving, at each iteration, the convex optimization problem obtained by linearizing h (i.e., the non convex part of $f = g - h$) around the current solution. The local convergence of DCA is proven in Theorem 3.7 of Tao and An (1998), and we refer to this paper for further theoretical guarantees on the stability and robustness of the algorithm. Although DCA is only guaranteed to reach a local minima, the authors of Tao and An (1998) state that DCA often converges to a global optimum. When this is not the case, using multiple restarts might be used to improve the solution. We note that DCA is very close to the concave-convex procedure (CCCP) introduced in (Yuille et al, 2002).

At iteration k of DCA, the linearized optimization problem is given by:

$$\underset{(\mathbf{U},\mathbf{v})}{\operatorname{argmin}}\{g(\mathbf{U}, \mathbf{v}) - Tr(\mathbf{U}^T\mathbf{A}) - \mathbf{v}^T\mathbf{b}\} \text{ subject to } \mathbf{v} \geq \epsilon. \tag{5}$$

where $(\mathbf{A}, \mathbf{b}) = \nabla h(\mathbf{U^k}, \mathbf{v^k})$ and $(\mathbf{U^k}, \mathbf{v^k})$ are the solution estimates at iteration k, and the functions g and h are defined in Proposition 2. Note that, due to the convexity of g, the problem in Eq. (5) is convex and can be solved using any convex optimization algorithm (Boyd and Vandenberghe, 2004). The method we propose to use here is a projected first-order stochastic subgradient descent algorithm. Stochastic gradient descent is an efficient optimization algorithm that can handle large training sets (Akata et al, 2014). To make the exposition clearer, we first define the function:

$$p(\mathbf{U}, \mathbf{v}; \mathbf{x_i}, y_i) = \max\Big(\sum_{j:s_j=y_i} q(\mathbf{u_j^T x_i} - v_j), \\ 1 + \sum_{j:s_j \neq y_i} q(\mathbf{u_j^T x_i} - v_j)\Big) \\ + \tfrac{1}{m}\left(\tfrac{\nu}{2}\|\mathbf{v}\|_2^2 - Tr(\mathbf{U}^T\mathbf{A}) - \mathbf{v}^T\mathbf{b}\right).$$

The objective function of Eq. (5) that we wish to minimize can then be written as $\sum_{i=1}^{m} p(\mathbf{U}, \mathbf{v}; \mathbf{x_i}, y_i)$. We solve this optimization problem with the projected stochastic subgradient descent algorithm in Algorithm 1.

Algorithm 1 Optimization algorithm to solve the linearized problem in Eq. (5)

```
1. Initialization: $\mathbf{U} \leftarrow \mathbf{U^k}$ and $\mathbf{v} \leftarrow \mathbf{v^k}$.
2. For $t = 1, \dots, T$
    2.1 Let $(\mathbf{x}, y)$ be a randomly chosen training point, and its associated label.
    2.2 Choose the stepsize $\rho_t \leftarrow \min(\rho, \rho\frac{t_0}{t})$.
    2.3 Update $\mathbf{U}$, and $\mathbf{v}$, by projected subgradient step:
        $\mathbf{U} \leftarrow \mathbf{U} - \rho_t \partial_{\mathbf{U}} p(\mathbf{U}, \mathbf{v}; \mathbf{x}, y)$,
        $\mathbf{v} \leftarrow \Pi_{\mathbf{v} \geq \epsilon} \left( \mathbf{v} - \rho_t \partial_{\mathbf{v}} p(\mathbf{U}, \mathbf{v}; \mathbf{x}, y) \right)$,
    where $\Pi_{\mathbf{v} \geq \epsilon}$ is the projection operator on the set $\mathbf{v} \geq \epsilon$.
3. Return $\mathbf{U^{k+1}} \leftarrow \mathbf{U}$ and $\mathbf{v^{k+1}} \leftarrow \mathbf{v}$.
```

In more details, at each iteration of Algorithm 1, a training sample $(\mathbf{x}, y)$ is drawn. $\mathbf{U}$ and $\mathbf{v}$ are then updated by performing a step in the direction $\partial p(\mathbf{U}, \mathbf{v}; \mathbf{x}, y)$. Many different stepsize rules can be used with stochastic gradient descent methods. In this paper, similarly to the strategy employed in Mairal et al (2012), we have chosen a stepsize that remains constant for the first t_0 iterations, and then takes the value $\rho t_0/t$.[1] Moreover, to accelerate the convergence of the stochastic gradient descent algorithm, we consider a small variation of Algorithm 1, where a minibatch containing several training samples along with their labels is drawn at each iteration, instead of a single sample. This is a classical heuristic in stochastic gradient descent algorithms. Note that, when the size of the minibatch is equal to the number of training samples, this algorithm reduces to traditional batch gradient descent.

Finally, our complete LAST learning algorithm based on DCA is formally given in Algorithm 2. Starting from a feasible point $\mathbf{U^0}$ and $\mathbf{v^0}$, LAST solves iteratively the constrained convex problem given in Eq. (5) with the solution proposed in Algorithm 1. Recall that this problem corresponds to the original DC program (P), except that the function h has been replaced by its linear approximation around the current solution $(\mathbf{U^k}, \mathbf{v^k})$ at iteration k. Many criteria can be used to terminate the algorithm. We choose here to terminate when a maximum number of iterations K has been reached, and terminate the algorithm earlier when the following condition is satisfied:

$$\min \left\{ |(\omega^{k+1} - \omega^k)_{i,j}|, \left| \frac{(\omega^{k+1} - \omega^k)_{i,j}}{(\omega^k)_{i,j}} \right| \right\} \leq \delta,$$

[1] The precise choice of the parameters ρ and t_0 are discussed later in Section 5.1.

