Learning Sparse Features with an Auto-Associator

Sébastien Rebecchi, Hélène Paugam-Moisy, Michèle Sebag

Learning Sparse Features with an Auto-Associator

Sébastien Rebecchi, Hélène Paugam-Moisy and Michèle Sebag

Abstract

A major issue in statistical machine learning is the design of a representation, or feature space, facilitating the resolution of the learning task at hand. Sparse representations in particular facilitate discriminant learning: On the one hand, they are robust to noise. On the other hand, they disentangle the factors of variation mixed up in dense representations, favoring the separability and interpretation of data. This chapter focuses on auto-associators (AAs), i.e. multi-layer neural networks trained to encode/decode the data and thus *de facto* defining a feature space. AAs, first investigated in the 80s, were recently reconsidered as building blocks for deep neural networks.
This chapter surveys related work about building sparse representations, and presents a new non-linear explicit sparse representation method referred to as *Sparse Auto-Associator* (SAA), integrating a sparsity objective within the standard auto-associator learning criterion. The comparative empirical validation of SAAs on state-of-art handwritten digit recognition benchmarks shows that SAAs outperform standard auto-associators in terms of classification performance and yield similar results as denoising auto-associators. Furthermore, SAAs enable to control the representation size to some extent, through a conservative pruning of the feature space.

Sébastien Rebecchi
CNRS, LRI UMR 8623, TAO, INRIA Saclay, Université Paris-Sud 11, F-91405 Orsay,
e-mail: sebastien.rebecchi@lri.fr

Hélène Paugam-Moisy
CNRS, LIRIS UMR 5205, Université Lumière Lyon 2, F-69676 Bron,
e-mail: helene.paugam-moisy@liris.cnrs.fr

Michèle Sebag
CNRS, LRI UMR 8623, TAO, INRIA Saclay, Université Paris-Sud 11, F-91405 Orsay,
e-mail: michele.sebag@lri.fr

1 Introduction

A major issue in statistical machine learning is the design of a representation, or *feature space*, facilitating the resolution of the learning task at hand. For instance, binary supervised learning often considers linear hypotheses, although a hyper-plane actually separating the two classes seldom exists for real-world data (non-separable data). A standard approach thus consists of mapping the initial input space onto a, usually high-dimensional, feature space, favoring the data separability. The feature space can be i) explicitly defined using hand-crafted features; ii) implicitly defined using the so-called kernel trick [32]; iii) automatically learned by optimizing a criterion involving the available data. In the former two cases, the feature space ultimately relies on the user's expertise and involves a trial-and-error process, particularly so when dealing with high-dimensional data (e.g. images). In the third case, the feature space optimization considers either a linear search space (dictionary learning [26]) or a non-linear one, through neural networks.

This chapter focuses on the trade-off between the expressiveness and the computational efficiency of non-linear feature space optimization. On the one hand, non-linear functions can be represented more efficiently and compactly (in terms of number of parameters) by iteratively composing non-linear functions, defining deep architectures where each layer defines a more complex feature space elaborated on the basis of the previous one [3]. Indeed, the principle of hierarchically organized representations − corresponding to increasing levels of abstraction − is appealing in a cognitive perspective [31]. The direct optimization of deep architectures however raises severe challenges in a supervised setting: while the standard back-propagation of the gradient is effective for shallow neural architectures, the gradient information is harder to exploit when the number of layers increases. The breakthrough of deep neural architectures since the mid-2000s [4, 15] relies on a new training principle, based on the layer-wise unsupervised training of each layer. Note that the specific layer training depends on the particular setting, involving restricted Boltzmann machines [15], auto-associators [4] or convolutional networks [22, 25]. Only at a later stage is the pre-trained deep neural network architecture optimized (fine-tuned) using a supervised learning criterion (Figure 1). The chapter focuses on feature space optimization within the auto-associator-based deep neural network setting.

Formally, an auto-associator (AA) is a one-hidden layer neural network implementing an encoding-decoding process [5, 13]. The AA is trained by backpropagation to reconstruct the instances in the dataset. Its hidden layer thus defines a feature space. In the context of deep networks, this first feature-based representation of the data is used to feed another AA, which is likewise trained as an encoder/decoder, thus defining a second layer feature space.

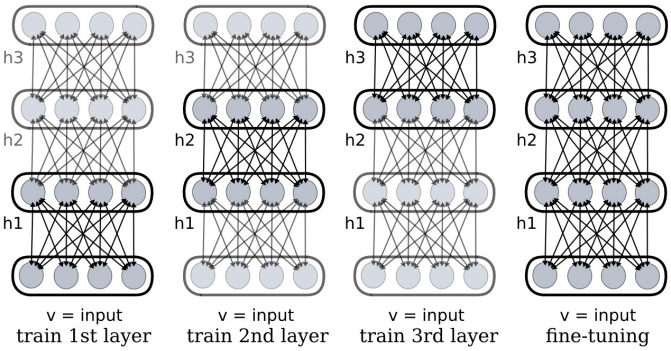

Fig. 1 Layer-wise training scheme of deep neural networks (e.g. stacked auto-associators or stacked RBMs) and overall fine-tuning.

The process is iterated in a layer-wise manner, thus defining a deep neural architecture. At each layer, the feature space can be assessed from its coding/decoding performance, a.k.a. the average reconstruction error or *accuracy* on the one hand, and its size or complexity on the other hand. As could have been expected, the quality of k-th layer feature space commands the quality of the $(k+1)$-th layer feature space, specifically so in terms of reconstruction error.

The sparsity of the feature space, that is, the fact that the coding of each data instance involves but a few features, is highly desirable for several reasons [3, 10, 28]. Firstly, sparse representations offer a better robustness w.r.t. the data noise. Secondly, they facilitate the interpretation of the data by disentangling the factors of variations mixed up in dense representations, and thus favor the linear separability of the data. Along this line, the complexity of the feature space might be assessed from its *sparsity*, i.e. the average number of features that are actively involved in example coding.
Formally, let $\mathbf{x}$ denote an instance in $\mathbb{R}^n$, let $\mathbf{y} \in \mathbb{R}^m$ denote its mapping onto the feature space. Feature i $(i = 1 \ldots m)$ is said to be active in $\mathbf{y}$ iff its cancelling out significantly decreases the ability to reconstruct $\mathbf{x}$ from $\mathbf{y}$. Sparse coding, involving a low percentage of active features on average on the data, has been intensively studied in the last decade, in relation with compressed sensing [6], signal decomposition [7] and dictionary learning [26]. The presented approach, referred to as *Sparse Auto-Associator* (SAA), focuses on integrating a sparsity-inducing approach within the auto-associator framework.

