Dictionary learning for fast classification based on soft-thresholding

Alhussein Fawzi* Mike Davies† Pascal Frossard*

February 9, 2014

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 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 built by applying 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* estimates the dictionary and linear classifier, is cast as a *difference of convex* (DC) program, and solved efficiently with an iterative DC solver. We conduct extensive experiments on several datasets, and show that our simple soft-thresholding based classifier competes with state-of-the-art sparse coding classifiers, when the dictionary is learned appropriately. Our classification scheme moreover achieves significant gains in terms of computational time at the testing stage, compared to other classifiers. The proposed scheme shows the potential of the soft-thresholding mapping for classification, and paves the way towards the development of very efficient classification methods for vision problems.

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 (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 sometimes achieve state-of-the-art results (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 this paper a computationally efficient sparsifying transform, the *soft-thresholding* mapping. Specifically, we consider a classification architecture that

*Ecole Polytechnique Federale de Lausanne (EPFL), Signal Processing Laboratory (LTS4), Lausanne 1015-Switzerland. Email: (alhussein.fawzi@epfl.ch, pascal.frossard@epfl.ch)

†IDCOM, The University of Edinburgh, Edinburgh, UK. Email: mike.davies@ed.ac.uk

first applies the soft-thresholding mapping in a dictionary $\mathbf{D}$, followed by a linear classifier denoted by $\mathbf{w}$. The soft-thresholding mapping has recently been shown to be useful in some classification applications (Coates and Ng, 2011). It is also popular in a number of deep learning architectures (Kavukcuoglu et al, 2010b), which suggests that it can act as a good feature extractor. 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 that of sparse coding, provided that the number of labeled samples and the dictionary size are large enough. However, when this is not the case, the adequate training of the scheme parameters $(\mathbf{D}, \mathbf{w})$ becomes crucial in the success of the soft thresholding based classifier.

We propose in this paper 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 soft-thresholding based scheme. 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, faces and digits datasets, and show that the proposed classifier, coupled with our dictionary learning approach, exhibits remarkable performance and efficiency gains with respect to numerous competitor methods. In particular, we show that our classifier provides comparable or better classification accuracy than sparse coding schemes.

To the best of our knowledge, this paper presents the first supervised dictionary learning approach tailored for the soft-thresholding encoder. Supervised dictionary learning for sparse coding has however been the focus of much recent research and has led to important improvements in classification accuracy. In particular, significant gains over the standard dictionary learning approaches are obtained (Mairal et al, 2012; Ramirez et al, 2010). 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 overcome this constraint and 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.

The rest of this paper is organized as follows. In Section 2, we formulate the dictionary learning problem for classifiers based on soft-thresholding. Section 3 then presents our novel learning algorithm, LAST, based on DC optimization. Finally, in Section 4, we perform extensive experiments on texture, faces and digits datasets.

2 Problem formulation

We formulate now the learning problem that is the core of our fast classification algorithm. We begin by introducing formally the soft-thresholding mapping and the classification architecture that we propose in this paper. We then present the joint dictionary and classifier learning problem that is tailored for soft-thresholding.

2.1 Soft-thresholding and linear classification

For any $\alpha \in \mathbb{R}$, we define the soft-thresholding map $h_\alpha : \mathbb{R} \to \mathbb{R}$ by

$$h_\alpha(z) = \max(0, z - \alpha) \triangleq (z - \alpha)_+,$$

with $(\cdot)_+ = \max(0, \cdot)$[1]. The map h_α is naturally extended to vectors $\mathbf{z}$ by applying the scalar map to each coordinate independently. This map can be applied to a transformed signal $\mathbf{z} = \mathbf{D^T x}$ that represents the coefficients of features in the signal $\mathbf{x}$. Its outcome, which only considers the most important features of $\mathbf{x}$ is used for classification. In more details, we adopt in this paper the following simple procedure for classification:

1. **Feature extraction:** Let $\mathbf{D} \in \mathbb{R}^{n \times N}$ and $\alpha \in \mathbb{R}$ Given a test point $\mathbf{x} \in \mathbb{R}^n$, compute $h_\alpha(\mathbf{D^T x})$.
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.

