"Learning sparse features with an auto-associator"

Sébastien Rebecchi, Hélène Paugam-Moisy and Michèle Sebag (CNRS, LRI, INRIA Saclay - Île-de-France)

Abstract—This paper aims at building a non-linear explicit sparse coding by optimizing the encoding layer of an auto-associator.

The proposed approach, called sparse auto-associator (SAA), introduces a sparsity constraint within standard auto-associator learning. Formally, SAA alternates the auto-associator update rule, minimizing the reconstruction error, and an original non-linear sparsification update rule, restricting the number of neurons that intervene in the reconstruction of the current example.

Experiments on standard handwritten digit recognition datasets show that the method is more competitive than standard auto-associators in terms of classification performance. Moreover, compared to denoising auto-associators, the present approach yields similar supervised learning performance, with the additional advantage of giving a simple way to prune the encoding layer, thus reducing the dimensionality of the feature space.
Learning sparse features with an auto-associator

Sébastien Rebecchi*, Hélène Paugam-Moisy*‡ and Michèle Sebag*

* CNRS, LRI UMR 8623
TAO, INRIA Saclay - Ile-de-France
Université Paris-Sud 11, F-91405 Orsay
Email: sebastien.rebecchi@lri.fr

‡ CNRS, LIRIS UMR 5205
Université Lumièrè Lyon 2, F-69676 Bron

Abstract—This paper aims at building a non-linear explicit sparse coding by optimizing the encoding layer of an auto-associator.

The proposed approach, called sparse auto-associator (SAA), introduces a sparsity constraint within standard auto-associator learning. Formally, SAA alternates the auto-associator update rule, minimizing the reconstruction error, and an original non-linear sparsification update rule, restricting the number of neurons that intervene in the reconstruction of the current example.

Experiments on standard handwritten digit recognition datasets show that the method is more competitive than standard auto-associators in terms of classification performance. Moreover, compared to denoising auto-associators, the present approach yields similar supervised learning performance, with the additional advantage of giving a simple way to prune the encoding layer, thus reducing the dimensionality of the feature space.

I. INTRODUCTION

Sparse coding, a.k.a. sparse approximation, consists in computing new representations \( y \in \mathbb{R}^m \) of examples \( x \in \mathbb{R}^n \) under two constraints, namely accuracy and sparsity (i.e. low density). The accuracy property imposes that \( x \), or a good approximation thereof, can be reconstructed from \( y \). The sparsity property requires that \( y \) is composed of a small number of active entries. An active component plays an important role in \( y \), in the sense that it can not be ignored without significantly altering reconstruction accuracy.

Sparse coding has been intensively studied in the last decade, in relation with compressed sensing [1], signal decomposition [2] and dictionary learning [3]. This research field, originating from the signal processing community, aims at finding a linear decomposition of \( x \) using a small number of atoms of a given dictionary \( D \). Formally, \( D \) is a \((n \times m)\) real matrix, its \( m \) columns vectors \( D_{sj} \) denoting the atoms of the dictionary. The sparse coding \( y \) is obtained by solving a combinatorial optimization problem, minimizing its \( \ell_0 \) norm \( \|y\|_0 \) subject to the reconstruction constraint:

\[
\min_{y \in \mathbb{R}^m} \|y\|_0 \quad \text{subject to} \quad x = \sum_{j=1}^m (D_{sj}y_j),
\]

where \( y_j \) denotes the \( j \)-th entry of \( y \). Equation 1 can be solved approximately by the greedy orthogonal matching pursuit (OMP) algorithm [4], subject to upper bounding \( \|y\|_0 \).

A convex relaxation of Equation 1 is obtained by replacing \( \ell_0 \) norm by \( \ell_1 \) norm and achieving some compromise between the accuracy and sparsity criteria:

\[
\min_{y \in \mathbb{R}^m} \left( \left\| x - \sum_{j=1}^m (D_{sj}y_j) \right\|_2^2 + \lambda \|y\|_1 \right), \quad (2)
\]

where \( \lambda > 0 \) is the trade-off parameter, \( \|\cdot\|_1 \) and \( \|\cdot\|_2 \) denote the \( \ell_1 \) and \( \ell_2 \) norm, respectively. The alternative problem of Equation 2 is known as basis pursuit (BP) denoising [2]. There exists conditions [5] under which a sufficiently \( \ell_0 \)-sparse solution of Equation 1 is also the unique \( \ell_1 \)-minimizing solution to Equation 2.

