Unsupervised Learning of Feature Hierarchies

by

Marc'Aurelio Ranzato

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

May 2009

Yann LeCun

DEDICATION

To my parents and siblings.

ACKNOWLEDGMENTS

I would like to thank my advisor Prof. Yann LeCun for his guidance and for the many opportunities he has offered me since the very beginning of my studies. I am also grateful to the members of my committee for the discussions we had during these years and for providing me with their precious insights that were helpful to gain different perspectives on my work. I want also to acknowledge fruitful discussions with the members of LeCun's lab at New York University.

ABSTRACT

The applicability of machine learning methods is often limited by the amount of available labeled data, and by the ability (or inability) of the designer to produce good internal representations and good similarity measures for the input data vectors. The aim of this thesis is to alleviate these two limitations by proposing algorithms to *learn* good internal representations, and invariant feature hierarchies from unlabeled data. These methods go beyond traditional supervised learning algorithms, and rely on unsupervised, and semi-supervised learning.

In particular, this work focuses on "deep learning" methods, a set of techniques and principles to train hierarchical models. Hierarchical models produce feature hierarchies that can capture complex non-linear dependencies among the observed data variables in a concise and efficient manner. After training, these models can be employed in real-time systems because they compute the representation by a very fast forward propagation of the input through a sequence of non-linear transformations. When the paucity of labeled data does not allow the use of traditional supervised algorithms, each layer of the hierarchy can be trained in sequence starting at the bottom by using unsupervised or semi-supervised algorithms. Once each layer has been trained, the whole system can be fine-tuned in an end-to-end fashion. We propose several unsupervised algorithms that can be used as building block to train such feature hierarchies. We investigate algorithms that produce sparse overcomplete representations and features that are invariant to known and learned transformations. These algorithms are designed using the Energy-

Based Model framework and gradient-based optimization techniques that scale well on large datasets. The principle underlying these algorithms is to learn representations that are at the same time sparse, able to reconstruct the observation, and directly predictable by some learned mapping that can be used for fast inference in test time.

With the general principles at the foundation of these algorithms, we validate these models on a variety of tasks, from visual object recognition to text document classification and retrieval.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

In many real world applications labeled data is too scarce to fit the parameters of those models that could describe it well and is too expensive to produce. Moreover, labels are often noisy in the sense that errors may be present in the labeling process. For instance, consider the problem of building an image retrieval system for the web. Traditional learning algorithms would need to have access to pairs consisting of a query and an image, and their relative score. However, it is impossible to have access to such information for all possible images and queries. Moreover, the few labeled samples that are available might be generated by taking into account the user click through data which is naturally noisy.

One way to cope with the paucity of labeled data is to engineer as much as possible the prediction system by exploiting the prior knowledge and the experience of human experts. This effectively reduces the number of parameters of the model and regularizes the system. For instance, a good description of natural images can be computed by using wavelets transforms (Simoncelli et al., 1998) or hand-designed descriptors (Lowe, 2004; Dalal and Triggs, 2005), and the prediction system can take advantage of such representation. However, such a system would not be able to easily adapt to other domains because other kinds of data might have different statistics and require different representations.

In this thesis, we propose a more general approach that relies on learning, and that allows adaptation to a variety of domains. While labeled data is scarce, unlabeled data

is often available in large amounts at virtually no cost. For instance, billions of images can be easily downloaded from the web. Our approach is to leverage unlabeled data to learn representations that capture the statistics of the input. Since both unlabeled and labeled data share the same underlying structure, the learned representations can provide a description in terms of typical features or frequent patterns occurring in the input data. Such representation is often more concise and more descriptive than the raw input data. Moreover, many popular supervised algorithms (Boser et al., 1992; Rasmussen and Williams, 2006) compute similarity measures between pairs of input samples and strongly rely on the representation used. In other words, the better the representation of the data is, the easier the subsequent classification will be.

In real world applications the representation has to be computed efficiently and it has to describe the input concisely. The former requirement implies that the computation has to be a fast feed-forward process, not involving iterative optimization procedures. The latter requirement is related to the concept of efficient coding (Attneave, 1954; Barlow, 1961), stating that units in the representation should have reduced dependencies. The most well known algorithm used for reducing statistical dependencies is principal component analysis which is able to remove second order correlations. More recently, independent component analysis (Bell and Sejnowski, 1995; Olshausen and Field, 1997; Hyvarinen et al., 2001) has been introduced as a method to remove higher order dependencies. However, recent studies (Bethge, 2006; Wegmann and Zetzsche, 1990; Simoncelli, 1997; Lyu and Simoncelli, 2008) suggest that these linear transformations leave strong higher order dependencies and that the use of non-linear transformations is needed to remove them.