To simplify the exposition, we drop the bias term in the linear classifier. Note however that it can be easily included, and our study can be extended straightforwardly. 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.

[1] Note that soft-thresholding is usually defined as $\mathrm{sgn}(x)(|x| - \alpha)_+$. We consider only the positive part here.

The proposed scheme 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

$$\underset{\mathbf{c}\in\mathbb{R}^N}{\operatorname{argmin}} \|\mathbf{x}-\mathbf{D}\mathbf{c}\|_2^2+\lambda\|\mathbf{c}\|_1. \tag{1}$$

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). However, solving the optimization problem in Eq. (1) for each test sample is computationally expensive and limits the applicability of sparse coding classifiers. In an insightful unifying view, the soft-thresholding operation can be viewed as a very coarse approximation to sparse coding (see 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.

Motivated by low complexity considerations, we adopt the soft-thresholding based scheme and tackle in the next section the learning of parameters $(\mathbf{D},\mathbf{w})$, tailored for our soft-thresholding classification architecture.

2.2 Learning problem

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. We consider the binary classification task where $\mathbf{X}=[\mathbf{x_1}|\ldots|\mathbf{x_m}]\in\mathbb{R}^{n\times m}$ and $\mathbf{y}=[y_1|\ldots|y_m]\in\{-1,1\}^m$ denote respectively the set of training points and their associated labels. We consider the following supervised learning formulation

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

where L denotes a convex loss function that penalizes incorrect classification of a training sample and ν is a regularization parameter that prevents overfitting. Typical loss functions 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 $\mathbf{w}^{\mathbf{T}}(\mathbf{D}^{\mathbf{T}}\mathbf{x_i}-\alpha)_+$ 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.

The problem formulation in Eq. (2) 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. (2), we note that, for any $\alpha>0$, the objective function is equal to:

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

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.

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.

3 Learning algorithm

The problem in Eq. (2) 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.

3.1 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 $1 \leq i \leq m$,

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

Therefore, the problem in Eq.(2), with $\alpha = 1$, can be rewritten in the following way:

$$\begin{aligned} \underset{\mathbf{U},\mathbf{v},\mathbf{s}}{\text{argmin}} \sum_{i=1}^{m} L\left(y_i \sum_{j=1}^{N} s_j(\mathbf{u_j}^\mathbf{T}\mathbf{x_i} - v_j)_+\right) + \frac{\nu}{2}\|\mathbf{v}\|_2^2, \\ \text{subject to } \mathbf{v} > 0. \end{aligned} \tag{3}$$

The equivalence between the two problem formulations in Eqs. (2) and (3) 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. (2) 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.

After the above change of variables, we now approximate the term $(\mathbf{u_j}^\mathbf{T}\mathbf{x_i} - v_j)_+$ in Eq.(3) with a smooth function $q(\mathbf{u_j}^\mathbf{T}\mathbf{x_i} - v_j)$ where $q(x) = \frac{1}{\beta}\log(1 + \exp(\beta x))$, and β is a parameter that controls the accuracy of the approximation. 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. (3) 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 finally end up with the optimization problem:

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

that is a relaxed version of the learning problem in Eq. (3). Once the optimal variables $(\mathbf{U}, \mathbf{v})$ are determined, $\mathbf{D}$ and $\mathbf{w}$ can be obtained using the above change of variables.

We recall that at the testing stage, the classification of a test point $\mathbf{x}$ is performed by assigning it to class '$+1$' if $\mathbf{w}^\mathbf{T} h_1(\mathbf{D}^\mathbf{T}\mathbf{x}) > 0$, and class '$-1$' otherwise.

3.2 DC decomposition

The problem (P) is still a nonconvex optimization problem that is 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 $\min_{\mathbf{x}}\{f(x) : f_i(\mathbf{x}) \leq 0, i = 1, \ldots, m\}$, where $f_i = g_i - h_i$ 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*

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

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.

3.3 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 Tao et al (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.

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

$$\operatorname*{argmin}_{(\mathbf{U},\mathbf{v})}\{g(\mathbf{U},\mathbf{v}) - Tr(\mathbf{U}^T\mathbf{A}) - \mathbf{v}^T\mathbf{b}\} \text{ subject to } \mathbf{v} \geq \epsilon. \tag{4}$$

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. (4) 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, 2013). To make the exposition clearer, we first define the function:

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

The objective function of Eq. (4) 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.