In both problems the objective is to find an approximate decomposition of \( x \) which activates a small number of atoms of \( D \), where the level of activity of \( D_{sj} \) is evaluated by \( |y_j| \). Indeed, canceling the entry \( j \), i.e. ignoring \( D_{sj} \) in the reconstruction process, consists in setting \( y_j \) to the specific value 0. This is at the cost of an additional reconstruction loss, the level of which increases with \( |y_j| \). Canceling entries corresponding to very weakly active atoms implies a very low decrease of accuracy.

However a given dictionary may not provide every example with a sparse decomposition. Hence it becomes natural to adapt the dictionary to the specificities of the examples at hand [3], [6]. If sparsity is measured with the \( \ell_1 \) norm then dictionary learning for sparse coding is reduced to solving the following joint optimization problem:

\[
\min_{D \in \mathbb{R}^{n \times m}, Y \in \mathbb{R}^{m \times l}} \left( \|X - DY\|_2^2 + \lambda \sum_{k=1}^l \|Y_{sk}\|_1 \right), \quad (3)
\]

with \( l \) the number of training examples, \( X \) the \((n \times l)\) real matrix storing the training examples in columns and \( Y \) the \((m \times l)\) real matrix storing the sparse representation of the \( k \)-th training example \( X_{sk} \) in its \( k \)-th column \( Y_{sk} \). However, a straightforward limitation of dictionary learning is that it remains necessary to solve a computationally expensive BP optimization problem (Equation 2) for each unseen pattern \( x \) to obtain the associated code \( y \).
The benefits of sparse coding are manifold [7]–[9]. Firstly, sparse representations are robust to noise. Secondly, as they favor example (and thus class) separability, they facilitate supervised learning and classification. Along the same lines, they are easier to interpret by disentangling the factors of variations mixed up in dense representations. Indeed, the minimization-based formulation in Equation 3 yields a variable-size representation, since some examples may need very few dictionary atoms (i.e. active components) to be accurately reconstructed while more informative examples may require a large number of active components.

A first limitation of the above dictionary-based approach is to define an implicit and computationally expensive sparse coding. A second limitation is to be restricted to linear coding. The approach proposed in this paper addresses both limitations and presents a non-linear and explicit sparse coding, based on an artificial neural network architecture, the auto-associator (AA) [10], [11]. The AA learning process aims at satisfying the only accuracy criterion. The sparse auto-associator (SAA) extends the AA in order to also comply with the sparsity criterion. The main benefit of SAA is to yield a non-linear coding, directly computable for unseen patterns without solving any additional optimization problem. Another benefit of neural network-based architectures is to be suited to iterative and hierarchical construction processes in order to form a deep hierarchy of nonlinear features [12], [13].

This paper is organized as follows. For the sake of self-containedness, the standard AA model and its best known variant, the denoising AA [14], are respectively introduced in sections II and III. Section IV describes the SAA algorithm, an alternate non-linear optimization process accommodating both accuracy and sparsity criteria. SAA is experimentally validated in section V and discussed w.r.t. the state of the art in section VI. Section VII concludes and presents some perspectives for further research.

II. THE AUTOASSOCIATOR
The AA model, a.k.a. auto-encoder, is a two-layer neural network trained unsupervisedly for feature extraction [10], [11]. The first layer is the encoding layer or encoder and the second layer is the decoding layer or decoder. Given an input example \( x \), the goal of an AA is to compute a code \( y \) from which \( x \) can be recovered with high accuracy, that is to model a two-stage approximation to the identity function:

\[
y = f^E(x) = a^E(W^Ex + b^E),
\]

\[
\hat{x} = f^D(y) = a^D(W^Dy + b^D),
\]

\[
\hat{x} \simeq x,
\]

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 \( W^E, b^E \) and \( a^E \) for the encoder and \( W^D, b^D \) and \( a^D \) for the decoder. Since the targets are equal to the inputs, the decoder’s output dimensionality (number of neurons) equals the encoder’s input dimensionality.