In this thesis we consider a general class of trainable non-linear functions, dubbed "deep networks" (Hinton et al., 2006; Hinton and Salakhutdinov, 2006; Bengio and LeCun, 2007; Bengio et al., 2007; Ranzato et al., 2007c; Lee et al., 2007). These models are composed of a sequence of non-linear transformations whose parameters are optimized to fit the data. If we take into account the intermediate representations produced by each layer in the sequence, we can interpret the deep network as a model producing a bottom up hierarchy of features. The representation becomes more and more abstract as it is transformed by more layers, and the hope is that the top level representation will be more closely related to the causes generating the data and to the labels we might want to predict.

A simple experiment reported in chapter 5 clarifies the abstraction achieved by such systems (Ranzato et al., 2007b). A deep network with two layers is trained on hand-written digits. While the first layer learns features that capture correlations between neighboring pixels in the form of digit "strokes", the second layer trained with only ten units learns longer range dependencies and it combines the first layer strokes into ten digit prototypes, one per class. Even though the network was trained without making use of labels, it discovered the highly non-linear mapping between input pixels and class labels in its top layer representation.

Deep networks are appealing because they lead to more efficient representations (Bengio and LeCun, 2007) since a top level representation can be defined by re-using intermediate computations, limiting the number of parameters and the number of computational units. The historical problem of these methods is that the optimization is very hard because it is highly non-linear and non-convex (Tesauro, 1992). Until a few years

ago, no network with more than a couple of layers could be successfully trained. The only exception were convolutional networks (LeCun et al., 1998) that exploit a highly constrained architecture due to the weight sharing. However, these networks are specifically designed for images, and they need quite a large number of labeled samples to train.

A more general solution was proposed by Hinton and collaborators (Hinton et al., 2006). They showed that a deep network can be trained in two steps. First, each layer is trained in sequence by using an unsupervised algorithm to model the distribution of the input. Once a layer has been trained, it is used to produce the input to train the layer above. After all layers have been trained in an unsupervised way, the whole network is trained by traditional back-propagation of the error (e.g., classification error), but the parameters are initialized using the weights learned in the first phase. Since the parameters are nicely initialized, the optimization of the whole system can be carried out successfully. This procedure and similar ideas have been applied to a variety of domains, such as computer vision (Hinton and Salakhutdinov, 2006; Ranzato et al., 2007c; Ranzato et al., 2007b; Larochelle et al., 2007; Vincent et al., 2008; Ahmed et al., 2008; Torralba et al., 2008), natural language processing (Salakhutdinov and Hinton, 2007a; Mnih and Hinton, 2007; Ranzato and Szummer, 2008; Weston et al., 2008; Collobert and Weston, 2008; Mnih and Hinton, 2008), robotics (Hadsell et al., 2008) and collaborative filtering (Salakhutdinov et al., 2007).

At a very high level, these works have demonstrated that it is possible to train feed-forward hierarchical models using unsupervised as well as semi-supervised and multi-task learning algorithms (Ranzato and Szummer, 2008; Weston et al., 2008; Collobert

and Weston, 2008; Ahmed et al., 2008). It remains an open research question to identify even better training protocols and to adapt those to the specific task at the hand.

Since the key to training deep networks is the use of an unsupervised learning algorithm, chapter 1 describes a general framework to design these algorithms, the *Energy-Based Model* framework (LeCun et al., 2006; Ranzato et al., 2007a). Energy-Based Models are non-normalized probabilistic models that assign an energy value to the joint set of observed and predicted variables. These models can be thought of as a local probabilistic model assigning higher likelihood to training samples only in regions of the input space that are of interest. This framework permits a richer class of algorithms than properly normalized probabilistic models, and allows greater computational efficiency both during training and inference. According to this framework, the goal of learning is to adjust the parameters of the model in such a way that points that are similar to training samples are assigned lower energy. In order to achieve this goal a loss functional is minimized during training. Although all loss functionals decrease the energy in correspondence of the training samples, they differ in the way they make sure that other points have higher energy: some require to identify candidate points where the energy has to be raised and others enforce more global constraints on the internal representation, such as sparsity or compactness. We show the equivalence between these two strategies and give an interpretation of traditional unsupervised algorithms in this framework. Chapter 2 provides simple visualizations of the energy surface on toy datasets in order to give a better intuition of these concepts. Since raising the energy by constraining the internal representation is more computationally efficient in high-dimensional spaces, we have investigated several sparse coding algorithms for feature extraction. Chapter 3 describes

one such algorithm, Predictive Sparse Decomposition, using the principles and the ideas developed in the previous chapters.

