A Sparse and Locally Shift Invariant Feature Extractor Applied to Document Images

Marc'Aurelio Ranzato Yann LeCun
Courant Institute of Mathematical Sciences
New York University - New York, NY 10003

Abstract

We describe an unsupervised learning algorithm for extracting sparse and locally shift-invariant features. We also devise a principled procedure for learning hierarchies of invariant features. Each feature detector is composed of a set of trainable convolutional filters followed by a max-pooling layer over non-overlapping windows, and a point-wise sigmoid non-linearity. A second stage of more invariant features is fed with patches provided by the first stage feature extractor, and is trained in the same way. The method is used to pre-train the first four layers of a deep convolutional network which achieves state-of-the-art performance on the MNIST dataset of handwritten digits. The final testing error rate is equal to 0.42%. Preliminary experiments on compression of bitonal document images show very promising results in terms of compression ratio and reconstruction error.

1. Introduction

Most computer vision systems for document analysis as well as for general image understanding include a feature extractor as first pre-processing step. Principal Component Analysis and K-Means are the most well known techniques used in these vision applications. The first problem we address in this paper is how we can learn invariant features. Some methods build invariance from pre-determined features based on local histograms such as the SIFT descriptors [8]. However, learning these features could make them more adaptive to the particular dataset in use. Other methods [11, 9, 10] achieve indirectly invariance to small shifts thanks to a sub-sampling layer which is added after the training phase is completed. The method we propose includes and integrates efficiently invariance in the learning stage. Each feature extractor is composed of a set of linear filters that are convolved with the input image, followed by a max-pooling layer which selects the largest values within non-overlapping windows, and by a point-wise sigmoid non-linearity. Sec. 2.1 and 2.4 describes the details of the architecture used during training and the learning algorithm which allows us to learn invariant and sparse features.

The second problem we address in the paper is how we can learn hierarchies of features in a principled way. Hierarchical feature extractor have already been proposed in the literature, but some of them introduce rather ad hoc learning procedures [4], or propose to hard-wire Gabor filters [11, 9], or seem inefficient when dealing with deep multi-layer systems and few labelled samples [7, 6]. Several recent works have shown the advantages (in terms of speed and accuracy) of pre-training each layer of a deep network in unsupervised mode, before tuning the whole system with a gradient-based algorithm [5, 2, 10]. The present work is inspired by these methods. Sec. 2.5 describes how we can learn layer by layer feature detectors with increasing level of invariance and complexity, and sec. 3 demonstrates the method for the classification of handwritten numerals and for compression of bitonal document images.

2. The System

The system we describe is derived from the Energy-Based Model for learning sparse features introduced by Ranzato et al. [10]. We extend that model by devising an architecture which produces features that are not only sparse, but also locally invariant to shifts. Moreover, we propose a hierarchical extension which learns features that correspond to larger and more complex receptive fields in input space.

2.1 Architecture

During training the architecture of the system is composed of an *encoder* and a *decoder* as shown in fig. 1. The encoder takes image patches Y and computes a prediction of the optimal internal representation Z, namely the *code*. We postpone the definition of code optimality until the later discussion. The decoder produces a reconstruction of Y

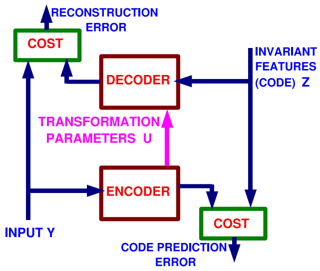


Figure 1. The *encoder-decoder* architecture for unsupervised learning of features. In an *invariant* feature extractor, Z represent the invariant part of the code while the variables U encode the transformation which the invariant features must undergo in order to reconstruct the input.

from the input code Z. The model has an associated energy which is the weighted sum of two terms, the encoder and decoder energies. The decoder energy is the square Euclidean distance between the output of the decoder and the input Y. The encoder energy is the square Euclidean distance between the encoder prediction and the optimal code Z. The code Z is optimal when it minimizes the overall energy of the system. At the same time it minimizes the reconstruction error of Y, and it is as close as possible to the encoder output for the given set of parameters in the system.