An AA is usually trained by gradient descent on a loss function, applying standard back-propagation of error derivatives [15] with \( x \) as target. Depending on the nature of the input data, the loss function can either be the squared error (SE) for continuous values or the cross-entropy (CE) for bit vectors or vectors of values interpreted as bit probabilities:

\[
SE(x, \hat{x}) = \sum_{i=1}^{n} (\hat{x}_i - x_i)^2,
\]

\[
CE(x, \hat{x}) = \sum_{i=1}^{n} [x_i \log(\hat{x}_i) + (1 - x_i) \log(1 - \hat{x}_i)].
\]

The training scheme of an AA is illustrated in Figure 1.

With a linear encoding activation function \( a^E \), an AA is very similar to PCA [16]. Since \( a^E \) is usually chosen as a non-linear function (e.g. sigmoid), an AA can learn more complex representations and is able to capture multimodal aspects of the input distribution [17].

III. THE DEINOISING AUTOASSOCIATOR
In addition to extracting a non-linear feature space, an AA can be trained to remove its input noise. This variant is called

\[
E(x, \hat{x}) = \sum_{i=1}^{n} (\hat{x}_i - x_i)^2 + \lambda \sum_{i=1}^{n} \log(\hat{x}_i).
\]
the denoising AA (DAA) [14].

To achieve the denoising goal, the training example fed to the network is no longer \( x \) but a corrupted version \( \tilde{x} \) of \( x \). There exist many ways of corrupting an input vector, a simple one being to cancel (i.e. set to 0) some of its components, randomly selected. Then the denoising step adds an extra hyperparameter to the standard AA, namely the input corruption rate \( \rho^D \). The training scheme of a DAA is illustrated in Figure 2.

Compared to standard AA, the DAA learning rule forces the encoding layer to learn more robust features of the inputs. These features lead to better classification performance when fed as input to a classifier. DAA learning can be related to manifold learning, with the encoding layer trying to move representations of corrupted versions of inputs closer to the manifold of the representations of clean inputs. Another underlying motivation is to learn a generative model [14].

**IV. THE SPARSE AUTOASSOCIATOR**

An SAA is a standard AA model that has been modified in order to encourage the encoding layer to learn sparse representations.

In numerical analysis and signal processing, a sparse vector is traditionally defined as a vector composed of a small number of non-zero entries. In the following, a sparse representation computed by the encoder of an AA is a vector composed of a small number of entries associated to active neurons. By *active*, we mean a neuron whose activation value is sufficiently above the minimum of the activation function. For instance, if \( a^E = \tanh \) then encoding neuron \( n_i \) is considered as totally active for \( x \) when \( y_i = 1 \) and totally inactive when \( y_i = -1 \).

However, it would be unrealistic to expect that each training example \( x \) would be given with its optimal sparse representation \( \hat{y} \) and that we can thus take advantage of this supervision to directly optimize the encoder. We rather assume that once the AA has been optimized on some input \( x \) in the standard manner, it has learned a code \( y \) that well approximates \( \hat{y} \). That means the code \( \hat{y} \) could be obtained by sparsifying \( y \), i.e. by totally deactivating the least active encoding neurons (setting their activity to the minimal activation value, e.g. \(-1\) if \( a^E = \tanh \)). Therefore, the standard AA update rule on input \( x \) will be followed by a second update rule applied to the encoding layer, with \( x \) as input and \( \hat{y} \) as target.

The assumption above will hold even more as the AA will keep on learning. Let us denote by \( x^{(t)} \) the training example fed to the AA at time \( t \) in an online learning setting and \( y^{(t)} \) the representation computed by the encoding layer once the AA has been trained with the previous examples \( x^{(1)}, \ldots, x^{(t)} \). 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 \( y^{(t)} \) will increase with \( t \), until \( y \) converges to a reliable proposition for \( \hat{y} \). Hence, \( y \) will reach a high level of sparsity that does not prevent the AA from reconstructing \( x \) with high accuracy. As a result, the encoding layer will have achieved a good compromise between sparsity and accuracy.