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 equals $\rho t_0/t$.[2] Moreover, to accelerate the convergence of the stochastic gradient descent algorithm, we consider a small variation of

[2]The precise choice of the parameters ρ and t_0 are discussed later in Section 4.1.

Algorithm 1 Optimization algorithm to solve the linearized problem in Eq. (4)

1. Initialization: $\mathbf{U} \leftarrow \mathbf{U^k}$ and $\mathbf{v} \leftarrow \mathbf{v^k}$.
2. For $t = 1, \ldots, T$
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 $\mathbf{v^{k+1}} \leftarrow \mathbf{v}$.

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 $k = 0, \ldots, K-1$,
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:

$$(\mathbf{U^{k+1}}, \mathbf{v^{k+1}}) \leftarrow \underset{(\mathbf{U},\mathbf{v})}{\operatorname{argmin}} \{ g(\mathbf{U}, \mathbf{v}) - Tr(\mathbf{U}^T \mathbf{A}) - \mathbf{v}^T \mathbf{b} \}$$
$$\text{subject to } \mathbf{v} \geq \epsilon.$$

2.3 If $(\mathbf{U^{k+1}}, \mathbf{v^{k+1}}) \approx (\mathbf{U^k}, \mathbf{v^k})$, return $(\mathbf{U^{k+1}}, \mathbf{v^{k+1}})$.

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. (4) 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:

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

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).

4 Experimental results

In this section, we evaluate the performance of our classification algorithm on textures, faces and digits datasets, and compare it to different competitor schemes. We expose in a first section the strategy used to choose 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 4.2), we therefore *fix the encoder* to be the soft-thresholding mapping and compare LAST to existing learning techniques. Then, in the following subsections, we compare our 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 competes with state-of-the-art classifiers, despite its simplicity.

4.1 Parameter selection

We first discuss the choice of the model parameters for our method. In all the experiments, we have chosen to set the vector $\mathbf{s}$ according to the distribution of the different classes in the training set. In particular, for classification problems with equal representation of each class, we set half of the entries of the classifier sign vector $\mathbf{s}$ to 1 and the others to -1, to use the same ratio of class-specific atoms in the dictionary. We also 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 noticed however empirically that choosing a different initialization strategy does not significantly change the testing accuracy. Then, we fix the maximum number of iterations of DCA algorithm 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 algorithm.

4.2 Analysis of the learning algorithm

In a first set of experiments, we focus on the performance of our learning algorithm (LAST), by fixing the encoder to be the soft-thresholding mapping. We first provide a description of the different methods used in our experiments. Then, we present a comparative study on textures and faces classification tasks.

4.2.1 Experimental settings

We consider the following algorithms for learning the dictionary $\mathbf{D}$:

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 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:

$$\min_{\mathbf{D},\mathbf{c_i}} \sum_{i=1}^{m} \|\mathbf{x_i} - \mathbf{D}\mathbf{c_i}\|_2^2 + \lambda \|\mathbf{c_i}\|_1 \text{ subject to } \forall j, \|\mathbf{d_j}\|_2^2 \leq 1. \tag{5}$$

   To solve this optimization problem, we used the algorithm proposed in 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 two other learning algorithms make use of the labels, this algorithm is unsupervised.

After learning $\mathbf{D}$ using any of these above algorithms, the features are computed by applying the soft-thresholding mapping h_α, where α is chosen in the set $\{0.1, 0.2, \ldots, 0.9, 1\}$ by cross-validation. We finally train a linear SVM classifier on the feature space. Note that for the random samples and K-means approaches, we set $\kappa = 0.5$ as we consider classification tasks with roughly equal number of training samples from each class.

The resulting classification algorithms are compared to our scheme, where we learn jointly the dictionary $\mathbf{D}$ and the linear classifier $\mathbf{w}$ using LAST, and use the encoder h_α, with $\alpha = 1$.

4.2.2 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. 1. 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 image. The test data does not contain any of the training images. All the patches are moreover normalized to have unit ℓ_2 norm. Fig. 2 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.