The chapter is organized as follows. Section 2 presents several points of view related to learning sparse representations.The domain of sparse feature space learning has been advancing at a fast pace in the recent years and the present chapter admittedly has no vocation to present an exhaustive survey. The standard auto-associator model and its best known variant,

the denoising auto-associator (DAA), are introduced in Section 3. Section 4 describes the proposed SAA algorithm, based on an alternate non-linear optimization process enforcing both accuracy and sparsity criteria. The Sparse Auto-Associator is experimentally validated in Section 5 and discussed in Section 6. Section 7 summarizes the contributions and presents some perspectives on non-linear learning of sparse features.

2 Learning Sparse Representations of Data

Originated from signal processing, the *sparse coding* domain essentially aims at expressing signals using a (given or learned) *dictionary*. This section briefly reviews some recent work related to sparse coding, in particular within the field of neural networks.

2.1 Dictionary-based sparse coding

Sparse coding a (n-dimensional) raw data signal $\mathbf{x}$ aims at finding a linear decomposition of $\mathbf{x}$ using a small number of atoms of a given dictionary $\mathbf{D}$. Formally, $\mathbf{D}$ is a $(n \times m)$ real matrix, its m columns vectors $\mathbf{D_{*j}}$ denoting the atoms of the dictionary. The sparse code $\mathbf{y}$ is obtained by solving a combinatorial optimization problem, minimizing its L_0-norm $\|.\|_0$ subject to the reconstruction constraint:

$$\min_{\mathbf{y}\in\mathbb{R}^m} \|\mathbf{y}\|_0 \quad \text{subject to} \quad \mathbf{x} = \sum_{j=1}^{m} (\mathbf{D_{*j}} y_j), \tag{1}$$

where y_j denotes the j-th entry of $\mathbf{y}$. An approximate resolution of Eq. (1) is yielded by the greedy Orthogonal Matching Pursuit (OMP) algorithm [30], subject to upper bounding $\|\mathbf{y}\|_0$.

A convex relaxation of Eq. 1 is obtained by replacing the L_0-norm by the L_1-norm and minimizing a weighted sum of the reconstruction loss and the sparsity:

$$\min_{\mathbf{y}\in\mathbb{R}^m} \left(\left\| \mathbf{x} - \sum_{j=1}^{m} (\mathbf{D_{*j}} y_j) \right\|_2^2 + \lambda \|\mathbf{y}\|_1 \right), \tag{2}$$

where $\lambda > 0$ is the trade-off parameter, $\|.\|_1$ and $\|.\|_2$ denote the L_1-norm and L_2-norm, respectively. The minimization problem defined by Equation (2) is known as Basis Pursuit (BP) denoising [7]. Under some conditions [9], the unique solution of Eq. (2) also is solution of Equation (1).

In both cases the objective is to find an approximate decomposition of $\mathbf{x}$ involving a small number of atoms of $\mathbf{D}$. The level of activity of $\mathbf{D_{*j}}$ relatively to $\mathbf{x}$ is $|y_j|$. Canceling the entry j, i.e. ignoring $\mathbf{D_{*j}}$ in the reconstruction process, consists in setting y_j to 0, thus incurring some additional reconstruction loss, the level of which increases with $|y_j|$. Canceling entries with very low levels of activity clearly entails a small loss of accuracy.

As a given dictionary may not provide every example with a sparse decomposition, it comes naturally to optimize the dictionary depending on the available examples [1, 26]. If sparsity is measured w.r.t. L_1-norm then *dictionary learning* for sparse coding is achieved by solving the following joint optimization problem:

$$\min_{\mathbf{D}\in\mathbb{R}^{n\times m},\mathbf{Y}\in\mathbb{R}^{m\times l}} \left(\|\mathbf{X}-\mathbf{D}\mathbf{Y}\|_2^2 + \lambda \sum_{k=1}^{l} \|\mathbf{Y_{*k}}\|_1 \right), \tag{3}$$

with l the number of training examples, $\mathbf{X}$ the $(n \times l)$ real matrix storing the training examples in columns and $\mathbf{Y}$ the $(m \times l)$ real matrix storing the sparse representation of the k-th training example $\mathbf{X_{*k}}$ in its k-th column $\mathbf{Y_{*k}}$.

Besides the general merits of sparse coding, mentioned in Section 1, sparse dictionary coding features specific strengths and weaknesses. On the positive side, the minimization-based formulation in Equation (3) yields a variable-size representation, since some examples may be reconstructed from very few dictionary atoms (i.e. active components) while other examples may require many more dictionary atoms. On the negative side, dictionary learning defines an implicit and computationally expensive sparse coding; formally, each new $\mathbf{x}$ requires a BP optimization problem (Equation (2)) to be solved to find the associated code $\mathbf{y}$. A second limitation of sparse dictionary learning is to be restricted to linear coding.

2.2 Sparse coding within a neural network

Aimed at overcoming both above limitations of dictionary learning, several neural network-based architectures have been defined to achieve sparse coding, through considering additional learning criteria. The main merit thereof is to yield a non-linear coding, directly computable for unseen patterns without solving any additional optimization problem. Another merit of such models is to accommodate the iterative construction of sparse non-linear features within a Deep Network architecture.

As already mentioned, sparse coding within the deep learning framework is a hot topic. Without pretending at an exhaustive review of the state of the art, this section summarizes the main four directions of research, which

will be discussed comparatively to the proposed approach in section 6.2.

Gregor and LeCun use a supervised approach to enforce sparse coding, where each input example is associated its optimal sparse representation (obtained through conventional dictionary-based methods). The neural encoder is trained to approximate this optimal code [12]. The proposed architecture interestingly involves a *local competition* between encoding neurons; specifically, if two (sets of neurons) can reconstruct an input equally well, the algorithm activates only one of them and deactivates the other.

Another strategy for encouraging sparse codes in a neuron layer is to apply back-propagation learning with an activation function that naturally favors sparsity, such as proposed by Glorot et al. with the so-called *rectifying* functions [10]. Rectifying neurons are considered more biologically plausible than sigmoidal ones. From the engineering viewpoint, rectifying neurons can be used in conjunction with an additional constraint, such as L_1-norm regularization, to further promote sparsity. Rectifying functions however raise several computational issues, adversely affecting the gradient descent used for optimization. The authors propose several heuristics to tackle these issues.

Lee et al. augment the Restricted Boltzmann Machine (RBM) learning rule [15] by setting a *temporal* sparsity constraint on each individual encoding neuron [24]. This constraint requires the average value ρ of encoding neuron activations to be close to the minimum of the activation $\hat{\rho}_i$ of each encoding neuron n_i. In other words, each encoding neuron is enforced to be active for a small number of input examples only. By limiting the average activity of each neuron, this *selectivity* property tends to favor the code sparsity, as it makes it unlikely that many encoding neurons are active for many examples. In particular, a method for updating selectivity consists in minimizing the cross-entropy loss between ρ and $\hat{\rho}_i$ [14].