The problem regarding invariant feature extraction in such architecture is about the reconstruction. Can we decode the input from its invariant representation? Unfortunately, we cannot. However, we can always augment the internal representation with the information necessary to reconstruct the input (i.e. the non-invariant part of the code) and use it for decoding. This is what we call *transformation parameters* in fig. 1. The encoder extracts both the transformation parameters U, and the invariant code Z. The parameters U are copied into the decoder in order to allow the reconstruction. For instance in our shift invariant model, Z represents *what* features are present in the input patch while U represents *where* these features appear in the image. Generally, the transformation parameters and the decoding are necessary only during the learning phase in order to assess whether the code is good in preserving enough information from the input, so that we are able to obtain accurate reconstructions. In applications where the goal is to extract invariant features, the transformation parameters and the decoder are disregarded after the training phase because the interest is only in the invariant code Z produced by the encoder.

2.2 Invariance to Shifts

In this section we present an instance of the previously described general model. This model allows the extraction of locally shift-invariant features. The encoder is composed of a set of filters that are convolved with the input, and a max-pooling layer. The filter bank performs feature detection and produces a set of feature maps. How the filters are learned is explained in sec. 2.4. The following max-pooling layer operates feature map by feature map separately, and selects the *largest* values in non overlapping windows. The resulting spatially reduced representation constitutes the locally shift-invariant code Z, while the positions of the selected largest values are the transformation parameters U. No matter where the features appear in the input patch, the code Z will be always the same and only the parameters U will be affected. The parameters U are copied into the decoder which is also composed of a set of linear filters. The reconstruction is computed by placing each code value of Z at the proper location in the decoder feature map, using the transformation parameters U obtained in the encoder, and setting all other values in the feature maps to zero. The reconstruction is simply the sum of the decoder basis functions weighted by the feature map values at all locations.

2.3 Sparsity

Sparsity is achieved by inserting a non-linearity in front of the decoder, dubbed Sparsifying Logistic as in [10]. This non-linearity is an adaptive logistic with a very high threshold which enforces sparsity of its output across training samples. After training the threshold is fixed, and the logistic turns into a standard logistic function (with a large bias). The sparsifying logistic module transforms the input code vector Z into a sparse code $\bar{Z}$ vector with positive components between $[0, 1]$. Let us consider the k-th training sample and the i-th component of the code, $z_i(k)$ with $i \in [1..m]$ where m is the number of components in the code vector. Let $\bar{z}_i(k)$ be its corresponding output after the sparsifying logistic. Given two parameters $\eta \in [0, 1]$ and $\beta > 0$, the transformation performed by this non-linearity is given by:

$$\bar{z}_i(k) = \frac{e^{\beta z_i(k)}}{\zeta_i(k)}, \text{ with } \zeta_i(k) = e^{\beta z_i(k)} + \frac{(1-\eta)}{\eta}\zeta_i(k-1) \quad (1)$$

This can be seen as a kind of weighted "softmax" function over past values of the code unit. This adaptive logistic can output a large value, i.e. a value close to 1, only if the unit has undergone a long enough quiescent period. The parameter η controls the sparseness of the code by determining the length of the window over which samples are summed up. β controls the gain of the logistic function, with large values yielding quasi-binary outputs.

Recalling the shift-invariant model described earlier, we have that the code Z is shift-invariant while the transformed code $\bar{Z}$ will be not only shift-invariant, but also sparse. This is the code that is used by the following decoder modules that perform the weighted sum of basis functions.