Algorithm 2 LAST (Learning Algorithm for Soft-Thresholding classifier)

```
1. Choose any initial point: $\mathbf{U^0}$ and $\mathbf{v^0} \geq \epsilon$.
2. For $k = 0, \dots, K-1$,
    2.1 Compute $(\mathbf{A}, \mathbf{b}) = \nabla h(\mathbf{U^k}, \mathbf{v^k})$.
    2.2 Solve with Algorithm 1 the convex optimization problem:
        $(\mathbf{U^{k+1}}, \mathbf{v^{k+1}}) \leftarrow \underset{(\mathbf{U},\mathbf{v})}{\text{argmin}} \{ g(\mathbf{U}, \mathbf{v}) - Tr(\mathbf{U}^T \mathbf{A}) - \mathbf{v}^T \mathbf{b} \}$
        subject to $\mathbf{v} \geq \epsilon$.
    2.3 If $(\mathbf{U^{k+1}}, \mathbf{v^{k+1}}) \approx (\mathbf{U^k}, \mathbf{v^k})$, return $(\mathbf{U^{k+1}}, \mathbf{v^{k+1}})$.
```

where the matrix $\mathbf{\Omega^k} = (\omega^k)_{i,j}$ is the row concatenation of $\mathbf{U}$ and $\mathbf{v^T}$, and δ is a small positive number. This condition detects the convergence of the learning algorithm, and is verified whenever the change in $\mathbf{U}$ and $\mathbf{v}$ is very small. This termination criterion is used for example in Sriperumbudur et al (2007).

5 Experimental results

In this section, we evaluate the performance of our classification algorithm on textures, digits and natural images datasets, and compare it to different competitor schemes. We expose in Section 5.1 the choice of the parameters of the model and the algorithm. We then focus on the experimental assessment of our scheme. Following the methodology of Coates and Ng (2011), we break the feature extraction algorithms into (i) a learning algorithm (e.g, K-Means) where a set of basis functions (or dictionary) is learned and (ii) an encoding function (e.g., ℓ_1 sparse coding) that maps an input point to its feature vector. In a first step of our analysis (Section 5.2), we therefore *fix the encoder* to be the soft-thresholding mapping and compare LAST to existing supervised and unsupervised learning techniques. Then, in the following subsections, we compare our complete classification architecture (i.e., learning and encoding function) to several classifiers, in terms of accuracy and efficiency. In particular, we show that our proposed approach is able to compete with recent classifiers, despite its simplicity.

5.1 Parameter selection

We first discuss the choice of the model parameters for our method. Unless stated otherwise, we choose the vector $\mathbf{s}$ according to the distribution of the different classes in the training set. We set the value of the regularization parameter to $\nu = 1$, as it was found empirically to be a good choice in our experiments. It is worth mentioning that setting ν by cross-validation might give better results, but it would also be computationally more expensive. We set moreover the parameter of the soft-thresholding mapping approximation to $\beta = 100$. Recall finally that the sparsity parameter α is always equal to 1 in our method, and therefore does not require any manual setting or cross-validation procedure.

In all experiments, we have moreover chosen to initialize LAST by setting $\mathbf{U^0}$ equal to a random subsample of the training set, and $\mathbf{v^0}$ is set to the vector whose entries are all equal to 1. We however noticed empirically that choosing a different initialization strategy does not significantly change the testing accuracy. Then, we fix the maximum number of iterations of LAST to $K = 50$. Moreover, setting properly the parameters t_0 and ρ in Algorithm 1 is quite crucial in controlling the convergence of the algorithm. In all the experiments, we have set the parameter $t_0 = T/10$, where T denotes the number of iterations. Furthermore, during the first $T/20$ iterations, several values of ρ are tested $\{0.1, 0.01, 0.001\}$, and the value that leads to the smallest objective function is chosen for the rest of the iterations. Finally, the minibatch size in Algorithm 1 depends on the size of the training data. In particular, when the size of the training data m is relatively small (i.e., smaller than 5000), we used a batch gradient descent, as the computation of the (complete) gradient is tractable. In this case, we set the number of iterations to $T = 1000$. Otherwise, we use a batch size of 200, and perform $T = 5000$ iterations of the stochastic gradient descent in Algorithm 1.

5.2 Analysis of the learning algorithm

In a first set of experiments, we focus on the comparison of our learning algorithm (LAST) to other learning techniques, and fix the encoder to be the soft-thresholding mapping for all the methods. We present a comparative study on textures and natural images classification tasks.

5.2.1 Experimental settings

We consider the following dictionary learning algorithms:

1. **Supervised random samples:** The atoms of $\mathbf{D}$ are chosen randomly from the training set, in a supervised manner. That is, if κ denotes the desired proportion of class '1' atoms in the dictionary, the dictionary is built by randomly picking κN training samples from class '1' and $(1-\kappa)N$ samples from class '-1', where N is the number of atoms in the dictionary.
2. **Supervised K-means:** We build the dictionary by merging the subdictionaries obtained by applying the K-means algorithm successively to training samples of class '1' and '-1', where the number of clusters is fixed respectively to κN and $(1-\kappa)N$.
3. **Dictionary learning for ℓ_1 sparse coding:** The dictionary $\mathbf{D}$ is built by solving the classical dictionary learning problem for ℓ_1 sparse coding:

$$\min_{\mathbf{D},\mathbf{c_i}} \sum_{i=1}^{m} \|\mathbf{x_i} - \mathbf{D}\mathbf{c_i}\|_2^2 + \lambda \|\mathbf{c_i}\|_1 \text{ subject to } \forall j, \|\mathbf{d_j}\|_2^2 \leq 1. \tag{6}$$

   To solve this optimization problem, we used the algorithm proposed by Mairal et al (2010) and implemented in the SPAMS package. The parameter λ is chosen by a cross-validation procedure in the set $\{0.1, 0.01, 0.001\}$. Note that, while the previous two learning algorithms make use of the labels, this algorithm is unsupervised.
4. **Stochastic Gradient Descent (SGD):** The dictionary $\mathbf{D}$ and classifier $\mathbf{w}$ are obtained by optimizing the following objective function using *mini-batch stochastic gradient descent*:

$$J(\mathbf{D}, \mathbf{w}) = \sum_{i=1}^{m} L(y_i \mathbf{w^T} q(\mathbf{D^T}\mathbf{x_i} - \alpha)) + \frac{\nu}{2}\|\mathbf{w}\|_2^2,$$

   with $q(x) = \frac{1}{\beta}\log(1 + \exp(\beta x))$. This corresponds to the original objective function in Eq. (3), where h_α is replaced with its smooth approximant. [2] This smoothing procedure is similar to the one used in our relaxed formulation (Section 4.1). As in LAST, we set $\beta = 100$, $\alpha = 1$, and use the same initialization strategy. This setting allows us to directly compare LAST and this generic stochastic gradient descent procedure widely used for training neural networks. Following Glorot et al (2011), we use a mini-batch size of 10, and use a constant step size chosen in $\{0.1, 0.01, 0.001, 0.0001\}$. The stepsize is chosen through a cross-validation procedure, with a randomly chosen validation set made up of 10% of the training data. The number of iterations of SGD is set to 250000.