Goh et al. propose another modification to the RBM learning rule in order to simultaneously favor both the sparsity and the selectivity of the model [11]. The method consists in computing a matrix $\mathbf{M}$ which stores the training example components in its columns and the encoding neuron activations in its rows, so that the *selectivity* (resp. *sparsity*) property can be evaluated horizontally (resp. vertically). $\mathbf{M}$ is modified in order to fit some target distribution P before updating the model. P encodes the prior knowledge about the problem domain: e.g. for image datasets, the authors consider a distribution P which is positively skewed with heavy tails, since similar characteristics of activity distribution for both selectivity and sparsity have been observed from recordings in biological neural networks. Notably, this approach requires batch learning.

After briefly introducing auto-associator (AA) learning for the sake of completeness (section 3), the rest of the chapter will present a new way of enforcing sparsity within AA learning (section 4) and report on its comparative validation (section 5).

3 Auto-Associator and Denoising Auto-Associator

The AA model, auto-associator a.k.a. auto-encoder, is a two-layer neural network trained for feature extraction [5, 13].

The first layer is the *encoding layer* or *encoder* and the second layer is the *decoding layer* or *decoder*. Given an input example $\mathbf{x}$, the goal of an AA is to compute a code $\mathbf{y}$ from which $\mathbf{x}$ can be recovered with high accuracy, i.e. to model a two-stage approximation to the identity function:

$$\begin{cases} \mathbf{y} = f^{\mathrm{E}}(\mathbf{x}) = a^{\mathrm{E}}(\mathbf{W}^{\mathrm{E}}\mathbf{x} + \mathbf{b}^{\mathrm{E}}), \\ \hat{\mathbf{x}} = f^{\mathrm{D}}(\mathbf{y}) = a^{\mathrm{D}}(\mathbf{W}^{\mathrm{D}}\mathbf{y} + \mathbf{b}^{\mathrm{D}}), \\ \qquad\quad \hat{\mathbf{x}} \simeq \mathbf{x}, \end{cases} \tag{4}$$

where f^{E} and f^{D} denote the function computed by the encoder and decoder, respectively. Parameters are the weight matrices, bias vectors and activation functions, respectively denoted by $\mathbf{\Theta}^{\mathrm{E}} = \{\mathbf{W}^{\mathrm{E}}, \mathbf{b}^{\mathrm{E}}\}$ and a^{E} for the encoder, and $\mathbf{\Theta}^{\mathrm{D}} = \{\mathbf{W}^{\mathrm{D}}, \mathbf{b}^{\mathrm{D}}\}$ and a^{D} for the decoder. The weight matrix $\mathbf{W}^{\mathrm{D}}$ is often (although not necessarily) the transpose of $\mathbf{W}^{\mathrm{E}}$. Since the AA has its target output same as its input, the decoder output dimension (number of neurons) equals the encoder input dimension.

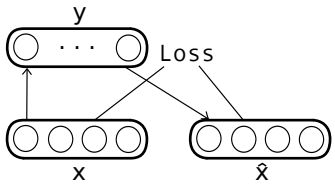

Fig. 2 The training scheme of an AA. The features $\mathbf{y}$ are learned to encode the input $\mathbf{x}$ and are decoded into $\hat{\mathbf{x}}$ in order to reconstruct $\mathbf{x}$.

The training scheme of an AA (Figure 3) consists in finding parameters $\mathbf{\Theta} = \mathbf{\Theta}^{\mathrm{E}} \cup \mathbf{\Theta}^{\mathrm{D}}$ (weights and biases) that minimize a reconstruction loss on a training dataset $\mathcal{S}$, with the following objective function:

$$\phi(\mathbf{\Theta}) = \sum_{\mathbf{x} \in \mathcal{S}} \mathcal{L}(\mathbf{x}, f^{\mathrm{D}} \circ f^{\mathrm{E}}(\mathbf{x})), \tag{5}$$

where $\mathcal{L}$ denotes the loss function.

An AA is usually trained by gradient descent, applying standard back-propagation of error derivatives [29] with $\mathbf{x}$ as target. Depending on the nature of the input examples, $\mathcal{L}$ can either be the traditional squared error or the cross-entropy for vectors of valued in $[0, 1]$ interpreted as probabilities:

$$\text{squared\_error}(\mathbf{x}, \hat{\mathbf{x}}) = \sum_{i=1}^{n} (\hat{x}_i - x_i)^2, \tag{6}$$

$$\text{cross-entropy}(\mathbf{x}, \hat{\mathbf{x}}) = \sum_{i=1}^{n} \left[x_i \log(\hat{x}_i) + (1 - x_i) \log(1 - \hat{x}_i) \right]. \tag{7}$$

With a linear encoding activation function a^{E}, an AA actually emulates principal component analysis [2]. When using non-linear a^{E} (e.g. a sigmoid function), an AA can learn more complex representations and shows able to capture multi-modal aspects of the input distribution [17].

The denoising auto-associator (DAA) model is an AA variant that aims to remove input noise, in addition to extracting a non-linear feature space [33]. To achieve the denoising goal, the training example fed to a DAA is a corrupted version $\tilde{\mathbf{x}}$ of $\mathbf{x}$. There exists many ways of corrupting $\mathbf{x}$. The simplest corruption rule proceeds by cancelling out some uniformly selected components of $\mathbf{x}$. DAA thus involves an additional hyper-parameter compared to AA, namely the input corruption rate ν. The training scheme of a DAA is illustrated in Figure 3.

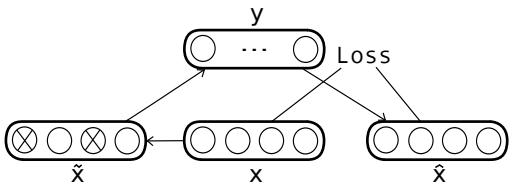

Fig. 3 The training scheme of a DAA. Input components modified by the corruption process are marked with a $\otimes$ cross.

The objective function is optimized by back-propagation, as in an AA:

$$\phi^{\mathrm{D}}(\boldsymbol{\Theta}) = \sum_{\mathbf{x} \in \mathcal{S}} \mathcal{L}(\mathbf{x}, f^{\mathrm{D}} \circ f^{\mathrm{E}}(\tilde{\mathbf{x}})) \quad \text{with} \quad \tilde{\mathbf{x}} = \text{corrupt}(\mathbf{x}). \tag{8}$$

Compared to the standard AA, the DAA learning rule forces the encoding layer to learn more robust features of the inputs, conducive to a better discriminative power. DAA learning can be thought of as manifold learning: the encoding layer maps the corrupted examples as close as possible to the manifold of the true examples.

4 Sparse Auto-Associator

The proposed sparse auto-associator (SAA) only differs from standard AA as its learning rule is modified to favor sparse representations on the encoding (hidden) layer.