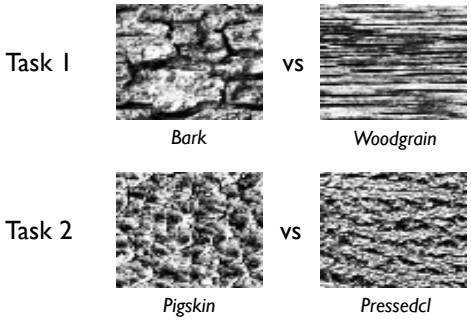
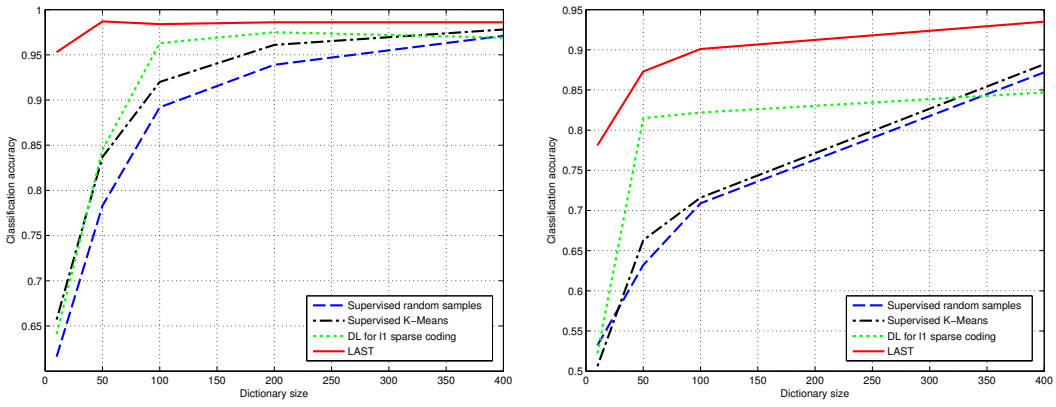


Figure 1: Two binary classification tasks (*bark* vs *woodgrain* and *pigskin* vs. *pressedcl*)

For the first task (*bark* vs. *woodgrain*), one can see that LAST significantly outperforms the other methods for small dictionaries. For large dictionaries (i.e., $N \approx 400$) however, all the learning algorithms yield approximately the same classification accuracy. This result agrees with the conclusions of Coates and Ng (2011), where the authors show emprirically 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 accuracies that are 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*

Figure 2: Texture classification results (fixed encoder)

Our second experiment focuses on face classification in the popular Extended Yale B faces database. This dataset consists of 2414 face images of 38 individuals (Georghiades et al, 2001). For each subject, we randomly select 30 images (i.e., approximately half the number of images per subject) for training and the rest for testing. All the images are normalized and cropped to 16×16 pixels. For simplicity and better interpretability of the results, we form two groups of subjects (Fig. 3), each containing 19 persons, and consider the binary classification task of predicting the group of a test face. We report the results in Fig. 4. Once again, the soft-thresholding based classifier with a dictionary and linear classifier learned with LAST outperforms all other dictionary learning techniques. This result has moreover an element of surprise as it shows that, using our learning algorithm, an accuracy of 80% can be reached with a dictionary of only 2 atoms, and 90% with only 4 atoms! On the other hand, other learning algorithms 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. 5 the 2-D features obtained with a dictionary of two atoms, when $\mathbf{D}$ is learned respectively with the K-Means method and LAST. Despite the

Figure 3: Two groups of subjects, where each row represents a group.

very low-dimensionality of the feature vectors, the two classes are nearly linearly separable with our algorithm (Fig. 5 (b)), whereas features obtained with the K-means algorithm clearly cannot be discriminated (Fig. 5 (a)).