The learning rule of an SAA is summarized by Algorithm 1. Sparsifying \( y \) in order to obtain \( \hat{y} \) (line 5 of the algorithm) can be done by following several strategies. A first one consists in deactivating a fixed number \( \eta^S \) of neurons. A second one consists in deactivating all neurons of which the activation is not higher than a threshold \( \theta^S \). For instance, if \( a^E = \tanh \) and thresholding to \( \theta^S = 0 \) is employed, then \( \hat{y}_i = -1 \) if \( y_i \) is negative and \( \hat{y}_i = y_i \) otherwise. Even if the first strategy can be useful to get representations that have more or less the same “size”, it seems to be generally not well suited. Indeed, some datasets can be composed of very diverse examples in terms of information content and some examples may need a large number of active neurons to be well represented while some may only need a small number. At least with the second strategy one does not lose the potential advantage of variable-size representations.

The hyperparameters of SAA are the usual ones for standard AA learning rule (e.g. the back-propagation learning rate \( \alpha^B \) plus only one additional coefficient (either \( \eta^S \) or \( \theta^S \)) required for the sparsification (line 5 of Algorithm 1). The two
A. Datasets and notations

The feature space dimensionality, without significant accuracy as input to a classifier. Compared to DAA, a by-product of AA, by comparing SAA to both the standard AA and its DAA of adding a sparsity constraint in the learning algorithm of an alternate updates of sparse learning in an SAA are illustrated in Figure 3.

By simply replacing the standard AA update rule (line 3 of Algorithm 1) with the DAA update rule, we obtain the sparse DAA (SDAA) model. The training scheme of an SDAA is illustrated in Figure 4.

Notice that algorithmic complexities of AA, DAA, SAA and SDAA learning rules are all equal to $O(n \times m)$ per training example, with $n$ the input dimensionality and $m$ the number of encoding neurons.

V. EXPERIMENTS

This section presents experiments highlighting the benefit of adding a sparsity constraint in the learning algorithm of an AA, by comparing SAA to both the standard AA and its DAA variant. Compared to AA, the sparse features learned by SAA appear to be more informative about the nature of the data since they improve the discriminative performance when fed as input to a classifier. Compared to DAA, a by-product of SAA is to remove useless encoding neurons, hence to reduce the feature space dimensionality, without significant accuracy change.

A. Datasets and notations

All experiments have been carried out on three handwritten digit recognition datasets that are variations of the original MNIST standard dataset, built by LeCun and Cortes\(^1\). The datasets were made available by Larochelle et al.\(^2\):

- MNIST-basic: the digits are written in white on an uniform black background;
- MNIST-back-random: the digits are written in white on a random grey-level background;
- MNIST-back-image: the digits are written in white on a background of grey-level natural image.

The three datasets are divided in 10 classes, from class “0” to class “9”. They are composed of a training set of 10000 examples, a validation set of 2000 examples and a test set of 50000 examples. Each example is given as a $(28 \times 28)$ grey-level pixels 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.

It is worth mentioning that MNIST-basic is more challenging than MNIST for learning tasks, precisely because it is composed of far less training/validation examples. Figure 5 shows an example of an image of digit in each considered dataset.

Let $C, X^{\text{train}}, X^{\text{val}}, X^{\text{test}}, L^{\text{train}}, L^{\text{val}}$ and $L^{\text{test}}$ be defined likewise:

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

B. Discriminative accuracy of sparsified coding

In order to estimate and compare the benefit for classification purposes of the features learned by the different variants of the AA presented in the previous sections, we have conducted the following protocol for each dataset:

1) from $(X^{\text{train}}, X^{\text{val}})$ train a standard AA, a denoising variant DAA, a sparse model SAA and an hybrid sparse denoising variant SDAA, respectively denoted by $A$ or $A^D$ or $A^S$ or $A^{SD}$;
2) from $A, A^D, A^S$ and $A^{SD}$ respectively, initialize the two-layer neural networks $N, N^D, N^S$ and $N^{SD}$ by plugging $c$ neurons as output layer, instead of their decoder;
3) from $(X^{\text{train}}, L^{\text{train}}, X^{\text{val}}, L^{\text{val}})$ train $N, N^D, N^S$ and $N^{SD}$.