For the first three algorithms, the parameter α in the soft-thresholding mapping is chosen with cross validation in $\{0.1, 0.2, \ldots, 0.9, 1\}$. The features are then computed by applying the soft thresholding map h_α, and a linear SVM classifier is trained in the feature space. For the random samples and K-means approaches,

[2] We also tested SGD on the original (non-smooth) optimization problem. This resulted in slightly worse performance. We therefore only report results obtained on the smoothed objective function.

we set $\kappa = 0.5$ as we consider classification tasks with roughly equal number of training samples from each class. Finally, for SGD and LAST, the dictionary $\mathbf{D}$ and linear classifier $\mathbf{w}$ are learned simultaneously. The encoder h_1 is used to compute the features.

5.2.2 Experimental results

In our first experiment, we consider two binary texture classification tasks, where the textures are collected from the 32 Brodatz dataset (Valkealahti and Oja, 1998) and shown in Fig. 4. For each pair of textures under test, we build the training set by randomly selecting 500 12×12 patches per texture, and the test data is constructed similarly by taking 500 patches per texture. The test data does not contain any of the training patches. All the patches are moreover normalized to have unit ℓ_2 norm. Fig. 5 shows the binary classification accuracy of the soft-thresholding based classifier as a function of the dictionary size, for dictionaries learned with the different algorithms.

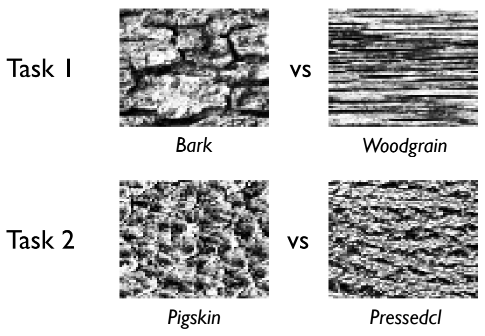
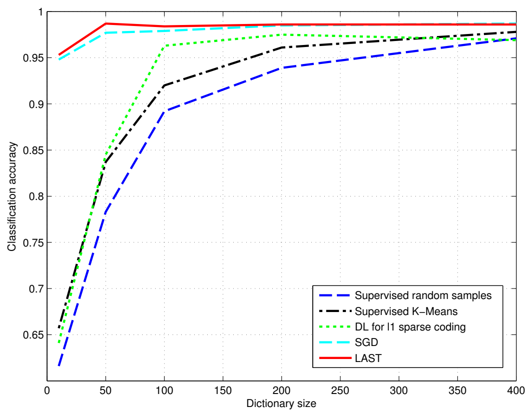
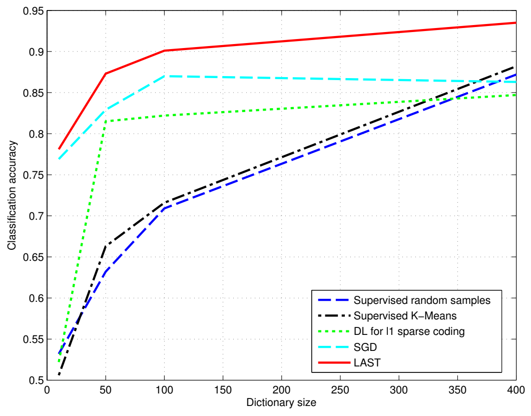


Fig. 4 Two binary classification tasks (*bark* vs *woodgrain* and *pigskin* vs. *pressedcl*)

For the first task (*bark* vs. *woodgrain*), one can see that LAST and SGD dictionary learning methods outperform the other methods for small dictionary sizes. For large dictionaries (i.e., $N \approx 400$) however, all the learning algorithms yield approximately the same classification accuracy. This result is in agreement with the conclusions of Coates and Ng (2011), where the authors show empirically that the choice of the learning algorithm becomes less crucial when dictionaries are very large. In the second and more difficult classification task (*pigskin* vs. *pressedcl*), our algorithm yields the best classification accuracy for all tested dictionary sizes ($10 \leq N \leq 400$). Interestingly, unlike the previous task, the design of the dictionary is crucial for all tested dictionary sizes. Using much larger dictionaries might result in performance that is close to the one obtained using our algorithm, but comes at the price of additional computational and memory costs.



(a) *Bark vs. Woodgrain*



(b) *Pigskin vs. Pressedcl*

Fig. 5 Texture classification results (fixed soft-thresholding encoder)

Fig. 6 further illustrates the evolution of the objective function with respect to the elapsed training time for LAST and SGD, for a dictionary of size 50. One can see that LAST quickly converges to a solution with a small objective function. On the other hand, SGD reaches a solution with larger objective function than LAST.

We now conduct experiments on the popular CIFAR-10 image database (Krizhevsky and Hinton, 2009). The dataset contains 10 classes of 32×32 RGB images. For simplicity and better comparison of the different learning algorithms, we restrict in a first stage the dataset to the two classes "deer" and "horse". We extend our results to the multi-class scenario later in Section 5.5. Fig. 7 illustrates some training examples from the two classes. The classification results are reported in Fig. 8. Once again, the soft-thresholding based classifier with a dictionary and linear classifier learned with LAST

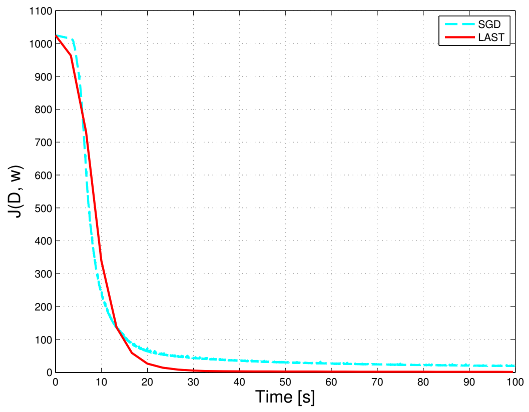


Fig. 6 $J(\mathbf{D}, \mathbf{w})$ as a function of the elapsed time [s] for Stochastic Gradient Descent and LAST. For SGD: $J(\mathbf{D}_{t=100}, \mathbf{w}_{t=100}) = 19$, LAST: $J(\mathbf{D}_{t=100}, \mathbf{w}_{t=100}) = 1.4$.