In numerical analysis and signal processing, a sparse vector is traditionally defined as a vector with small L_0-norm, i.e. involving few non-zero components. In the following, an encoder representation be sparse iff it involves few active neurons, where an *active* neuron is one with activation value close to the maximum of the activation function. For instance, if $a^{\mathrm{E}} = \tanh$ then the encoding neuron n_i is considered maximally active for $\mathbf{x}$ if $y_i = 1$ and maximally inactive for $\mathbf{x}$ if $y_i = -1$.

In contrast with [12], we do not require any training example $\mathbf{x}$ to be provided with its target optimal sparse representation $\tilde{\mathbf{y}}$ to achieve sparse encoding through supervised learning. The working assumption used in the following is that the code $\mathbf{y}$, learned using standard AA to encode input $\mathbf{x}$ *constitutes* a non-sparse approximation of the desired sparse $\tilde{\mathbf{y}}$. Accordingly, one aims at recovering $\tilde{\mathbf{y}}$ by sparsifying $\mathbf{y}$, i.e. by cancelling out the least active encoding neurons (setting their activity to the minimal activation value, e.g. -1 if $a^{\mathrm{E}} = \tanh$). As summarized in Algorithm 1, for each input $\mathbf{x}$, SAA alternates the standard accuracy-driven weight update, aimed at the reconstruction of $\mathbf{x}$, and a sparsity-driven weight update applied to the encoding layer only, with $\mathbf{x}$ as input and $\tilde{\mathbf{y}}$ as target output (Figure 4).

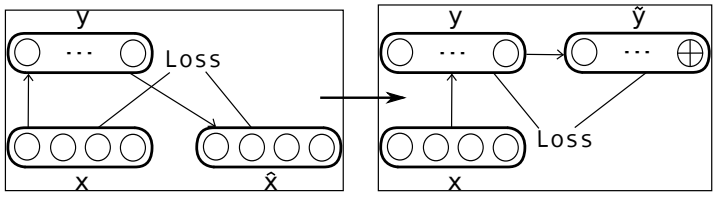

Fig. 4 The training scheme of an SAA. Code components modified by the sparsification process are marked with a $\oplus$ cross.

SAA thus iteratively and alternatively optimizes the accuracy and the sparsity of the encoder, as follows. Let us denote by $\mathbf{x^{(t)}}$ the training example fed to the AA at time t in an online learning setting and $\mathbf{y^{(t)}}$ the representation computed by the encoding layer once the AA has been trained with the previous examples $\mathbf{x^{(1)}}, \ldots, \mathbf{x^{(t-1)}}$. By alternatively applying an accuracy optimization step followed by a sparsity optimization step for each example, it is expected that the level of sparsity of $\mathbf{y^{(t)}}$ will increase with t, until code $\mathbf{y}$ consistently yields a "sufficiently sparse" approximation of $\tilde{\mathbf{y}}$. At such a point, $\mathbf{y}$ reaches a high level of sparsity while it still enables to reconstruct $\mathbf{x}$ with high accuracy, and the encoding layer achieves a good compromise between coding sparsity and accuracy.

Algorithm 1: SAA alternate optimization learning rule.

1 **for** several epochs (training examples presented in random order) **do**
2 **foreach** training example $\mathbf{x^{(t)}}$ **do**
3 **Accuracy-driven learning rule**: perform one step back-propagation for updating $\mathbf{\Theta}^{\mathrm{E}}$ and $\mathbf{\Theta}^{\mathrm{D}}$ with $\mathbf{x^{(t)}}$ as input and $\mathbf{x^{(t)}}$ as target
4 (*see Figure 4, left*);
5 Run a forward pass on the encoding layer of the AA with $\mathbf{x^{(t)}}$ as input to compute $\mathbf{y^{(t)}}$;
6 **Sparsify** $\mathbf{y^{(t)}}$ to obtain $\tilde{\mathbf{y}}^{\mathbf{(t)}}$;
7 **Sparsity-driven learning rule**: perform one step back-propagation on the encoding layer only, for updating $\mathbf{\Theta}^{\mathrm{E}}$ with $\mathbf{x^{(t)}}$ as input and $\tilde{\mathbf{y}}^{\mathbf{(t)}}$ as target (*see Figure 4, right*);

The criterion for the accuracy-driven learning rule (line 3 in Algorithm 1) is the same as the AA criterion (Equation (5)):

$$\phi^{\mathrm{S,accuracy}}(\mathbf{\Theta}) = \phi(\mathbf{\Theta}) = \sum_{\mathbf{x}\in\mathcal{S}} \mathcal{L}(\mathbf{x}, f^{\mathrm{D}} \circ f^{\mathrm{E}}(\mathbf{x})), \tag{9}$$

whereas the criterion for the sparsity-driven learning rule (line 6) only regards the encoding parameters:

$$\phi^{\mathrm{S,sparsity}}(\mathbf{\Theta}^{\mathrm{E}}) = \sum_{\mathbf{x}\in\mathcal{S}} \mathcal{L}(f^{\mathrm{E}}(\mathbf{x}), \tilde{\mathbf{y}}) \quad \text{with} \quad \tilde{\mathbf{y}} = \mathrm{sparsify} \circ f^{\mathrm{E}}(\mathbf{x}). \tag{10}$$

Several sparsification rules have been considered to map $\mathbf{y}$ onto $\tilde{\mathbf{y}}$ (line 5 of Algorithm 1). The first two rules are deterministic, parameterized from a fixed number of neurons, or a fixed activation threshold. Formally, the first one proceeds by cancelling out the η neurons with lowest activity, while the second one cancels out all neurons with activation less than threshold τ. For instance, for $\tau = 0$ and $a^{\mathrm{E}} = \tanh$, then $\tilde{y}_i = -1$ if y_i is negative and $\tilde{y}_i = y_i$ otherwise. A third sparsification rule proceeds stochastically, where the probability of canceling out an entry increases as its activation decreases, e.g. $P(\tilde{y}_i = -1) = (e^{1-y_i} - 1)/(e^2 - 1)$.

By construction, the deterministic size-driven sparsification rule results in a sparse coding with same size for all examples, which might over-constrain the sparse coding solution. Indeed, many datasets involve examples with different information content: some examples may need a large number of active neurons to be well represented while some may only need a small number. The last two sparsification rules achieve a variable-size coding.

In summary, the SAA framework involves the same hyper-parameters as the standard AA (chiefly the back-propagation learning rate), plus one hyper-parameter controlling the sparsification step (line 5 of Algorithm 1), respec-

tively η (for the size-driven sparsification rule) or τ (for the threshold-driven sparsification rule).