---

\(^1\)MNIST can be downloaded at http://yann.lecun.com/exdb/mnist/
\(^2\)MNIST variations can be downloaded at http://www.iro.umontreal.ca/~lisa/utwiki/bin/view.cgi/Public/MnistVariations.
4) from $(X^{\text{test}}, L^{\text{test}})$ estimate the classification accuracy of $N, N^D, N^S$ and $N^{SD}$.

The unsupervised feature extraction training phase and the supervised classifier training phase (steps 1 and 3 of the protocol above) consist each of 50 iterations (epochs) over $X^{\text{train}}$ and $X^{\text{val}}$ of the stochastic version of the back-propagation procedure, with the squared error as loss function. At each epoch the dataset is presented in random order. The encoding dimensionalities of the AA variants, i.e. the number of neurons in their encoding layer, were set to 784, the input dimensionality. In all neuron layers, the activation function is $\tanh$ and the bias were initialized to 0 before training while the weights were randomly initialized following advice in [19]. The code sparsification strategy in $A^S$ and $A^{SD}$ consists in thresholding to $\theta^S = 0$, i.e. setting negative activation values to $-1$, the minimum of $\tanh$. The input corruption process in $A^D$ and $A^{SD}$ consists in randomly setting a rate of the input entries to $-1$, after having normalized these inputs in the range of $\tanh$. The experiment was performed 10 times on each dataset, corresponding to 10 different configurations of the random variables (random order of the training set and weight initializations).

The values of the relevant hyperparameters $\alpha^B$ (back-propagation learning rate) and $\rho^D$ (input corruption rate) have each been selected from a set of candidates according to their performance evaluated on the validation set. This validation pre-phase consists in repeating the above protocol 10 times for each hyperparameter configuration and respectively memorizing the configuration that yields the best mean classification accuracy for each AA variant. The only differences between the above protocol and the one used for each validation run reside in the facts that the different AA and classifier variants are trained only from $X^{\text{train}}$ (steps 1 and 3) and that the classifiers accuracies are evaluated from $X^{\text{val}}$ (step 4). Hyperparameters selected by validation together with their different candidate values and involved architectures are presented in Table I. Notice that the learning rates $\alpha^B$ have been forced to be equal for the two training phases (steps 1 and 3). Identical results between AA and DAA, or between SAA and SDAA, may be observed in some cases. They are due to the fact that the input corruption rate $\rho^D$ selected by validation is 0. In such cases $A = A^D$ and $A^S = A^{SD}$.

Results are presented in Table II. The SAA algorithm for adding a sparsity constraint in the learned features leads to a good improvement in classification performance in comparison to the standard AA, without modifying the encoding dimensionality. This improvement is statistically significant for the three datasets. $A^{SD}$ has captured more discriminative features of the data than $A$, hence has allowed the gradient descent procedure of the classifier learning phase to converge toward a more satisfying point for the loss function in the parameter space. These results confirm that the SAA approach well and truly takes practical advantage of one of the more appealing theoretical assets of sparse representations, namely their tendency to favor class separability compared to dense representations. The performance is not only due to the input reconstruction objective but also to the code density reduction objective and to the specific strategy chosen to reach it.

On the other hand, there is no pairwise statistical difference between accuracies obtained by $A^D$, $A^S$ and $A^{SD}$. This observation can be due to the fact that the DAA and SAA algorithms have some of their objectives in common and are hardly compatible, maybe especially with the type of sparsification we used. Indeed, sparse coding theoretically enhance robustness to noise and this is exactly the goal of the DAA. The lack of cooperation between the two methods is confirmed by the values of the input corruption rate $\rho^D$ selected by validation. Indeed this hyperparameter is always low for $A^{SD}$ (0 or 0.1) while it is always high for $A^D$ (0.2 or 0.4). $A^{SD}$ is always very close to $A^S$ and we can hardly take advantage of the DAA learning rule with an AA variant that has already to balance between accuracy and low density. One way to enhance compatibility of the denoising and sparsity objectives could be to adapt not only the input corruption rate but also the strength of the sparsification process.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|}
\hline
Hyperparameter & Candidate values & Involved architectures \\
\hline
Learning rate & $10^{-4}$, $5 \times 10^{-4}$, $10^{-3}$, $5 \times 10^{-3}$, $10^{-2}$, $10^{-1}$ & All AA and classifier variants \\
Input corruption rate & 0, 0.1, 0.2, 0.4 & DAA and SDAA \\
\hline
\end{tabular}
\caption{Hyperparameters selected by validation.}
\end{table}