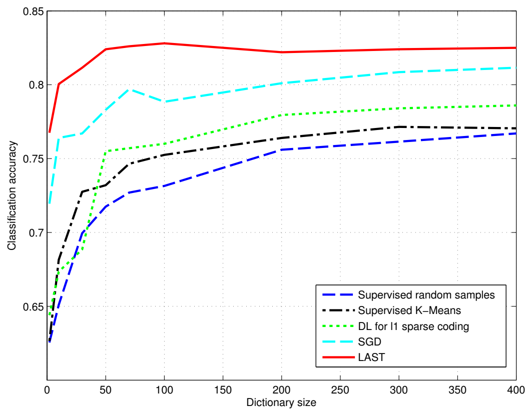
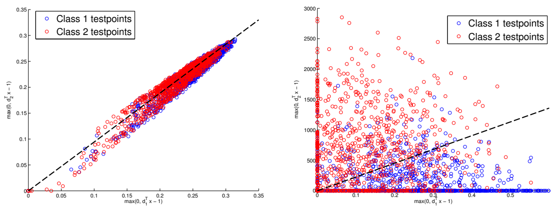


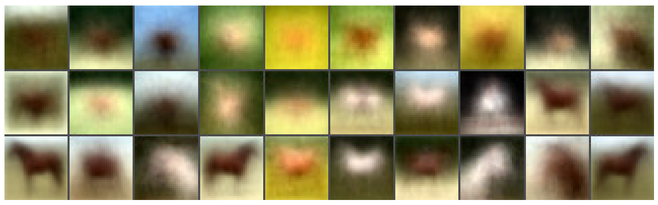
Fig. 8 Performance of the "deer" vs. "horse" binary classification task (fixed soft-thresholding encoder)

outperforms all other learning techniques. In particular, using the LAST dictionary learning strategy results in significantly higher performance than stochastic gradient descent for all dictionary sizes. We further note that with a very small dictionary (i.e., $N = 2$), LAST reaches an accuracy of 77%, whereas some learning algorithms (e.g., K-means) do not reach this accuracy even with a dictionary that contains as many as 400 atoms. To further illustrate this point, we show in Fig. 9 the 2-D testing features obtained with a dictionary of two atoms, when $\mathbf{D}$ is learned respectively with the K-Means method and LAST. Despite the very low-dimensionality of the feature vectors, the two classes can be separated with a reasonable accuracy using our algorithm (Fig. 9 (b)), whereas features obtained with the K-means algorithm clearly cannot be discriminated (Fig. 9 (a)). We finally illustrate in Fig. 10 the dictionaries learned using K-Means and LAST for $N = 30$ atoms. It can be observed that, while K-Means dictionary consists of smoothed images that minimize the reconstruction error, our algorithm learns a discriminative dictionary whose goal is to underline the difference between the images of the two classes.

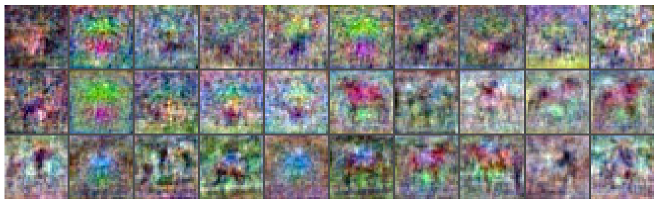


(a) K-Means (b) LAST

Fig. 9 Learned 2D features and linear classifiers with K-Means and LAST for the "deer" vs. "horse" classification task ($N = 2$).



(a) K-Means



(b) LAST

Fig. 10 Normalized dictionary atoms learned with K-Means and LAST, for the "deer" vs. "horse" binary classification task ($N = 30$).

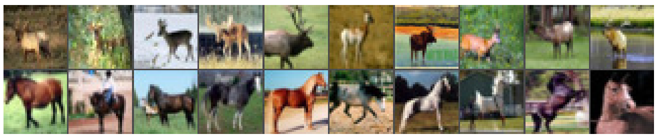


Fig. 7 Examples of CIFAR-10 images in categories "deer" and "horse".

In summary, our supervised learning algorithm, specifically tailored for the soft-thresholding encoder provides significant improvements over traditional dictionary learning schemes. Our classifier can reach high accuracy rates, even with very small dictionaries, which is not possible with other learning schemes.

	Task 1 [%]	Task 2 [%]
Linear SVM	49.5	49.1
RBF kernel SVM	98.5	90.1
Sparse coding ($N = 50$)	97.5	85.5
Sparse coding ($N = 400$)	98.1	90.9
NN ($N = 50$)	94.3	84.1
NN ($N = 400$)	97.8	86.6
LAST ($N = 50$)	**98.7**	87.3
LAST ($N = 400$)	98.6	**93.5**

Table 1 Classification accuracy for binary texture classification tasks.

5.3 Classification performance on binary datasets

In this section, we compare the proposed LAST classification method[3] to other classifiers. Before going through the experimental results, we first present the different methods under comparison:

1. **Linear SVM:** We use the efficient Liblinear (Fan et al, 2008) implementation for training the linear classifier. The regularization parameter is chosen using a cross-validation procedure.
2. **RBF kernel SVM:** We use LibSVM (Chang and Lin, 2011) for training. Similarly, the regularization and width parameters are set with cross-validation.
3. **Sparse coding:** Similarly to the previous section, we train the dictionary by solving Eq. (6). We use however the encoder that "matches naturally" with this training algorithm, that is:

   $$\underset{\mathbf{c}}{\operatorname{argmin}} \|\mathbf{x} - \mathbf{D}\mathbf{c}\|_2^2 + \lambda \|\mathbf{c}\|_1,$$

   where $\mathbf{x}$ is the test sample, $\mathbf{D}$ the previously learned dictionary and $\mathbf{c}$ the resulting feature vector. A linear SVM is then trained on the resulting feature vectors. This classification architecture, denoted "sparse coding" below, is similar to that of Raina et al (2007).
4. **Nearest neighbor classifier (NN):** Our last comparative scheme is a nearest neighbor classifier where the dictionary is learned using the supervised K-means procedure described in 5.2.1. At test time, the sample is assigned the label of the dictionary atom (i.e., cluster) that is closest to it.

Note that we have dropped the supervised random samples learning algorithm used in the previous section as it was shown to have worse classification accuracy than the K-means approach.

Table 1 first shows the accuracies of the different classifiers in the two binary textures classification tasks described in 5.2.2. In both experiments, the linear SVM classifier results in a very poor performance, which is close to the random classifier. This suggests that the considered task is nonlinear, and has to be tackled with a nonlinear classifier. One can see that the RBF kernel SVM results in a significant increase in the classification accuracy. Similarly, the ℓ_1 sparse coding non linear mapping also results in much better performance compared to the linear classifier, while the nearest neighbor approach performs a bit worse than sparse coding. We note that, for a fixed dictionary size, our classifier outperforms NN and sparse coding classifiers in both tasks. Moreover, it provides comparable or superior performance to the RBF kernel SVM in both tasks.