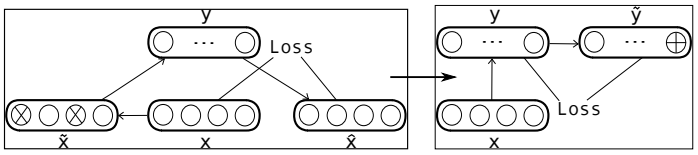

Fig. 5 The training scheme of an SDAA. Input components modified by the corruption process are marked with a $\otimes$ cross and code components modified by the sparsification process are marked with a $\oplus$ cross.

Note also that integrating the sparsification steps within a denoising auto-associator is straightforward, by replacing the AA update rule (line 3 of Algorithm 1) with the DAA update rule, thus yielding a hybrid model referred to as sparse DAA (SDAA). The training scheme of an SDAA is illustrated in Figure 5, where the accuracy and sparsity criteria are derived from Eq. (8) and Eq. (10) as follows:

$$\phi^{\text{SD,accuracy}}(\mathbf{\Theta}) = \phi^{\text{D}}(\mathbf{\Theta}) = \sum_{\mathbf{x} \in \mathcal{S}} \mathcal{L}(\mathbf{x}, f^{\text{D}} \circ f^{\text{E}}(\tilde{\mathbf{x}})), \tag{11}$$

$$\phi^{\text{SD,sparsity}}(\mathbf{\Theta}^{\text{E}}) = \phi^{\text{S,sparsity}}(\mathbf{\Theta}^{\text{E}}) = \sum_{\mathbf{x} \in \mathcal{S}} \mathcal{L}(f^{\text{E}}(\mathbf{x}), \tilde{\mathbf{y}}). \tag{12}$$

The computational complexity of all above schemes (AA, DAA, SAA and SDAA) are $\mathcal{O}(n \times m)$ per training example, with n the input dimension and m the number of encoding neurons.

5 Experiments

This section reports on the experimental validation of the sparse auto-associator (SAA) and the sparse denoising auto-associator (SDAA), comparatively to the standard and denoising auto-associators (AA and DAA). After describing the experimental setting, the section discusses the trade-off between the size of the feature space and the predictive accuracy of the classifier built on this feature space.

5.1 Experimental setting

All experiments have been carried out on three handwritten digit recognition, variants of the original MNIST dataset built by LeCun and Cortes[1]. These variants due to Larochelle et al. [20][2], differ from the original MNIST by the background image (Figure 6):

- MNIST-basic: white digits on a uniform black background;
- MNIST-bg-rand: white digits on a random grey-level background;
- MNIST-bg-img: white digits on grey-level natural images as a background.

Fig. 6 Images of a digit in class "8", taken from each dataset: MNIST-basic (left), MNIST-bg-rand (middle) and MNIST-bg-img (right).

Notably, MNIST-basic is a far more challenging learning problem than MNIST due to its reduced size (10,000 example training set; 2,000 example validation set and 50,000 example test set). Each dataset involves 10 classes, where each example is given as a (28×28) grey-level pixel image and processed as a vector in $\mathbb{R}^{784}$, by scrolling the image in a left-to-right top-to-bottom fashion and recording each visited pixel.

Let C, X^{train}, X^{val}, X^{test}, L^{train}, L^{val} and L^{test} be defined as follows:

- $C = \{1, \ldots, c\}$ is the set of class labels, with c the number of classes ($c = 10$ for all datasets);
- X^{train} is the sequence of training examples $\mathbf{x^{(k),\text{train}}} \in \mathbb{R}^n$;
- L^{train} is the sequence of the class label $l^{(k),\text{train}} \in C$ for each $\mathbf{x^{(k),\text{train}}}$;
- X^{val} is the sequence of validation examples $\mathbf{x^{(k),\text{val}}} \in \mathbb{R}^n$;
- L^{val} is the sequence of the respective class label $l^{(k),\text{val}} \in C$;
- X^{test} is the sequence of test examples $\mathbf{x^{(k),\text{test}}} \in \mathbb{R}^n$;
- L^{test} is the sequence of the respective class label $l^{(k),\text{test}} \in C$.

All AA variants will be comparatively assessed on the basis of their discriminant power and the feature space dimension.

---

[1] Original MNIST database: `http://yann.lecun.com/exdb/mnist/`.

[2] MNIST variants site: `http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/MnistVariations`.

5.2 Discriminant power

The discriminant power of an AA variant is measured as the predictive accuracy of a classifier learned from the feature space built from this AA variant, along the following procedure:

1. **Unsupervised learning:** from $(X^{\text{train}}, X^{\text{val}})$ train an AA, a DAA, a SAA and a SDAA, respectively denoted by $\mathcal{A}$, $\mathcal{A}^{\text{D}}$, $\mathcal{A}^{\text{S}}$ and $\mathcal{A}^{\text{SD}}$;
2. **Classifier building:** from $\mathcal{A}$, $\mathcal{A}^{\text{D}}$, $\mathcal{A}^{\text{S}}$ and $\mathcal{A}^{\text{SD}}$ respectively, initialize the two-layer neural network classifiers $\mathcal{N}$, $\mathcal{N}^{\text{D}}$, $\mathcal{N}^{\text{S}}$ and $\mathcal{N}^{\text{SD}}$ by removing the decoders and plugging c neurons at the output of the encoders, one for each class;
3. **Supervised fine-tuning:** from $(X^{\text{train}}, X^{\text{val}})$ and $(L^{\text{train}}, L^{\text{val}})$ train $\mathcal{N}$, $\mathcal{N}^{\text{D}}$, $\mathcal{N}^{\text{S}}$ and $\mathcal{N}^{\text{SD}}$;
4. **Performance measuring:** from X^{test} and L^{test} estimate the classification accuracy of $\mathcal{N}$, $\mathcal{N}^{\text{D}}$, $\mathcal{N}^{\text{S}}$ and $\mathcal{N}^{\text{SD}}$.

Figure 7 comparatively displays the auto-associator architecture $\mathcal{A}$ and the associated multi-layer classifier $\mathcal{N}$. With architecture $\mathcal{A}$, the AA is trained to build a feature space by reconstructing its inputs on its output neurons. With architecture $\mathcal{N}$, the feature space is further trained by back-propagation to yield the class associated to the AA inputs, using c output neurons (one for each class).

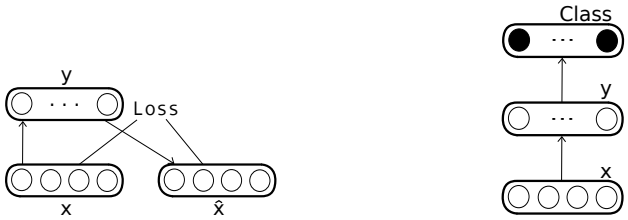

Fig. 7 From an auto-associator $\mathcal{A}$ (left) to the corresponding classifier $\mathcal{N}$ (right).