In chapter 4 the Predictive Sparse Decomposition algorithm is extended to learn representations that are not only sparse, but also invariant to either known or learned transformations. Learning representations that are invariant to irrelevant transformations of the input is crucial towards building robust recognition systems. Invariant representations are desirable because they are more compact and they can be used by even simple recognition systems, since they do not encode irrelevant properties of the input data. For instance, a face detector should be invariant (or robust) to the pose of the subject, to lighting conditions, and to facial expressions, while still encoding the information that is necessary to locate and identify a face. In particular, much of the progress in computer vision is based on hand-designed descriptors that are invariant to lighting conditions, and changes in scale and orientation (Schmid and Mohr, 1997; Lowe, 2004; Lazebnik et al., 2004; Dalal and Triggs, 2005). However, these methods work well only on natural images for which they were designed, and they are conceivably sub-optimal once a large dataset of examples is available. Therefore, designing a generic algorithm that can learn representations that are invariant to learned transformations can make possible the development of a system that adapts to the data in an end-to-end fashion.

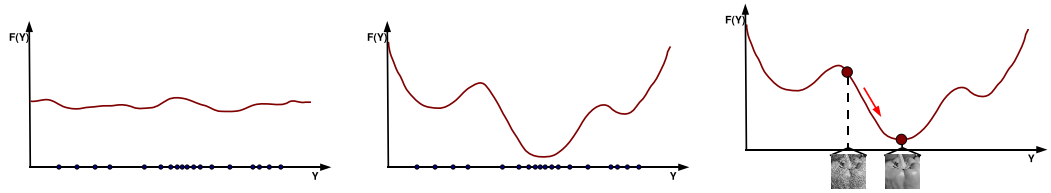
Finally, chapter 5 demonstrates how these unsupervised algorithms can be used to build deep networks and reports several experiments, ranging from visual object recognition to text document classification and retrieval.

1

ENERGY-BASED MODELS FOR UNSUPERVISED LEARNING

Unsupervised learning algorithms capture regularities in the data for the purpose of restoring corrupted data or for extracting representations of the data that can be used for tasks such as prediction, classification, or visualization. We will view an unsupervised machine as a function $F(Y)$ that maps input vectors Y to *scalar energy values*. An unsupervised machine captures dependencies between input variables by producing low energy values in regions of high data density, and higher energy values in regions with little or no data.

For instance, figure 1.1(a) shows an energy surface before training. The energy is not lower around areas of high data density. At this stage, the machine is not able to predict if an input data vector is similar to the samples in the training set. However, after training the energy takes the desired shape as shown in figure 1.1(b), that is, it is lower around high data density areas. Figure 1.1(c) shows how such a model could be used for denoising. The denoised image is computed by searching for the minimum of the energy that is closest to the input sample. Loosely speaking, this process returns the most likely data vector nearby the noisy input. This task, and more generally, estimating regions of high data density can be accomplished only if the energy is lower around areas of high data density. In this sense, a model assigning an energy that is constant over the whole input space has failed to learn because any data vector gets the same "score" as a



(a) (b) (c)

Figure 1.1: Toy illustration of an unsupervised EBM. The blue dots are training samples, the red curve is the energy surface. (a) Before training, the energy does not have the desired shape and the model does not discriminate between areas of high and low data density. (b) After training, the energy is lower around areas of high data density. (c) The model can be used for denoising, for instance. Denoising consists of finding the nearest local minimum nearby the noisy observation.

training sample under the model. This paper describes the principles behind successful learning of energy functions, and it introduces a common framework, the *energy-based model* framework, to describe most unsupervised learning algorithms.