We now turn to the binary experiment "deer" vs. "horse" described in the previous subsection. We show the classification accuracies of the different classifiers in Table 2. LAST outperforms sparse coding and nearest neighbour classifiers for the tested dictionary sizes. RBF kernel SVM however slightly outperforms LAST with $N = 100$ in this experiment. Note however that the RBF kernel SVM approach is much slower at test time, which makes it impractical for large-scale problems.

	"deer" vs. "horse" [%]
Linear SVM	72.6
RBF kernel SVM	**83.5**
Sparse coding ($N = 10$)	70.6
Sparse coding ($N = 100$)	76.2
NN ($N = 10$)	67.7
NN ($N = 100$)	70.9
LAST ($N = 10$)	80.1
LAST ($N = 100$)	82.8

Table 2 Binary classification accuracy on the binary classification problem "deer" vs. "horse".

Overall, the proposed LAST classifier compares favorably to the different tested classifiers. In particular, LAST outperforms the sparse coding technique for a fixed dictionary size in our experiments. This result is notable, as sparse coding classifiers are known to provide very good classification performance in vision tasks. Note that, when used with another standard learning approach as K-Means, the soft-thresholding based classifier is outperformed by sparse coding, which shows the importance of the learning scheme in the success of this classifier.

5.4 Handwritten digits classification

We now consider a classification task on the MNIST (LeCun et al, 1998) and USPS (Hull, 1994) handwritten digits datasets. USPS contains 9298 images of size 16 ×

[3] By extension, we define the LAST classifier to be the soft-thresholding based classifier, where the parameters $(\mathbf{D}, \mathbf{w})$ are learned with LAST.

	MNIST	USPS
Linear SVM	8.19	9.07
RBF kernel SVM	1.4	4.2
K-NN ℓ_2	5.0	5.2
LAST	1.32	4.53
Sparse coding	3.0	5.33
Huang and Aviyente (2006)	-	6.05
SDL-G L (Mairal et al, 2008)	3.56	6.67
SDL-D L (Mairal et al, 2008)	1.05	3.54
Ramirez et al (2010)	1.26	3.98
SGD	2.22	5.88
3 layers ReLU net (Glorot et al, 2011)	1.43	-

Table 3 Classification error (percentage) on MNIST and USPS datasets.

16 pixels, with 7291 images used for training and 2007 for testing. The larger MNIST database is composed of 60000 training images and 10000 test images, all of size 28×28 pixels. We preprocess all the images to have zero-mean and to be of unit Euclidean norm.

We address the multi-class classification task using a one-vs-all strategy, as it is often done in classification problems. Specifically, we learn a separate dictionary and a binary linear classifier by solving the optimization problem for each one-vs-all problem. Classification is then done by predicting using each binary classifier, and choosing the prediction with highest score. In LAST, for each one-vs-all task, we naturally set 1/10 of the entries of $\mathbf{s}$ to 1 and the other entries to -1, assuming the distribution of features of the different classes in the dictionary should roughly be that of the images in the training set. In our proposed approach and SGD, we used dictionaries of size $N = 200$ for USPS and $N = 400$ for MNIST as the latter dataset contains much more training samples. We compare LAST to baseline classification techniques described in the previous section, as well as to sparse coding based methods. In addition to building the dictionary in an unsupervised way, we consider the sparse coding classifiers in Mairal et al (2008); Huang and Aviyente (2006); Ramirez et al (2010), which construct the dictionary in a supervised fashion.

Classification results are shown in Table 3. One can see that LAST largely outperforms linear and nearest neighbour classifiers. Moreover, our method has a slightly better accuracy than RBF-SVM in MNIST, while being slightly worse on the USPS dataset. Our approach also outperforms the soft-thresholding based classifier optimized with stochastic gradient descent on both tasks, which highlights the benefits of our optimization technique compared to the standard algorithm used for training neural networks. We also report from Glorot et al (2011) the performance of a three hidden layer rectified network optimized with stochastic gradient decent, without unsupervised pre-training. It can be seen that LAST, while having a much simpler architecture, slightly outperforms the deep rectifier network on the MNIST task. Furthermore, LAST outperforms the unsupervised sparse coding classifier in both datasets. Interestingly, the proposed scheme also competes with, and sometimes outperforms the discriminative sparse coding techniques of (Huang and Aviyente, 2006; Mairal et al, 2008; Ramirez et al, 2010), where the dictionary is tuned for classification. While providing comparable results, the LAST classifier is much faster at test time than sparse coding techniques and RBF-SVM classifiers. It is noteworthy to mention that the best discriminative dictionary learning results we are aware of on these datasets are achieved by Mairal et al (2012) with an error rate of 0.54% on MNIST and 2.84% on USPS. Note however that in this paper, the authors explicitly incorporate translation invariance in the problem by augmenting the training set with shifted versions of the digits. Our focus goes here instead on methods that do not augment the training set with distorted or transformed samples.

5.5 CIFAR-10 classification

We now consider the multi-class classification problem on the CIFAR-10 dataset (Krizhevsky and Hinton, 2009). The dataset contains 60000 color images of size 32×32 pixels, with 50000 images for training and 10000 for testing. The classifier input consists of vectors of raw pixel values of dimension $32 \times 32 \times 3 = 3072$. This setting, similar to that of Glorot et al (2011), takes no advantage of the fact that we are dealing with images and is sometimes referred to as "permutation invariant", as columns in the data could be shuffled without affecting the result. We consider this scenario to focus on the comparison of the performance of the classifiers. Due to the relatively high dimensions of the problem ($n = 3072$, $m = 50000$), we limit ourselves to classifiers with feedforward architectures. In fact, using RBF-SVM for this task would be prohibitively slow at the training and testing stage. For each one-vs-all task, we set the dictionary size of LAST and SGD methods to 400. Moreover, unlike the previous experiment, we set in LAST half of the entries of the sign vector $\mathbf{s}$ to 1 and the other half to -1. This is due to the high variability of intra-class images and the relatively small dictionary size: the number of atoms required to encode the positive class might not be sufficient if $\mathbf{s}$ is set according to the distribution of images in the training set. The results are reported in Table 4.