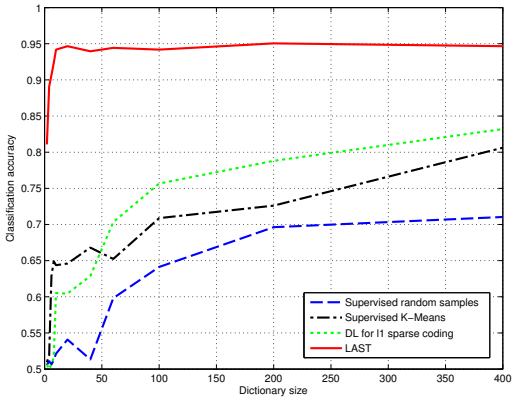
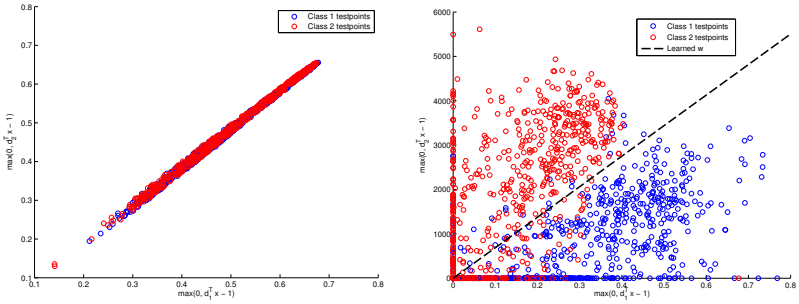


Figure 4: Face classification results (fixed encoder)



(a) K-means features

(b) Features learned with LAST, and learned linear classifier $\mathbf{w}$

Figure 5: 2D features obtained with K-Means and our approach.

We finally illustrate in Fig. 6 the dictionaries learned using K-Means and LAST for $N = 20$ atoms. We see that, while K-Means finds atoms that yield good reconstruction quality, our algorithm learns a dictionary that underlines the difference between the faces in different classes, which makes it readily suitable for classification tasks.

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 very high accuracy rates, even with very small dictionaries, which is not possible with other learning schemes.

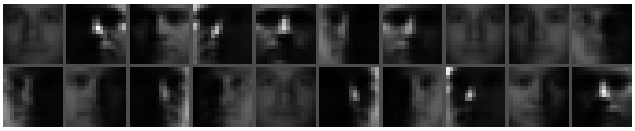
4.3 Classification performance on textures and faces datasets

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.

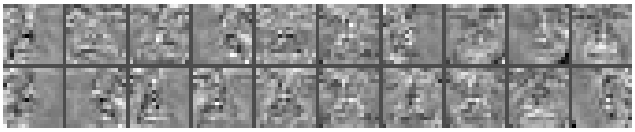
4.3.1 Experimental setting

We consider the following classification schemes:

[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.



(a)



(b)

Figure 6: Normalized dictionary of faces learned with (a) K-Means method and (b) LAST, with $N = 20$ atoms.

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 representation based classifier:** Similarly to the previous section, we train the dictionary by solving Eq. (5). 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:** Our last comparative scheme is a nearest neighbor classifier, denoted "NN", where the dictionary is learned using the supervised K-means procedure described in 4.2.1. 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 as it was shown in the previous experiments to have worse classification accuracy than the K-means approach.

4.3.2 Experimental results

Table 1 shows the accuracies of the different classifiers in the two binary textures classification tasks described in 4.2.2. 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. It is worth mentioning that, not only is our classifier more computationally efficient than sparse coding and RBF-SVM classifiers, but it also gives better classification accuracy in the experimented texture tasks.

	bark vs. *woodgrain* [%]	*pigskin* vs. *pressedcl* [%]
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.

We now turn to the binary faces experiments described in the previous subsection. We show the classification accuracies of the different classifiers in Table 2. LAST significantly outperforms sparse coding and nearest neighbour classifiers for small dictionaries. This result is notable as sparse coding is known to achieve good results in face classification (Wright et al, 2009). Our approach moreover shows a performance that is comparable to RBF-SVM. For reference, we also compute the classification performance of the sparse coding classifier, where the size of the dictionary is equal to the number of training samples[4]. This scheme achieves a slightly better classification performance with 96.2% of accuracy. This is not surprising as sparse representation based classifiers are well-adapted to tasks where datapoints live approximately in unions of subspace (Wright et al, 2009; Elhamifar and Vidal, 2013), which is the case in our face classification experiments. Note however that this improvement comes at the cost of (i) using a much larger dictionary, (ii) using the sparse coding non linear mapping, which is more computationally expensive to compute than our feed-forward non linear map.