A particularly important class of unsupervised algorithms, which includes principal component analysis, K-means, and many others, produces internal *representations* of data vectors as part of the energy computation. These representations, also known as *feature vectors*, or *codes* can be used as input for further processing such as prediction. Moreover, many unsupervised machines make explicit use of such representations by *reconstructing the data vectors from the representations*, and by using the reconstruction error as part of the energy function. For example in clustering methods such as K-Means (or vector quantization), the code is the index of the prototype in the codebook that is closest to the data vector. The reconstruction error is the distance between the data

vector and its closest prototype. Similarly in principal component analysis, the code is the set of coordinates of the projection of the data vector on a linear subspace, and the reconstruction error is the distance between the data vector and its projection. In auto-encoder neural networks (Rumelhart et al., 1986), the code is the state of a low-dimensional hidden layer from which the data vector is reconstructed with a possibly non-linear mapping. In restricted Boltzmann machines (RBMs) (Freund and Haussler, 1994; Hinton, 2002), the code is a vector of stochastic binary variables, from which the input can be (stochastically) reconstructed, even though the machines are not explicitly trained to reconstruct but to maximize log likelihood. Finally, in sparse coding and other related methods (Lee and Seung, 1999; Olshausen and Field, 1997; Aharon et al., 2005; Ranzato et al., 2006; Lee et al., 2007), the code is a high-dimensional vector in which most of the components are constrained to be zero (or near zero), and the energy is the reconstruction error under sparsity constraints.

Unlike these methods, some probabilistic density models like Product of Experts methods (Teh et al., 2003; Ning et al., 2005; Roth and Black, 2005) do not use the internal representations to reconstruct the input data, but only to compute the negative log likelihood. This can be interpreted as their energy function and it has the property that the difference of energies of two points is equal to their log likelihood ratio.

Training an unsupervised machine consists in shaping the energy landscape so that regions of high data density have lower energies than other regions. This is generally achieved by parameterizing a family of energy functions $\{F(Y;W), \quad W \in \mathcal{W}\}$ indexed by a parameter W, and by searching for the W that minimizes a particular loss functional that depends on F and on the training set. We will show that essentially every

unsupervised learning algorithm has a term in the loss functional whose purpose is to decrease the energy of the training samples. However, different algorithms use different techniques to ensure that the energy values associated with regions of low data density are higher.

Unsupervised methods appear very diverse, and based on very different principles. We argue that the various unsupervised methods merely differ on two points: (1) how $F(Y;W)$ is parameterized, and (2) how the loss functional is defined, particularly how the energy of unobserved points is made larger than the energy around training samples. This work discusses which combinations of architectures and loss functionals are allowed, which combinations are efficient, and which combinations do not work. One problem is that pulling up on the energies of unobserved points in high dimensional spaces is often very difficult and even intractable. In particular, we show that probabilistic models use a particular method for pulling up on the energy of unobserved points that turns out to be very inefficient in many cases. We propose new loss functionals for pulling up energies that have efficiency advantages over probabilistic approaches. We show that unsupervised methods that reconstruct the data vectors from internal codes can alleviate the need for explicitly pulling up on the energy of unobserved points by limiting the information content of the code.

1.1 Energy-Based Models for Unsupervised Learning

Unsupervised algorithms often compute internal representations of input data vectors. In density estimation models, such as mixture of Gaussians or Product of Experts models, these representations are implicit because they are only used to produce a likelihood

value. Otherwise, internal representations are often used to reconstruct the input, ensuring that most of the information contained in the input has been captured by the model. Internal representations are useful in a variety of applications such as dimensionality reduction, feature extraction, and clustering. These representations are referred to as *codes*, or features, and they can have desirable properties such as sparsity, compactness, and independence of the components.

Probabilistic unsupervised models can be graphically represented as in fig. 1.1. There

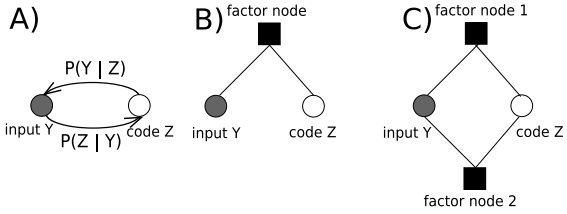


Figure 1.2: Probabilistic graphical models of unsupervised learning. The set of observed variables is denoted by Y, while the set of latent variables, or *codes*, is denoted by Z. **A)** A loopy Bayes network modelling two consistent conditional distributions, one predicting the latent code from the input, and another one predicting the input from the code. This model would be able not only to generate data, but also to produce fast inference of Z; unfortunately, learning is intractable in general. **B)** A factor graph describes the constraint between input and latent variables by connecting them with a factor node. The joint distribution between Y and Z can have two factors, as shown in **C)**. Many unsupervised models have one factor measuring the compatibility between Y and some transformation of Z, and another factor measuring the compatibility between Z and a transformation of Y. Unlike the model in A), the factor nodes are not necessarily modelling conditional distributions.