Once again, this experiment confirms the superiority of our learning algorithm over linear SVM. More-

	CIFAR-10
Linear SVM	59.70
LAST ($N = 400$)	46.56
SGD ($N = 400$)	52.96
3 layers ReLU net	50.86
3 layers ReLU net + sup. pre-train	49.96

Table 4 Classification error (percentage) on the CIFAR-10 dataset. ReLU net results are reported from (Glorot et al, 2011).

	Complexity	Time [s]
Linear SVM	$O(n)$	0.4
RBF kernel SVM	$O(nm)$	154
Sparse coding	$O\left(\frac{nN}{\sqrt{\epsilon}}\right)$[4]	14 [5]
LAST classifier	$O(nN)$	1.0

Table 5 Computational complexity for classifying one test sample, and time needed to predict the labels of the 10000 test samples in the MNIST dataset. For reference, all the experiments are carried out on a 2.6 GHz Intel Core i7 machine with 16 GB RAM.

over, LAST significantly outperforms the generic SGD training algorithm (by more than 6%) in this challenging classification example. What is more surprising is that LAST significantly surpasses the rectifier neural network with 3 hidden layers (Glorot et al, 2011) trained using a generic stochastic gradient descent algorithm (with or without pre-training). This shows that, despite the simplicity of our architecture (it can be seen as one hidden layer), the adequate training of the classification scheme can give better performance than complicated structures that are potentially difficult to train. We finally report the results of sparse coding classifier with a dictionary trained using Eq. (6). If we use a dictionary with 400 atoms, we get an error of 53.9%. By using a much larger dictionary of 4000 atoms, the error reduces to 46.5%. The computation of the test features is however computationally very expensive in that case.

6 Discussion

We first discuss in this section aspects related to the computational complexity of LAST. Then, we analyze the sparsity of the obtained solutions. We finally explain some of the differences between LAST and the generic stochastic gradient descent algorithm.

[4] The complexity reported here is that of the FISTA algorithm Beck and Teboulle (2009), where ϵ denotes the required precision. Note that another popular method for solving sparse coding is the homotopy method, which is efficient in practice, however it has exponential theoretical complexity Mairal and Yu (2012).

[5] To provide a fair comparison with our method, we used dictionaries of the same size as for our proposed approach, for the sake of this experiment.

6.1 Computational complexity at test time

We compare the computational complexity and running times of LAST classifier to the ones of different classification algorithms. Table 5 shows the computational complexity for classifying one test sample using various classifiers and the time needed to classify MNIST test images. We recall that n, m, and N denote respectively the signals dimension, the number of training samples and the dictionary size. Clearly, linear classification is very efficient as it only requires the computation of one inner product between two vectors of dimension n. Nonlinear SVMs however have a test complexity that is linear in the number of support vectors, which scales linearly with the training size (Burges, 1998). This solution is therefore not practical for relatively large training sets, like MNIST or CIFAR-10. Feature extraction with sparse coding involves solving an optimization problem, which roughly requires $1/\sqrt{\epsilon}$ matrix-vector multiplications, where ϵ controls the precision (Beck and Teboulle, 2009). For a typical value of $\epsilon = 10^{-6}$, the complexity becomes $1000nN$ (neglecting other constants), that is 3 orders of magnitude larger than the complexity of the proposed method. This can be seen clearly in the computation times, as our approach is slightly more expensive than linear SVM, but remains much faster than other methods. Note moreover that the soft-thresholding classification scheme is very simple to implement in practice at test time, as it is a direct map that only involves max and linear operations.

6.2 Sparsity

Sparsity is a highly beneficial property in representation learning, as it helps decomposing the factors of variations in the data into high level features (Bengio et al, 2013; Glorot et al, 2011). To assess the sparsity of the learned representation, we compute the average sparsity of our representation over all data points (training and testing combined) on the MNIST and CIFAR-10 dataset. We obtain an average of 96.7% zeros in the MNIST case, and 95.3% for CIFAR-10. In other words, our representations are very sparse, without adding an explicit sparsity penalization as in (Glorot et al, 2011). Interestingly, the reported average sparsity in (Glorot et al, 2011) is 83.4% on MNIST and 72.0% on CIFAR-10. Our one-layer representation therefore exhibits an interesting sparsity property, while providing good predictive performance.

6.3 LAST vs. stochastic gradient descent

As discussed earlier, the soft-thresholding classification scheme belongs to the more general neural network models. Neural networks are commonly optimized with stochastic gradient descent algorithms, as opposed to the DC method proposed in this paper. The proposed learning algorithm has several advantages compared to SGD:

- **Better local minimum:** In all our experiments, LAST reached a better solution than SGD in terms of the testing accuracy. This confirms the observations of Tao and An (1998) whereby DCA converges to "good" local minima, and often to global minima in practice.
- **Descent method:** Unlike stochastic gradient descent, LAST (and more generally DCA) is a descent method. Moreover, it is guaranteed to converge to a critical point (Tao and An, 1998).
- **No stepsize selection:** Stochastic gradient descent (and more generally gradient descent based algorithms) are very sensible to the difficult choice of the stepsize. Choosing a large stepsize in SGD can be beneficial as it helps escaping local minimas, but it can also lead to an oscillatory behaviour that prevents convergence. Interestingly, our optimization algorithm does not involve any stepsize selection, when given a convex optimization solver. In fact, our algorithm solves a sequence of convex problems, which can be solved with any off-the-shelf convex solver. Note that even if the intermediate convex optimization problems are solved with a gradient-descent based technique, the choice of the stepsize is less challenging as we have a better understanding of the theoretical properties of stepsize rules in convex optimization problems.

As we have previously mentioned, unlike SGD, our algorithm assumes the sign vector of the linear classifier $\mathbf{w}$ to be known. A simple heuristic choice of this parameter was shown however to provide very good results in the experiments, compared to SGD. Of course, choosing this parameter with cross-validation might lead to better results, but also implies a slower training procedure.

7 Conclusion

We have proposed a supervised learning algorithm tailored for the soft thresholding based classifier. The learning problem, which jointly estimates a discriminative dictionary $\mathbf{D}$ and a classifier hyperplane $\mathbf{w}$ is cast as a DC problem and solved efficiently with an iterative algorithm. The proposed algorithm (LAST), which leverages the DC structure, significantly outperforms stochastic gradient descent in all our experiments. Furthermore, the resulting classifier consistently leads to better results than the unsupervised sparse coding classifier. Our method moreover compares favorably to other standard techniques as linear, RBF kernel or nearest neighbour classifiers. The proposed LAST classifier has also been shown to compete with recent discriminative sparse coding techniques in handwritten digits classification experiments. We should mention that, while the sparse coding encoder features some form of competition between the different atoms in the dictionary (often referred to as *explaining-away* (Gregor and LeCun, 2010)), our encoder acts on the different atoms independently. Despite its simple behavior, our scheme is competitive when the dictionary and classifier parameters are learned in a suitable manner.