The unsupervised feature extraction training phase and the supervised classifier training phase (steps 1 and 3 of the above procedure) each involve 50 epochs of stochastic back-propagation over X^{train} and X^{val}, with the squared error as loss function. At each epoch the examples are presented in random order. The encoding dimension m of each AA variant has been set equal to the input dimension n, i.e. $m = 784$ neurons in the encoding layer.
The activation function is tanh. The bias are initialized to 0 while the weights are randomly initialized following advice in [23]. The sparsification heuristics is the thresholded one[3] (section 4) where the threshold is set to 0. In other

[3] The probabilistic sparsification heuristics has been experimented too and found to yield similar results (omitted for the sake of brevity).

words, all features with negative values are set -1 (minimum value of tanh). For inputs normalized in $[-1, 1]$, the input corruption process in $\mathcal{A}^{\mathrm{D}}$ and $\mathcal{A}^{\mathrm{SD}}$ consists in setting independently each input entry to -1 with probability ν.

The values of the back-propagation learning rate α and input corruption rate ν have been selected by grid-search, where each hyper-parameter setting is assessed from 10 runs (with different weight initialization and example ordering), learning on X^{train} and evaluating the performance on X^{val}. By consistency, both training phases (steps 1 and 3) use the same α value. The candidate hyper-parameters values are reported in Table 1.

Name	**Candidate values**	**Models**
Learning rate α	0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5	All AA variants and all classifier variants
Input corruption rate ν	0, 0.1, 0.2, 0.4	DAA and SDAA

Table 1 Hyper-parameter range tested through the grid-search procedure.

Table 2 displays the clasification error rates averaged over 10 independent runs. Note that in some cases, the best ν value is 0, resulting in identical results for AA and DAA, or SAA and SDAA. On all datasets, SAA is shown to significantly[4] improve on AA in terms of predictive accuracy. The feature space built by $\mathcal{A}^{\mathrm{S}}$ thus seems to capture discriminant features better than the standard $\mathcal{A}$. It must be emphasized that both feature spaces have same size, are trained using unsupervised learning and only differ in the sparsity step. These results suggest that the SAA approach does take practical advantage of one of the more appealing theoretical assets of sparse representations, namely their tendency to favor class separability compared to dense representations.

Dataset	**Error rate**			
	AA	**DAA**	**SAA**	**SDAA**
MNIST-basic	4.49% (0.06%)	**3.87**% (0.04%)	**3.98**% (0.09%)	**3.98**% (0.08%)
MNIST-bg-rand	22.4% (0.10%)	**19.5**% (0.09%)	**19.5**% (0.23%)	**19.5**% (0.23%)
MNIST-bg-img	25.6% (0.37%)	**23.6**% (0.15%)	**23.4**% (0.57%)	**23.0**% (0.43%)

Table 2 Mean and standard deviation of the classification error rate when the encoding dimension is set to the input dimension. Significantly best results are in boldface.

Secondly, it is observed that DAA, SAA and SDAA do not offer any significant difference regarding their predictive accuracies. A tentative interpretation for this fact goes as follows. The denoising and sparsification heuristics respectively involved in DAA and SAA aim at comparable properties (namely

[4] All statistical tests are heteroscedastic bilateral T tests. A difference is considered significant if the p-value is less than 0.001.

coding robustness), although through different ways. It seems that both ways are incompatible, in the sense that they cannot be combined effectively. This interpretation is confirmed as the best ν value selected for SDAA is close to 0 (0 or .1) whereas it is clearly higher for DAA (.2 or .4): SDAA in such different conditions almost coincides with DAA. In other words, the standard AA can hardly achieve at the same time a low reconstruction error, a good robustness to noise, and sparsity. Further investigation could be performed to search how the denoising and sparsity heuristics could be made more compatible through simultaneously adjusting the input corruption rate and the sparsification threshold.

5.3 Pruning the feature space

It comes naturally to investigate whether all features in the feature space are active, i.e. if there exists some encoding neurons which are never active during the unsupervised SAA training. Such features could be removed from the feature space at null cost in terms of reconstruction error, i.e. they would enable a pruning of the feature space − although there is no evidence naturally that sparsity would favor such a pruning.

The pruning of the feature space has been investigated through considering an additional pruning step on the top of learning the encoding layer of $\mathcal{A}^{\mathrm{S}}$ and $\mathcal{A}^{\mathrm{SD}}$. Formally, the pruning phase proceeds by removing all neurons the activity of which is consistently negative over all training examples. The approach considers the same hyper-parameter values as previously selected by grid-search on $\mathcal{A}^{\mathrm{S}}$ and $\mathcal{A}^{\mathrm{SD}}$.

Dataset	**Reduced encoding dimension**		**Error rate**	
	SAA	**SDAA**	**SAA**	**SDAA**
MNIST-basic	**719** (7.9)	757 (7.7)	**3.96**% (0.07%)	**4.00**% (0.16%)
MNIST-bg-rand	**634** (9.9)	**634** (9.9)	**19.3**% (0.29%)	**19.3**% (0.29%)
MNIST-bg-img	**248** (12.0)	761 (5.0)	**23.3**% (0.36%)	**23.0**% (0.43%)

Table 3 Mean and standard deviation of the classification error rate and encoding dimension as obtained by removing useless neurons after $\mathcal{A}^{\mathrm{S}}$ or $\mathcal{A}^{\mathrm{SD}}$ training.

The relevance of the pruning heuristics is empirically demonstrated in Table 3. While the pruning heuristics strictly reduces the original feature space dimension (set to $m = n = 784$ in all experiments), it does not hinder the predictive accuracy (comparatively to Table 2). In other words, the neurons which have been removed along the pruning step did not convey discriminant information and/or could not be efficiently recruited in the supervised learning stage.

It can be observed that the efficiency of the pruning heuristics significantly varies depending on the datasets, that is, depending on the background image. A constant background (MNIST-basic, 719 features on average) entails more active features than a random one (MNIST-bg-rand, 634 features on average) and considerably more than a non-random one (MNIST-bg-img, 248 features on average). A tentative interpretation for this result is that, the more informative the background, the larger the number of neurons trained to reconstruct the background patterns, and consequently the smaller the number of neurons trained to reconstruct the digit patterns. The variability of the background patterns, higher than those of the digit patterns, might explain why background-related neurons are more often silenced than the others. Further investigation is required to confirm or infirm this conjecture, analyzing the internal state of the $\mathcal{A}^{\mathrm{S}}$ and $\mathcal{A}^{\mathrm{SD}}$ architectures.

Interestingly, the pruning heuristics is ineffective in the SDAA case, where the feature space dimension tends to be constant. This observation both confirms that the sparsity and the denoising heuristics hardly cooperate together, as discussed in the previous subsection. It also suggests that the sparsity heuristics is more flexible to take advantage of the input data structure.

Notably, the pruning heuristics can be seen as a particular case of the *common sparsity profile* constraint [27] involved in the dictionary learning field (section 1), penalizing the use of atoms which are seldom used.