	Accuracy [%]
Linear SVM	86.5
RBF kernel SVM	**95.4**
Sparse coding ($N = 10$)	58.2
Sparse coding ($N = 200$)	90.3
NN ($N = 10$)	54.6
NN ($N = 200$)	65.2
LAST ($N = 10$)	94.2
LAST ($N = 200$)	95.1

Table 2: Binary classification accuracy for face images.

We now consider imperfect settings where data samples are noisy. We corrupt a percentage π of randomly chosen pixels from each test image, replacing their values with independent and identically distributed samples from a uniform distribution. These experiments permit to assess the robustness of the different classifiers to random sparse noise (Wright et al, 2009). We limit ourselves to the best performing methods; that are linear, RBF-SVM, sparse coding ($N = 200$), and LAST ($N = 200$) classifiers. The results are reported in Fig. 7. In the low noise regime, LAST significantly outperforms all other methods, including RBF-SVM, which suggests that our method is more robust to noise. For a noise level of 40%, the classification performance of our method is comparable to that of sparse coding classifier[5]. Our method therefore is relatively robust to noise, as sparse coding classifiers are known to be robust to heavy levels of random noise (Wright et al, 2009).

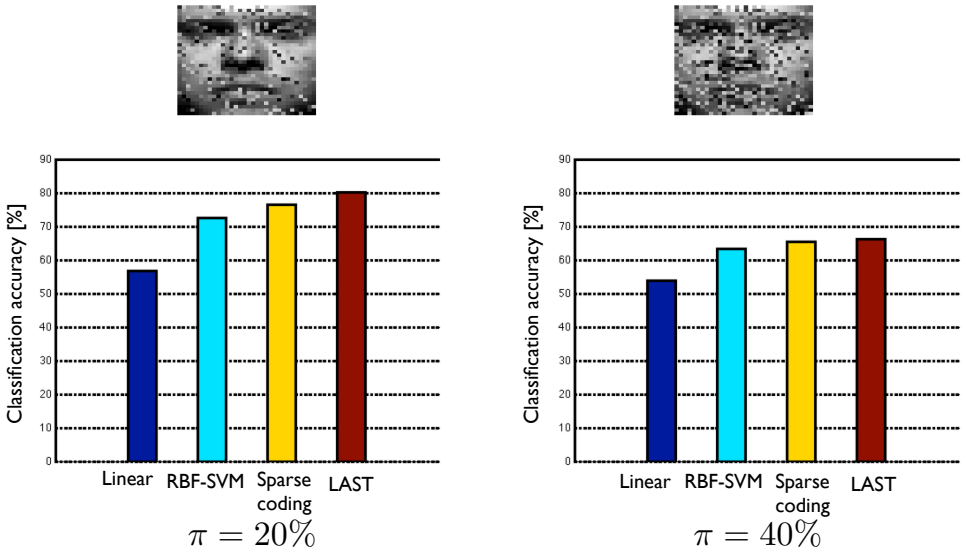


Figure 7: Classification accuracy in noisy settings. π denotes the percentage of corrupted pixels. The face images show examples of corrupted images.

[4]This classifier shares strong similarities with the popular SRC classifier introduced in Wright et al (2009). The main difference is however that SRC uses a classifier based on the Euclidean distance between the test image and its reconstruction using training images from one class, whereas we use here a linear classifier.

[5]In the noisy case, λ is set to 0.1, instead of choosing it with cross-validation. Indeed, having smaller values of λ resulted in significant performance drop for the sparse coding classifier in the noisy settings.

Overall, the proposed LAST classifier compares favorably to the different classifiers under comparison. In particular, LAST outperforms the sparse coding technique for a fixed dictionary size. 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 learning approach in 4.2.1, the soft-thresholding based classifier is outperformed by sparse coding, which shows the importance of the learning scheme in the success of this classifier.

4.4 Handwritten digits classification

In a final series of experiments, we consider a classification task on the MNIST (LeCun et al, 1998) and USPS (Hull, 1994) handwritten digits datasets. USPS contains 9298 images of size 16×16 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, and all images are moreover resized to a resolution of 16×16 pixels. The aim of this set of experiments is three-fold: (i) to evaluate our classifier in the multi-class settings, (ii) to compare our classifier to state-of-the art published results on these popular datasets and (iii) to show the generality of our approach with respect to different types of images.