The classification scheme adopted in this paper can be seen as a one hidden layer neural network with a soft-thresholding activation function. This activation function has recently gained significant attention in the deep learning community, as it is believed to make the training procedure easier and less prone to bad local minima. Our work reveals an interesting structure of the optimization problem for the one-hidden layer version of that network that allows to reach good minima. An interesting question is whether it is possible to find a similar structure for networks with many hidden layers. This would help the training of deep networks, and offer insights on this challenging problem, which is usually tackled using stochastic gradient descent.

A Soft-thresholding as an approximation to non-negative sparse coding

We show here that soft-thresholding can be viewed as a coarse approximation to the non-negative sparse coding mapping (Denil and de Freitas, 2012). To see this, we consider the proximal gradient algorithm to solve the sparse coding problem with additional nonnegativity constraints on the coefficients. Specifically, we consider the following mapping

$$\underset{\mathbf{c}\in\mathbb{R}^N}{\operatorname{argmin}} \|\mathbf{x}-\mathbf{D}\mathbf{c}\|_2^2 + \lambda\|\mathbf{c}\|_1 \text{ subject to } \mathbf{c} \geq \mathbf{0}.$$

The proximal gradient algorithm proceeds by iterating the following recursive equation to convergence:

$$\mathbf{c}^{\mathbf{k+1}} = \text{prox}_{\lambda t\|\cdot\|_1 + \mathscr{I}_{\cdot\geq 0}}(\mathbf{c}^{\mathbf{k}} + t\mathbf{D}^{\mathbf{T}}(\mathbf{x}-\mathbf{D}\mathbf{c}^{\mathbf{k}})),$$

where prox is the proximal operator, t is the chosen stepsize and $\mathscr{I}_{\cdot\geq 0}$ is the indicator function, which is equal to 0 if all the components of the vector are nonnegative, and $+\infty$ otherwise. Using the definition of the proximal mapping, we have

$$\begin{aligned}\mathrm{prox}_{\lambda t\|\cdot\|_1+\mathscr{I}_{\cdot\geq 0}}(\mathbf{x}) &\triangleq \underset{\mathbf{u}\geq 0}{\mathrm{argmin}}\{\frac{1}{2}\|\mathbf{u}-\mathbf{x}\|_2^2+\lambda t\|\mathbf{u}\|_1\}\\ &= \max(0, \mathbf{x}-\lambda t).\end{aligned}$$

Therefore, imposing the initial condition $\mathbf{c^0}=\mathbf{0}$, and a step-size $t=1$, the first step of the proximal gradient algorithm can be written

$$\mathbf{c^1}=\max(0,\mathbf{D^T x}-\lambda)=h_\lambda(\mathbf{D^T x}),$$

which precisely corresponds to our soft-thresholding map. In this way, our soft-thresholding map corresponds to an approximation of sparse coding, where only one iteration of proximal gradient algorithm is performed.

B Proofs

B.1 Proof of Proposition 1

Before going through the proof of Proposition 1, we need the following results in (Horst, 2000, Section 4.2):

Proposition 3 *1. Let $\{f_i\}_{i=1}^l$ be DC functions. Then, for any set of real numbers $(\lambda_1,\ldots,\lambda_l)$, $\sum_{i=1}^l \lambda_i f_i$ is also DC.*
2. Let $f:\mathbb{R}^n\to\mathbb{R}$ be DC and $g:\mathbb{R}\to\mathbb{R}$ be convex. Then, the composition $g(f(\mathbf{x}))$ is DC.

We recall that the objective function of (P) is given by:

$$\sum_{i=1}^m L\left(y_i\sum_{j=1}^N s_j q(\mathbf{u_j^T x_i}-v_j)\right)+\frac{\nu}{2}\|\mathbf{v}\|_2^2,$$

The function $\|\mathbf{v}\|_2^2$ is convex and therefore DC. We show that the first part of the objective function is also DC. We rewrite this part as follows:

$$\sum_{i=1}^m L\left(\sum_{j:s_j=y_i} q(\mathbf{u_j^T x_i}-v_j)-\sum_{j:s_j\neq y_i} q(\mathbf{u_j^T x_i}-v_j)\right).$$

Since q is convex, $q(\mathbf{u_j^T x_i}-v_j)$ is also convex (Boyd and Vandenberghe, 2004). As the loss function L is convex, we finally conclude from Proposition 3 that the objective function is DC. Moreover, since the constraint $\mathbf{v}\geq\epsilon$ is convex, we conclude that (P) is a DC optimization problem.

B.2 Proof of Proposition 2

We now suppose that $L(x)=\max(0,1-x)$, and derive the DC form of the objective function. We have:

$$\begin{aligned}&\sum_{i=1}^m L\Big(y_i\sum_{j=1}^N s_j q(\mathbf{u_j^T x_i}-v_j)\Big)\\ &=\sum_{i=1}^m \max\Big(0,1+\sum_{j:s_j\neq y_i} q(\mathbf{u_j^T x_i}-v_j)-\sum_{j:s_j=y_i} q(\mathbf{u_j^T x_i}-v_j)\Big)\\ &=\sum_{i=1}^m \max\Big(\sum_{j:s_j=y_i} q(\mathbf{u_j^T x_i}-v_j)-\sum_{j:s_j=y_i} q(\mathbf{u_j^T x_i}-v_j),\\ &\qquad 1+\sum_{j:s_j\neq y_i} q(\mathbf{u_j^T x_i}-v_j)-\sum_{j:s_j=y_i} q(\mathbf{u_j^T x_i}-v_j)\Big)\\ &=\sum_{i=1}^m \max\Big(\sum_{j:s_j=y_i} q(\mathbf{u_j^T x_i}-v_j),1+\sum_{j:s_j\neq y_i} q(\mathbf{u_j^T x_i}-v_j)\Big)\\ &-\sum_{i=1}^m\sum_{j:s_j=y_i} q(\mathbf{u_j^T x_i}-v_j).\end{aligned}$$

The objective function of (P) can therefore be written as $g-h$, with:

$$\begin{aligned}g &= \tfrac{\nu}{2}\|\mathbf{v}\|_2^2+\sum_{i=1}^m\max\Big(\sum_{j:s_j=y_i} q(\mathbf{u_j^T x_i}-v_j),\\ &\qquad 1+\sum_{j:s_j\neq y_i} q(\mathbf{u_j^T x_i}-v_j)\Big),\\ h &= \sum_{i=1}^m\sum_{j:s_j=y_i} q(\mathbf{u_j^T x_i}-v_j),\end{aligned}$$

where g and h are convex functions.