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|}
\hline
Dataset & AA & DAA & SAA & SDAA \\
\hline
MNIST-basic & 4.49\% (0.06\%) & 3.87\% (0.04\%) & 3.98\% (0.09\%) & 3.98\% (0.08\%) \\
MNIST-back-random & 22.4\% (10.1\%) & 19.5\% (0.9\%) & 19.5\% (0.23\%) & 19.5\% (0.23\%) \\
MNIST-back-image & 25.0\% (0.37\%) & 23.6\% (0.15\%) & 23.4\% (0.57\%) & 23.0\% (0.43\%) \\
\hline
\end{tabular}
\caption{Mean and standard deviation of the classification error rate when the encoding dimensionality is set to the input dimensionality. Best results are in boldface.}
\end{table}

\footnote{All statistical tests presented in this paper are heteroscedastic bilateral T tests. A difference is considered significant if the p-value is strictly lower than 0.1\%.}
Table III

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Reduced encoding dimensionality</th>
<th>Error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SAA</td>
<td>SDAA</td>
</tr>
<tr>
<td>MNIST-basic</td>
<td>719 (7.9)</td>
<td>757 (7.7)</td>
</tr>
<tr>
<td>MNIST-back-random</td>
<td>634 (9.9)</td>
<td>634 (9.9)</td>
</tr>
<tr>
<td>MNIST-back-image</td>
<td>248 (12.0)</td>
<td>248 (12.0)</td>
</tr>
</tbody>
</table>

**Mean and standard deviation of the classification error rate and encoding dimensionality when the encoding dimensionality is determined by removing useless neurons after training. Best results are in boldface.**

Table IV

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Encoding dimensionality</th>
<th>Error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AA</td>
<td>DAA</td>
</tr>
<tr>
<td>MNIST-basic</td>
<td>719</td>
<td>4.53% (0.19%)</td>
</tr>
<tr>
<td>MNIST-back-random</td>
<td>634</td>
<td>22.0% (0.07%)</td>
</tr>
<tr>
<td>MNIST-back-image</td>
<td>248</td>
<td>25.9% (0.22%)</td>
</tr>
</tbody>
</table>

**Mean and standard deviation of the classification error rate when the encoding dimensionality is determined by the mean one obtained by the SAA in the previous experiment (see Table III). Best results are in boldface.**

C. Reducing the feature space dimensionality

A by-product of the SAA learning algorithm could also be a certain selectivity in the code, i.e. the fact that some encoding neurons would never be considered as active after training $A^S$. If so, these neurons could then be totally deactivated and removed from the architecture without much more accuracy loss. In theory, there is no evidence for sparsity implying such a kind of selectivity, but it might be the case in practice.

To answer this question the previous experiment has been performed again, but with an additional step for pruning the encoding layer of $A^S$ and $A^{SD}$ after training. The pruning phase consists in removing neurons that are considered as useless because they are never active for any training example, in the sense that their activation value is never positive.

The hyperparameters previously selected by validation on $A^S$ and $A^{SD}$, without subsequent pruning, were kept unchanged for this experiment.

Results are presented in Table III. The learned codes are compressed as their dimensionality is strictly lower than the input dimensionality, which shows the effectiveness of the pruning post-stage. Moreover, the differences between these results and the corresponding ones in Table II are not statistically significant, which proves the relevance of the pruning stage. There remains a clear advantage of adding a sparsity constraint to the standard AA even with compressed representations. However, we can deduce by looking at the resulting encoding dimensionalities that this stage is as effective with the SDAA learning rule as with the SAA learning rule only when the denoising objective is disregarded (i.e. when the SDAA is an SAA, as obtained with MNIST-back-random). We can observe that the more informative the background on which the handwritten digits are drawn, the smaller the size of the resulting SAA code. We suspect that this evolution is due to the fact that the more informative the background, the larger the number of neurons trained to detect patterns mostly present in the background, hence the smaller the number of neurons trained to detect patterns mostly present in the digits. Indeed, we think that the removed neurons could be in majority the ones that are strongly influenced to be inactive when some patterns are detected in the digits, in comparison to the influence they undergo to be active because of the presence of some patterns in the background. But this conjecture requires a more in-depth study of the internal state of the trained architecture. A similar evolution is not observed with the SDAA for which the code dimensionality tends to remain constant, which confirms the lack of cooperation between the SAA and DAA learning methods. This observation points out an additional advantage of using the SAA learning rule rather than the DAA one.