For the sake of completeness, let us investigate whether AA or DAA would stand the reduction of the feature space dimension as yielded by the pruning heuristics above. The comparative assessment proceeds by setting the size of the coding layer to the reduced $m^\star$ obtained as above. The hyper-parameters are set by grid-search, on the same candidate values (Table 1).

Dataset	**Encoding dimension**	**Error rate**	
		AA	**DAA**
MNIST-basic	719	4.53% (0.19%)	**3.80**% (0.07%)
MNIST-bg-rand	634	22.0% (0.07%)	**19.4**% (0.20%)
MNIST-bg-img	248	25.9% (0.22%)	25.9% (0.22%)

Table 4 Mean and standard deviation of the classification error rate when the encoding dimension is determined by the mean one obtained by the SAA in the previous experiment (see Table 3).

As shown in Table 4, the feature space reduction adversely affects the AA and DAA accuracy, particularly so in the case of the MNIST-bg-img dataset. The fact that AA and DAA yield same accuracy is due to the fact that $\nu = 0$. Indeed, the case of an information-rich background makes it more difficult for DAA to achieve a low reconstruction error and a high predictive accuracy, all the more so as the noise rate is high. Overall, it is suggested that the DAA strategy is more demanding than the SAA one in terms of the size of the feature space, as the former comprehensively aims at

coping with all encountered noise patterns in the example, whereas the latter follows a destructive strategy ignoring all weak patterns in the features. These antagonistic goals might further explain why the two strategies can hardly be combined.

6 Discussion

As mentioned in section 1, the main contribution of this chapter has been to present (sections 4 and 5) a sparsity-driven procedure to enforce the learning of sparse feature space in the auto-associator framework. The use of this procedure within the standard stochastic gradient AA learning procedure, referred to as sparse auto-associator, promotes codes which are both *accurate* in terms of representation, and *sparse* in terms of the low number of encoding neurons activated by an input example on average. A primary merit of SAA over the most popular sparse dictionary learning is to yield a *non-linear* code. A second merit is this code is explicit and can be computed for any new example with linear complexity (feedforward pass to compute the features, i.e. the hidden layer states), whereas it requires solving an optimization problem in the dictionary case.

After summarizing the main benefits of SAA w.r.t. its accuracy for classification purpose and its ability for sparse coding, the method will be viewed in lights of the four directions of research briefly presented in section 2.2.

6.1 Benefits of the Sparse Auto-Associator

Accuracy-wise, SAA yields similar results as denoising auto-associator on three well-studied MNIST variants, involving different digit backgrounds (constant, uniform, or image-based). Both SAA and DAA significantly improve on AA, which is explained from the regularization effect of respectively the sparsity- and denoising-driven procedures. Both procedures implicitly take advantage of the fact that the data live in a low-dimensional feature space. Uncovering this low-dimensional space enforces the description robustness.

Interestingly, combining the sparsity- and denoising-driven procedures does not help: experimentally, SDAA does neither outperform SAA nor DAA. Our tentative interpretation for this fact, the incompatibility of both procedures, is that the denoising-driven procedure aims at getting rid of weak perturbation patterns on the input layer, whereas the sparsity-driven procedure aims at getting rid of weak patterns in the encoding layer.

Coding-wise, it seems that SAA can be made more efficient than DAA as far as the size of the feature space is concerned. The pruning heuristics, removing all features which are never activated by the training examples above a given threshold, yields a significant reduction of the feature space dimension *with no accuracy loss*, albeit it was implemented with a naive activity threshold[5]. The use of this pruning heuristics enables SAA to autonomously adjust the size of the feature space, starting with a large number of features and pruning the inactive ones after the AA training, thus yielding a sparse and *compressed* representation.

Admittedly, the effectiveness of the pruning rule depends on the dataset: experimentally, it is best when the variability of the input patterns is neither too high (random background) nor too low (constant background). Interestingly, the most realistic cases (image background) are the most favorable cases, since SAA enables a 3-fold reduction of the feature space size, actually uncovering the common sparsity profile of the examples.

A main merit of the pruning heuristics is to yield an appropriate feature space dimension for a given dataset. A second one, empirically demonstrated, is that it does preserve the discriminant power of the feature space. Note however that directly considering the same feature space dimension with AA or DAA significantly degrades the predictive accuracy.

6.2 Comparison with related work

Let us discuss the strengths and weaknesses of the SAA framework comparatively to the sparse coding neural-based methods, presented in section 2.2. It must first be emphasized that all sparse features learning methods have a quite similar computational cost that hardly scales up for *big data*, albeit recent achievements show that impressive results can be obtained when increasing the size of the dataset by several orders of magnitude [8].

On the positive side, SAA does not require any prior knowledge about the problem domain and the target optimal code, as opposed to [11] and [12]. SAA actually uncovering the common sparsity profile of the examples, although it does not explicitly consider any selectivity property, as opposed to [11]. It accommodates online learning, also as opposed to [11], through a simple stochastic gradient approach, which does not require any indicators about the neurons to be maintained, as opposed to the average past activity used in [24] and [14]. Importantly, the straightforward sparsity-driven procedure makes SAA easy to understand and implement, without requiring any particular trick to make the standard back-propagation procedure to work, as opposed to [10].

[5] Complementary experiments, varying the pruning threshold in a range around 0, yield same performance (results omitted for brevity).

On the negative side, SAA needs to be given theoretical foundations, or could be related to the theory of biological neuron computation as e.g. [11] or [10]. In particular, further work will investigate how the sparsity-driven procedure can be analyzed in terms of ill-posed optimization resolution.

7 Conclusion and perspectives

Focused on sparse and low-dimensional feature coding, this chapter has presented the new sparse auto-associator framework. The main motivation for this framework is rooted in Information Theory, establishing the robustness of sparse code w.r.t. transmission noise. In the Machine Learning field, sparse coding further facilitates the separability of examples, which made sparse coding a hot topic for the last decade [6,9,27]. Such good properties of sparse representations have been experimentally confirmed in terms of predictive accuracy compared to standard auto-associator.

SAA should be viewed as an alternative to sparse dictionary learning [26], in several respects. On the one hand, they provide a *non-linear* feature space, more easily able to capture complex data structures than linear coding. On the other hand, while both SAA and dictionary-based codings are rather costly to be learned, the SAA coding is explicit and computationally cheap in generalisation phase: the coding of a further example is computable by feed-forward propagation, whereas it results from solving an optimization problem in the dictionary framework.

The SAA approach offers several perspectives for further research. A primary perspective, inspired from the deep learning field [3, 19], is to stack SAA in a layer-wise manner, expectedly yielding gradually more complex and abstract non-linear features. Preliminary experiments show that the direct stacking of an SAA however is ineffective as the reconstruction of an already sparse coding derives a degenerated optimization problem. Further work will be concerned with controlling and gradually adapting the sparsity level along the consecutive layers, using an appropriate sparsity criterion and schedule. Another possibility is to alternate SAA layers, with subsequent pruning, and AA layers, without sparse coding, in layer-wise building a deep architecture, thus taking inspiration from the alternative stacking of convolutional and pooling layers promoted by LeCun et al. [18,21].