We tackle the multi-class classification 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 (P) for each one-vs-all problem. Classification is then done by predicting using each binary classifier, and choosing the prediction with highest score. We recall that, 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. We use a minibatch stochastic subgradient descent procedure in Algorithm 1 to solve the linearized convex problem in each iteration of LAST, with a minibatch size of 200, and $T = 5000$. In our proposed approach, 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

	MNIST [%]	USPS [%]
Linear SVM	8.26	9.07
RBF kernel SVM	1.4	4.2
K-NN ℓ_2	5.0	5.2
LAST	1.23	4.53
Sparse coding (DL: Eq. 5)	2.85	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

Table 3: Classification error on MNIST and USPS datasets.

techniques 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) that 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 better accuracy than RBF-SVM in MNIST, while being slightly worse on the USPS dataset. Furthermore, LAST outperforms the unsupervised sparse representation classifier in both datasets. Interestingly, the proposed scheme also competes and often 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, LAST classifier is much faster at test time than sparse coding techniques and RBF-SVM classifiers. Note finally that the obtained classification errors are comparable to best results[6] in the literature (Jarrett et al, 2009), which are often reached using methods that are specifically tuned to handwritten digits recognition, while our framework is general and is not specifically adapted to the particular task at hand.

4.5 Computational complexity at test time

We compare now the computational complexity and running times of LAST classifier to the ones of different classification algorithms. Table 4 shows the computational complexity for classifying one test sample using

[6] We focus on techniques that do not augment the training set with distorted/transformed samples.

[7] 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).

[8] 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.

	Complexity	Time [s]
Linear SVM	$O(n)$	0.2
RBF kernel SVM	$O(nm)$	60.6
Sparse coding	$O\left(\frac{nN}{\sqrt{\epsilon}}\right)^7$	10.2^8
LAST classifier	$O(nN)$	0.53

Table 4: 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, 16 GB RAM laptop.

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. 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 several orders of magnitude faster than other methods. Note moreover that our classifier is very simple to implement in practice at test time, as it is a direct map that only involves max and linear operations.

5 Conclusion

We 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 DCA. The resulting classifier consistently outperformed the unsupervised sparse coding classifier in all the experiments. Our method moreover compares favorably to other standard techniques as linear, RBF kernel or nearest neighbour classifiers. Moreover, the proposed LAST classifier was shown to compete with state-of-the-art 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. Using the simple nonlinear soft-thresholding function, one can therefore considerably extend the power of linear classifiers to nonlinear classification tasks, with a very small additional computational cost. Our result paves the way towards a more extended use of the soft-thresholding map for nonlinear classification tasks. In a future work, we plan to extend our study in order to explain in details the success of this simple classification scheme, and provide solid performance guarantees.

A Soft-thresholding as an approximation to sparse coding

We show here that soft-thresholding can be viewed as a coarse approximation to the sparse coding mapping (Denil and de Freitas, 2012). To see this, we consider the popular Iterative Soft Thresholding Algorithm (ISTA) (Daubechies et al, 2004) to solve the sparse coding problem in Eq. (1) with additional nonnegativity constraints on the coefficients. Specifically, we consider the following sparse coding mapping

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

ISTA proceeds by iterating the following recursive equation to convergence:

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

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

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

Therefore, imposing the initial condition $\mathbf{c^0} = \mathbf{0}$, and a stepsize $t = 1$, the first step of ISTA 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 ISTA 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.

References

Akata Z, Perronnin F, Harchaoui Z, Schmid C (2013) Good practice in large-scale learning for image classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*

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

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

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

Daubechies I, Defrise M, De Mol C (2004) An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on pure and applied mathematics* 57(11):1413–1457

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

Elhamifar E, Vidal R (2013) Sparse subspace clustering: Algorithm, theory, and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35(11):2765–2781

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

Georghiades AS, Belhumeur PN, Kriegman DJ (2001) From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 23(6):643–660

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

Jarrett K, Kavukcuoglu K, Ranzato M, LeCun Y (2009) What is the best multi-stage architecture for object recognition? In: *International Conference on Computer Vision (ICCV)*, pp 2146–2153

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

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

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

Tao PD, et al (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

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