To conclude with this experiment, we can relate the fact that some neurons are never considered active after training with the concept of learning simultaneous sparse coding (SSC). SSC consists in computing representation of a set $X$ of examples with, on the one hand, the same constraints of accuracy and low density as in traditional sparse coding, but also with, on the other hand, a third constraint stated as common sparsity profile of the new representations of the examples in $X$. Within the dictionary paradigm developed in the introduction, this additional constraint is expressed as the fact that the number of dictionary atoms active for at least one sparse example representation must be minimized. Then the atoms that are not active for any example can be considered as useless and thus be removed from the dictionary. In our context, having been able to prune the encoding layer of the SAA after training means that this layer also detected common sparsity profile of its training examples. For more details about SSC in the context of dictionaries see [20].

D. Comparing low dimensionality feature spaces

A last experiment has been done in order to confirm the attractiveness of SAA. In order to state that $N$ and $N^D$ would also be robust to code dimensionality reduction, we wonder if limiting the number of encoding features of $A$ and $A^D$ to the small values obtained by $A^S$ after pruning would lead to still comparable classification results.

The relevant hyperparameters of this experiment were se-
lected by validation, as in the first experiment (section V-B). The only difference is that the encoding dimensionality of \( A \) and \( A^D \) is not set to the input dimensionality but restricted to the mean encoding dimensionality computed by \( A^S \) after pruning.

Results are presented in Table IV. The performance of the DAA obtained on MNIST-back-image shows a clear argument for using an SAA in comparison to a DAA for extracting low dimensional representations. On this dataset, the DAA obtains the same result than the standard AA because the input corruption rate \( \rho^D \) selected by validation has been 0. The error rate increases between the DAA result obtained in the first experiment and its results obtained in the present one is statistically significant. It seems that the DAA needs a large number of feature extractors (encoding neurons) to be able to reach its denoising goal without accuracy loss. A tentative interpretation the fact that much more encoding neurons are required to take into account all possibly encountered corruptions of the input. The DAA follows a strategy of exhaustiveness, while the SAA has a destructive strategy. Instead of memorizing all possible corruptions of the input in the code, since it sparsifies its code, the SAA ignores irrelevant patterns of its input detected by encoding neurons. This opposition between exhaustiveness and sparsity could be an explanation for the observed incompatibility between the DAA and the SAA learning rules. These hypotheses would require further investigation.

V. RELATED WORK

Several methods have been proposed in the literature for learning sparse features with a neural network-like architecture. We briefly summarize some of them before highlighting the originality of SAA.

Lee et al. augmented the so-called restricted Boltzmann machine (RBM) learning rule [12] by adding a temporal sparsity constraint undergone by each encoding neuron individually [21]. This constraint encourages their activation to have an expected value \( \rho \) close to the minimum of \( a^E \). In other words, each encoding neuron is forced to be only active for a small number of input examples. Even if this is not guaranteed in theory, one can expect that such a selectivity will lead to sparsity in the code because one can expect that a small number of encoding neurons will be active at the same time, i.e. for the same input example. In an online learning setting, one can keep a running estimate of the mean activation value \( \hat{\rho}_i \) of each encoding neuron \( n_i \). Then, it is possible to make \( \hat{\rho}_i \) closer to \( \rho \) by updating only the bias of \( n_i \) [21]. Another update method for selectivity consists in minimizing the cross-entropy loss between \( \rho \) and \( \hat{\rho}_i \) [22].