Another promising perspective is to see SAA in the light of the *dropout* procedure recently proposed by Hinton et al. [16], where some features are randomly omitted during the back-propagation step. The dropout heuristics assuredly resulted in dramatic performance improvements on hard supervised learning problems, which was attributed to the fact that this random perturbation breaks the spurious coalitions of features, each covering for the errors of others. Indeed the sparsity-driven procedure, especially in

its stochastic form (section 4) can be viewed as a dropout rule biased to low-activity neurons. Extensive further experiments are required to see how the sparsity-driven procedure can be best combined with the many other ingredients involved in the dropout-based deep neural network architecture.

Acknowledgements

This work was supported by ANR (the French National Research Agency) as part of the ASAP project under grant ANR_09_EMER_001_04.

References

1. Aharon, M., Elad, M., Bruckstein, A.: K-SVD: an algorithm for designing overcomplete dictionaries for sparse representation. IEEE Transactions on Signal Processing **54**, 4311–4322 (2006)
2. Baldi, P., Hornik, K.: Neural networks and principal component analysis: learning from examples without local minima. Neural Networks **2**, 53–58 (1989)
3. Bengio, Y.: Learning deep architectures for AI. Foundations and Trends in Machine Learning **2**, 1–127 (2009)
4. Bengio, Y., Lamblin, P., Popovici, V., Larochelle, H.: Greedy layer-wise training of deep networks. In: Neural Information Processing Systems (NIPS), pp. 1–8 (2007)
5. Bourlard, H., Kamp, Y.: Auto-association by multilayer perceptrons and singular value decomposition. Biological Cybernetics **59**, 291–294 (1988)
6. Candès, E.J.: The restricted isometry property and its implications for compressed sensing. Comptes Rendus de l'Académie des Sciences **346**, 589–592 (2008)
7. Chen, S.S., Donoho, D.L., Saunders, M.A.: Atomic decomposition by basis pursuit. SIAM Journal on Scientific Computing **20**, 33–61 (1998)
8. Coates, A., Karpathy A. ans Ng, A.Y.: Emergence of object-selective features in unsupervised feature learning. In: Neural Information Processing Systems (NIPS) (2012)
9. Donoho, D.L., Elad, M.: Optimally sparse representation in general (nonorthogonal) dictionaries via ℓ^1 minimization. Proceedings of the National Academy of Sciences of the United States of America **100**, 2197–2202 (2003)
10. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: International Conference on Artificial Intelligence and Statistics (AISTATS), pp. 315–323 (2011)
11. Goh, H., Thome, N., Cord, M.: Biasing restricted Boltzmann machines to manipulate latent selectivity and sparsity. In: Neural Information Processing Systems (NIPS): Workshop on Deep Learning and Unsupervised Feature Learning, pp. 1–8 (2010)
12. Gregor, K., LeCun, Y.: Learning fast approximations of sparse coding. In: International Conference on Machine Learning (ICML), pp. 399–406 (2010)
13. Hinton, G.E.: Connectionist learning procedures. Artificial Intelligence **40**, 185–234 (1989)
14. Hinton, G.E.: A practical guide to training restricted Boltzmann machines. Tech. Rep. UTML TR 2010–003, University of Toronto (2010)

15. Hinton, G.E., Osindero, S., Teh, Y.W.: A fast learning algorithm for deep belief nets. Neural Computation **18**, 1527–1554 (2006)
16. Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.R.: Improving neural networks by preventing co-adaptation of feature detectors. In: Neural Information Processing Systems (NIPS) (2012). ArXiv:1207.0580v1 [cs.NE] 3 Jul 2012
17. Japkowicz, N., Hanson, S.J., Gluck, M.A.: Nonlinear autoassociation is not equivalent to PCA. Neural Computation **12**, 531–545 (2000)
18. Kavukcuoglu, K., Ranzato, M.A., LeCun, Y.: Fast inference in sparse coding algorithms with applications to object recognition. In: Neural Information Processing Systems (NIPS): Workshop on Optimization for Machine Learning (2008). ArXiv:1010.3467v1 [cs.CV] 18 Oct 2010
19. Larochelle, H., Bengio, Y., Louradour, J., Lamblin, P.: Exploring strategies for training deep neural networks. Journal of Machine Learning Research **10**, 1–40 (2009)
20. Larochelle, H., Erhan, D., Courville, A., Bergstra, J., Bengio, Y.: An empirical evaluation of deep architectures on problems with many factors of variation. In: International Conference on Machine Learning (ICML), pp. 473–480 (2007)
21. LeCun, Y.: Learning invariant feature hierarchies. In: European Conference in Computer Vision (ECCV), *Lecture Notes in Computer Science*, vol. 7583, pp. 496–505. Springer-Verlag (2012)
22. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
23. LeCun, Y., Bottou, L., Orr, G.B., Müller, K.R.: Efficient backprop. In: Neural Networks: Tricks of the Trade, pp. 9–50 (1998)
24. Lee, H., Ekanadham, C., Ng, A.Y.: Sparse deep belief net model for visual area V2. In: Neural Information Processing Systems (NIPS), pp. 873–880 (2007)
25. Lee, H., Grosse, R., Ranganath, R., Ng, A.Y.: Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In: International Conference on Machine Learning (ICML), p. 77 (2009)
26. Mairal, J., Bach, F., Ponce, J., Sapiro, G.: Online dictionary learning for sparse coding. In: International Conference on Machine Learning (ICML), pp. 689–696 (2009)
27. Rakotomamonjy, A.: Surveying and comparing simultaneous sparse approximation (or group-LASSO) algorithms. Signal Processing **91**, 1505–1526 (2011)
28. Ranzato, M., Huang, F.J., Boureau, Y.L., LeCun, Y.: Unsupervised learning of invariant feature hierarchies with applications to object recognition. In: Computer Vision and Pattern Recognition (CVPR), pp. 1–8 (2007)
29. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Nature **323**, 533–536 (1986)
30. Tropp, J.A.: Greed is good: algorithmic results for sparse approximation. IEEE Transactions on Information Theory **50**, 2231–2242 (2004)
31. Utgoff, P.E., Stracuzzi, D.J.: Many-layered learning. Neural Computation **14**, 2497–2539 (2002)
32. Vapnik, V.N.: Statistical Learning Theory. Wiley (1998)
33. Vincent, P., Larochelle, H., Bengio, Y., Manzagol, P.A.: Extracting and composing robust features with denoising autoencoders. In: International Conference on Machine Learning (ICML), pp. 1096–1103 (2008)