2.4 Learning Algorithm

Let W_C and W_D be the trainable parameters in the encoder and decoder, respectively. These parameters are the set of filters in the encoder, and the set of basis functions in the decoder. The goal of the learning algorithm is to find a value for W_C and W_D that minimize the energy of the system over the training dataset. This energy is the weighted sum of encoder energy E_C and decoder energy E_D. Denoting the output of the encoder with $\mathrm{Enc}(Y; W_C)$ and the output of the decoder with $\mathrm{Dec}(Z, U; W_D)$, these quantities are defined as: $E_C = ||Z - \mathrm{Enc}(Y; W_C)||^2$ and $E_D = ||Y - \mathrm{Dec}(Z, U; W_D)||^2$.
Then, we need to solve for $min_{W_C,W_D} min_Z E_C + \alpha E_D$, where α has been set to 1 in our experiments. Learning proceeds in a EM-like fashion with the following *on-line* algorithm:

1. propagate the input through the encoder to produce a prediction Z_0 of the optimal code Z, and copy the transformation parameters U into the decoder

2. keeping both U and the set of parameters W_C and W_D fixed, find by gradient descent the code Z^* that minimizes the energy $E_C + \alpha E_D$ starting from the initial value Z_0 that was provided by the encoder

3. keeping fixed the code at Z^*, update the decoder parameters W_D by one step of gradient descent in the decoder energy

4. update the encoder parameters W_C by one step of gradient descent in the encoder energy where Z^* will play the role of target value for the encoder output.

After training, the system converges to a state where minimum energy codes Z are predicted by the encoder in one shot without the need for a minimization in code space, and the decoder produces good reconstructions from the encoder output.

2.5 Hierarchies of Locally-Invariant Features

Once the system is trained, it can be applied to larger images in order to extract locally invariant feature maps. These feature maps can be used to train another machine which will produce features that are more invariant and more complex than the first level features. Disregarding the effect of the filter size used in the convolution, let us assume that the input image has size pxq, and that the first level feature extractor performs a pooling in NxN neighborhoods while the second level feature extractor pools in a MxM neighborhood. While the output of the first level feature extractor of (approximate) size p/Nxq/N is invariant in NxN max-pooling windows, the output of the second level feature extractor is invariant in MNxMN windows in input space. Moreover, the second level feature extractor combines many first level feature maps into each output feature map increasing in this way its representational power for encoding complex patterns in input space. The connection between each set of input feature maps with a single output feature map is given by a pre-determined table, which in the experiments described in sec. 3.1 is random.

Learning a hierarchy of feature extractors proceeds in sequence. Each level is trained separately and, when trained, it provides the input for the next higher level feature extractor. As a final step, a global relaxation throughout the whole system can be done in order to fine tune the parameters. This procedure was originally proposed by Hinton et al. [5] for training deep belief nets.

3. Experiments

We present two applications of the proposed unsupervised method for learning invariant features. In the first example we have considered the MNIST dataset [1] of handwritten digits, and we have used the algorithm to pre-train the first layers of a deep convolutional network. After this unsupervised layer-by-layer training yielded a hierarchical shift-invariant feature extractor, all the parameters of the network were tuned together in a supervised way, as proposed by Hinton et al. [5]. The second example shows a straightforward application of this algorithm for compression of bitonal text images yielding very promising results. In both experiments, η and β in the Sparsifying Logistic are fixed to 0.01 and 1, respectively. Also, a small lasso regularization term of the order of 0.001 is added to the energy loss.

3.1 Classification of MNIST dataset

In this section we describe the training protocol of a 6 layer convolutional network trained on the MNIST dataset. The training dataset was augmented with elastically distorted digits, as discussed in Simard et al. [12]. For each training sample 10 distorted replicas have been added to the set, making the total number of training digits 660,000. For training we have selected 5,000 digits (from the original training dataset) for validation.

The training protocol is the following. First, we train *unsupervised* on the whole original training dataset the first

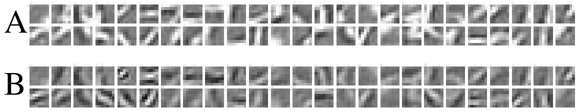


Figure 2. (A) The fifty 7x7 filters that were learned by the sparse and shift-invariant model trained on the MNIST dataset. These filters are used to initialize the first layer of a deep convolutional network. (B) The fifty 7x7 filters in the first layer of the convolutional network after supervised training on the augmented dataset.