Goh et al. proposed another modification to the RBM learning rule so as to encourage at the same time sparsity and selectivity in the model [23]. In their method, the encoding neuron activation matrix \( \mathcal{M} \) is computed and transformed in order to fit some desired distribution \( P \) before updating the model. \( \mathcal{M} \) stores the training examples in its columns and the encoding neurons in its rows and the proposed method is only convenient for a batch learning setting. \( P \) has to be designed based on prior knowledge depending on the nature of the problem. The authors applied their work to modelling and classification of image data. They chose \( P \) to be positively skewed with heavy tails. Indeed, it was found from neural recordings that positive skewness and heavy tails are the characteristics of activity distributions for both selectivity and sparsity in biological neural networks.

Gregor and LeCun trained a neural network in a supervised manner to mimic sparse coding [24]. In their paradigm, each input example is given with its optimal sparse representation obtained through conventional dictionary-based methods and the neural encoder is trained to approximate this optimal code. Their architecture exhibits interesting features such as local competition between encoding neurons. Indeed, if two sets of neurons can reconstruct an input equally well, the algorithm activates only one of them and deactivates the other.

A simple strategy for encouraging sparse codes in a neuron layer is to use an activation function that naturally favors it, such as proposed by Glorot et al. with so-called rectifying functions [9]. Rectifying neurons are considered more biologically plausible than sigmoidal ones and can be used in conjunction with an additional sparsity constraint to even more promote sparse representations. But rectifying functions lead to several optimization difficulties in a gradient descent context and the authors proposed some heuristics to tackle possible issues.

We saw in section IV that our method does not need any prior knowledge about the optimal code to learn, as opposed to [23] and [24]. It is well suited to online learning, as opposed to [23], moreover without the need to memorize outdated and approximative information from past examples, as opposed to [21] and [22]. It is also very simple to understand and implement because it does not need any particular trick to be able to apply the standard back-propagation procedure, as opposed to [9]. On the other hand it is not related to biological neurons computation, as opposed to [23] and [9].

VII. CONCLUSION

We presented in this paper a method for learning sparse features of input examples by means of optimizing the encoding layer of an AA.

To do so, we augmented the standard AA learning scheme with an update rule enforcing low density in the encoding layer, which is expressed in terms of the activation of a small number of encoding neurons. This additional rule promotes codes corresponding to sparsified versions of the representations learned by the standard AA. By choosing an appropriate sparsification strategy such as thresholding, it is not necessary to set as an hyperparameter the desired level of encoding activity, that can be adapted to each input example, depending on the quantity of information contained in it.

We showed experimentally the effectiveness of the SAA method. Recall that sparse codes are theoretically more robust to noise and favor separability for classification tasks. We noticed this behaviour on three handwritten digit classification
problems, where the drawings of the digits were enclosed by background images embedding a variable quantity of information. In comparison to the standard AA, our method was always more capable of extracting salient features from the noise, the background, which led to better classification accuracy when the features where fed as input to a trainable classifier. Theses results were also as good as the ones obtained by classifiers fed with features learned by the competitive denoising variant of the AA.

Moreover the learning algorithm can result in an automatic selection of an appropriate encoding dimensionality for a particular dataset, as we saw that some neurons can be considered as useless after training and thus be removed without more classification accuracy loss. By initializing the number of encoding neurons with the input dimensionality and then removing useless neurons from the architecture after training, we can obtain both sparse and compressed representations. In experiments, these representations were still better for classification tasks than compressed representations of the same dimensionality learned by a standard AA. They were also better than compressed representations learned by the DAA when the compression level is rather hard. Naturally, some more informative examples than the MNIST digits would probably need an over-complete representation to be accurately represented by sparse activation vectors, i.e. a representation of a higher dimensionality than the input one [25]. Then a good strategy could be to initialize the encoding layer size to a large number and to let the pruning stage select a relevant dimensionality for the sparse representation (possibly larger than the input dimensionality). The effectiveness of the pruning stage is due to the fact that the SAA algorithm tends to learn simultaneous sparse coding, even if the common sparsity profile of the new representation of the examples is never explicitly stated as an objective to be reached.

This paper raises several questions. For instance, it could be challenging to study the potential of the SAA method to be stacked in a greedy manner to perform deep feature learning [8], [26]. Notice that it should not be fully relevant to directly stacked in a greedy manner to perform deep feature learning challenging to study the potential of the SAA method to be learned simultaneous sparse coding, even if the common sparsity

## 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