Acknowledgments

The authors would like to thank the associate editor and the anonymous reviewers for their valuable comments and references that helped to improve the quality of this paper.

References

Akata Z, Perronnin F, Harchaoui Z, Schmid C (2014) Good practice in large-scale learning for image classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence* pp 507–520

An LTH, Tao PD (2005) The DC (difference of convex functions) programming and DCA revisited with dc models of real world nonconvex optimization problems. *Annals of Operations Research* 133(1-4):23–46

Beck A, Teboulle M (2009) A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences* 2(1):183–202

Bengio Y, Courville A, Vincent P (2013) Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35(8):1798–1828

Bishop CM (1995) Neural Networks for Pattern Recognition. *Oxford University Press, Inc.*

Boyd S, Vandenberghe L (2004) *Convex optimization.* Cambridge University Press

Burges C (1998) A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery* 2(2):121–167

Chang CC, Lin CJ (2011) LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology* 2:27:1–27:27

Chen CF, Wei CP, Wang YC (2012) Low-rank matrix recovery with structural incoherence for robust face recognition. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp 2618–2625

Coates A, Ng A (2011) The importance of encoding versus training with sparse coding and vector quantization. In: *International Conference on Machine Learning (ICML)*, pp 921–928

Denil M, de Freitas N (2012) Recklessly approximate sparse coding. *arXiv preprint arXiv:12080959*

Elad M, Aharon M (2006) Image denoising via sparse and redundant representations over learned dictionaries. *IEEE Transactions on Image Processing* 15(12):3736–3745

Fadili J, Starck JL, Murtagh F (2009) Inpainting and zooming using sparse representations. *The Computer Journal* 52(1):64–79

Fan RE, Chang KW, Hsieh CJ, Wang XR, Lin CJ (2008) LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research* 9:1871–1874

Glorot X, Bordes A, Bengio Y (2011) Deep sparse rectifier networks. In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*, vol 15, pp 315–323

Gregor K, LeCun Y (2010) Learning fast approximations of sparse coding. In: *International Conference on Machine Learning (ICML)*, pp 399–406

Horst R (2000) *Introduction to global optimization*. Springer

Huang K, Aviyente S (2006) Sparse representation for signal classification. In: *Advances in Neural Information Processing Systems*, pp 609–616

Hull JJ (1994) A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 16(5):550–554

Kavukcuoglu K, Ranzato M, LeCun Y (2010a) Fast inference in sparse coding algorithms with applications to object recognition. *arXiv preprint arXiv:10103467*

Kavukcuoglu K, Sermanet P, Boureau YL, Gregor K, Mathieu M, LeCun Y (2010b) Learning convolutional feature hierarchies for visual recognition. In: *Advances in Neural Information Processing Systems (NIPS)*, pp 1090–1098

Krizhevsky A, Hinton G (2009) Learning multiple layers of features from tiny images. Master's thesis, Department of Computer Science, University of Toronto

Larochelle H, Mandel M, Pascanu R, Bengio Y (2012) Learning algorithms for the classification restricted boltzmann machine. The Journal of Machine Learning Research 13:643–669

LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11):2278–2324

Ma L, Wang C, Xiao B, Zhou W (2012) Sparse representation for face recognition based on discriminative low-rank dictionary learning. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp 2586–2593

Maas A, Hannun A, Ng A (2013) Rectifier nonlinearities improve neural network acoustic models. In: *International Conference on Machine Learning (ICML)*

Mairal J, Yu B (2012) Complexity analysis of the lasso regularization path. In: *International Conference on Machine Learning (ICML)*, pp 353–360

Mairal J, Bach F, Ponce J, Sapiro G, Zisserman A (2008) Supervised dictionary learning. In: *Advances in Neural Information Processing Systems (NIPS)*, pp 1033–1040

Mairal J, Bach F, Ponce J, Sapiro G (2010) Online learning for matrix factorization and sparse coding. *The Journal of Machine Learning Research* 11:19–60

Mairal J, Bach F, Ponce J (2012) Task-driven dictionary learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34(4):791–804

Raina R, Battle A, Lee H, Packer B, Ng AY (2007) Self-taught learning: transfer learning from unlabeled data. In: *International Conference on Machine Learning (ICML)*, pp 759–766

Ramirez I, Sprechmann P, Sapiro G (2010) Classification and clustering via dictionary learning with structured incoherence and shared features. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp 3501–3508

Shawe-Taylor J, Cristianini N (2004) *Kernel methods for pattern analysis*. Cambridge University Press

Sriperumbudur BK, Torres DA, Lanckriet GR (2007) Sparse eigen methods by DC programming. In: *International Conference on Machine learning (ICML)*, pp 831–838

Tao PD, An LTH (1998) A DC optimization algorithm for solving the trust-region subproblem. *SIAM Journal on Optimization* 8(2):476–505

Valkealahti K, Oja E (1998) Reduced multidimensional co-occurrence histograms in texture classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20(1):90–94

Figueras i Ventura R, Vandergheynst P, Frossard P (2006) Low-rate and flexible image coding with redundant representations. *IEEE Transactions on Image Processing* 15(3):726–739

Wright J, Yang A, Ganesh A, Sastry S, Ma Y (2009) Robust face recognition via sparse representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31(2):210–227

Yang J, Yu K, Gong Y, Huang T (2009) Linear spatial pyramid matching using sparse coding for image classification. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp 1794–1801

Yuille A, Rangarajan A, Yuille A (2002) The concave-convex procedure (cccp). In: *Advances in Neural Information Processing Systems (NIPS)*, vol 2, pp 1033–1040

Zeiler M, Ranzato M, Monga R, Mao M, Yang K, Le Q, Nguyen P, Senior A, Vanhoucke V, Dean J, Hinton G (2013) On rectified linear units for speech processing. In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp 3517–3521

Zhang Y, Jiang Z, Davis L (2013) Learning structured low-rank representations for image classification. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp 676–683