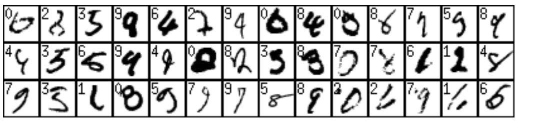


Figure 3. The 42 misclassified digits in the testing MNIST dataset. On the upper left corner there is the true label.

four layers of the convolutional network with the hierarchical feature extractor described in the previous sections. Second, we train *supervised* on the whole training dataset the top two layers using the features provided by the feature extractor. By taking the parameters that gave the minimum loss on the validation set, we have found the initial value of the parameters for the last supervised training of the *whole* network, including the first four layers. Testing is done on the parameters that gave the minimum loss on the validation dataset.

In particular, the unsupervised training of the first four layers is done as follows. First, we learn the filters in the convolutional layer with the sparsifying encoder-decoder model described in sec. 2.4 trained on patches randomly extracted from training images. We learn fifty 7x7 filters that capture feature that are invariant in a 2x2 neighborhood since the max-pooling layer considers 2x2 windows. Once training is complete, the encoder and decoder filters are frozen, and the sparsifying logistic is replaced by a $\tanh$ sigmoid function with a trainable bias and a gain coefficient. The bias and the gain are trained with a few iterations of back-propagation through the encoder-decoder system. The rationale for relaxing the sparsity constraint is to produce representation with a richer information content. While the the sparsifying logistic drives the system to produce good filters, the quasi-binary codes produced do not carry enough information for the later classification. Then, training images are run through this level to generate patches for the next level in the hierarchy. The second level feature extractor (i.e. layer 3 and 4 of the convolutional net) has 1,280 filters of size 5x5 that connect 10 randomly chosen input feature maps to each output feature map. There are 128 output feature maps that are subsequently max-pooled over 2x2 neighborhoods. This second feature extractor is trained in the same way as before. Once the feature extractor is fed with 34x34 padded digits, it produces features of size 128x5x5 that are given to the top two layers of the convolutional network. These layers form a two-layer neural net with 200 hidden units and 10 output units. The supervised training of both the top two layers as well as of the whole network is done by error back-propagation using a loss which is the average square Euclidean distance between the output of the net and the target value of the input digit (a 1 of N code of the label). The error rate on the testing dataset is equal to 0.42%, very close to 0.39% which is the record [10] for this dataset. For comparison, training the same network from random initial condition by supervised back-propagation of gradients yields an error rate equal to 0.48%.

3.2 Compression of Text Document Images

In this section we describe how we can apply the unsupervised learning algorithm to compression of document images. We considered the CCITT black and white test image number 5, which contains both text and drawings. With a resolution of 200dpi, the image has size 2376x1728 pixels. First, we ran a connected component (CC) analysis which revealed 1,421 components. We considered patches of size 30x30 that cover about 95% of these CC's, and we built a dataset of patches that we used to train the system. If a CC is bigger than 30x30 pixel, it is split in chunks of 30x30 pixels. If it is smaller, it is centered in a 30x30 patch. In total there were 3102 patches, some are shown in fig. 5. We considered a (single stage) invariant feature extractor with 256 30x30 filters in both encoder and decoder. Since the input is evenly padded in a 34x34 window, the max-pooling layer operates in a 5x5 neighborhood. Encoding consists of performing the CC analysis, encoding the locations of the patches that are extracted from the document (whose entropy is 5.16 bits), propagating the patch through the encoder filter bank and the Sparsifying Logistic, thresholding the code in order to make it binary, and finally, encoding both the code and locations of the largest values selected by the max-pooling layer (which amounts in 13.9 bits per patch). The decoding has to uncompress the bit stream, use the decoder filters to reconstruct the codes and the transformation parameters, and place the thresholded reconstruction in the given location in the image. Ideally, the filters

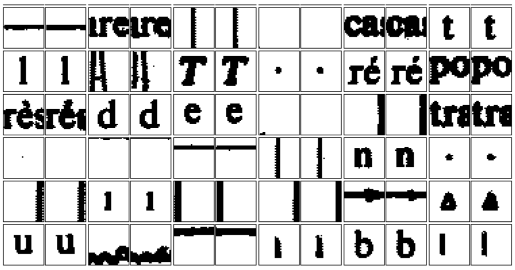
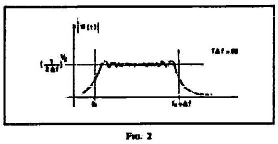


Figure 4. LEFT PANELS **Examples of windows that are used for training and compressing the CCITT test page number 5.** RIGHT PANELS **Reconstructions provided by the algorithm.**



$$\varphi = -2\pi\left[T_0+\frac{f_0T}{\Delta f}\right]f+\pi\frac{T}{\Delta f}f^2$$

Et cette phase est bien l'opposé de $/\phi(f)$, à un déphasage constant prés (sans importance) et à un retard T_0 près (inévitable).

Un signal utile $S(t)$ traversant un tel filtre adapté donne à la sortie (à un retard T_0 près et à un déphasage prés de la porteuse) un signal dont la transformée de Fourier est réelle, constante entre f_0 et $f_0+\Delta f$.

Figure 5. Detail of the test CCITT image number 5 at 200dpi which has been compressed and reconstructed by our algorithm.

would be learned from a variety of images with different fonts and would be hard-wired in the decoder. Under the assumption that we do not need to encode also the decoder filters, the compression ratio would be equal to 95. The percentage of reconstructed pixels whose value has been flipped is equal to 1.35%. An example of patches that are encoded and reconstructed is shown in fig. 4. Fig. 5 shows a detail of the reconstructed image. For comparison, the state-of-the-art method for compressing images is the JB2 [3] which achieves a lossy compression ratio equal to 55 and produces a visually lossless result on this test image.

4. Conclusions

We have described an unsupervised method for learning hierarchies of shift-invariant features. We have applied this algorithm to pre-train the first four layers of a deep convolutional network achieving state-of-the-art performance in the classification of handwritten digits in the MNIST dataset. An application in compression using a single level invariant feature extractor has also been presented and shows promising results.

In this paper we have defined a novel method for learning locally shift invariant features. Moreover, we have proposed a general and principled method for learning hierarchies of features. In our experiments, we have limited ourselves to two layers of features but one could stack as many modules like this as one needs in order to achieve the desired level of invariance and complexity. This method is useful to pre-train deep architectures such as convolutional neural networks. But also, it can be used to learn and extract features from datasets with few labelled examples, and it can be used to learn complex invariant features with the desired dimensionality.

The future work will take into account other learning algorithms to train multiple levels of feature extractor all together, the definition of a more efficient and hierarchical method for compression, the possibility to learn also the parameters in the Sparsifying Logistic which controls the sparsity of the representation, and a more exhaustive experimentation in order to understand better in which conditions unsupervised learning is beneficial to supervised learning.

References

[1] http://yann.lecun.com/exdb/mnist/.

[2] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *NIPS*. MIT Press, 2007.

[3] L. Bottou, P. Haffner, P. Howard, P. Simard, Y. Bengio, and Y. LeCun. High quality document image compression with djvu. *Journal of Electronic Imaging*, 7(3):410–425, 1998.

[4] K. Fukushima and S. Miyake. Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern Recognition*, 15(6):455–469, 1982.

[5] G. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

[6] F.-J. Huang and Y. LeCun. Large-scale learning with svm and convolutional nets for generic object categorization. In *CVPR*. IEEE Press, 2006.

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

[8] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[9] J. Mutch and D. Lowe. Multiclass object recognition with sparse, localized features. In *CVPR*, 2006.

[10] M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun. Efficient learning of sparse representations with an energy-based model. In *NIPS*. MIT Press, 2006.

[11] T. Serre, L. Wolf, and T. Poggio. Object recognition with features inspired by visual cortex. In *CVPR*, 2005.

[12] P. Simard, D. Steinkraus, and J. Platt. Best practices for convolutional neural networks. In *ICDAR*, 2003.