are two sets of variables, the observed input vector $Y \in \mathcal{R}^M$ and the latent code $Z \in \mathcal{R}^N$, whose value has to be inferred. Training such models generally means adjusting the parameters in such a way that the marginal distribution over Y gives higher likelihood to the training vectors, and also, that the joint distribution over Y and Z assigns higher likelihood to training vectors and corresponding "compatible" codes. The ideal probabilistic model is the loopy Bayes network shown in fig. 1.1 A). This model includes a module generating data from codes, that can easily check how well the model fits the data, and also, another module directly inferring the code from the input. Unfortunately, learning two consistent conditional distributions in a loopy graph is generally intractable. A more general representation is given by the factor graph in fig. 1.1 B), where the factor node describes the compatibility constraint Y and Z have to meet in order to be assigned high likelihood value. This model can be extended by considering a joint distribution that factorizes into two factors as shown in fig. 1.1 C). Often, one factor measures the compatibility between Y and some transformation of Z, while the other factor considers Z and some transformation of Y.

Indeed, factors often have a preferred "directionality", favoring inference of one variable given the other one. Any model can be interpreted as belonging to one of the following classes:

- an **encoder** model that provides a direct mapping of input data into a feature representation; the PoE model proposed by (Teh et al., 2003) and ICA based on information maximization (Herault and Jutten, 1986; Jutten and Herault, 1991; Bell and Sejnowski, 1995) are examples of such a model. While producing representations of input data is straightforward, generating data from the model is generally

complicated, requiring the use of expensive Monte Carlo sampling techniques.

- a **decoder** model that is based on a generative model reconstructing the input from an internal latent representation; a mixture of Gaussians as well as generative ICA based on maximum likelihood (MacKay, 1999) and traditional sparse coding algorithms (Olshausen and Field, 1997) can be interpreted in this way. While generating data is straightforward, inferring the representation might require computationally expensive marginalization or minimization procedures.

- an **encoder-decoder** model that has both a factor producing direct representations as well as another factor reconstructing the data from it; the most popular (non-probabilistic) encoder-decoder model is PCA and the most notable probabilistic model of this kind is RBM. Both data generation and feature extraction are easy in this model, but learning might be very difficult because of the normalization requirement of the model.

An energy-based model (EBM) (LeCun et al., 2006; Ranzato et al., 2007a) is a model that assigns lower energy values to input vectors that are similar to training samples and higher energy values elsewhere. Un-normalized models are much more computationally efficient in large and high dimensional spaces because they require the energy to be higher only within a suitable *neighborhood* of the training samples. For instance, in image restoration the corrupted data is usually near the "clean" data, and restoring a corrupted input vector may be performed by finding an area of low energy near that input vector (Teh et al., 2003; Portilla et al., 2003; Elad and Aharon, 2006). As it will be discussed in sec. 1.2.1, a probabilistic model is a special kind of EBM.

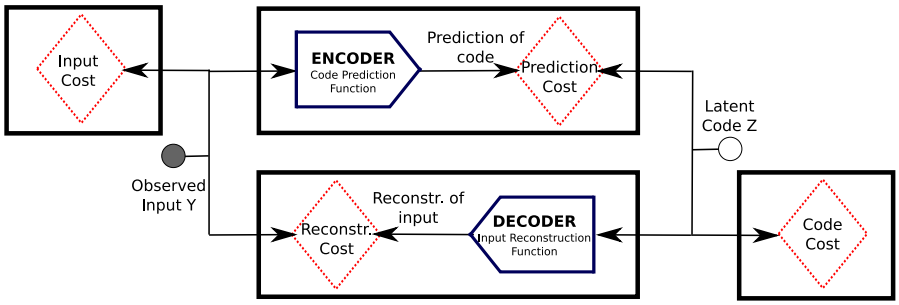


Figure 1.3: Generic unsupervised architecture in the energy-based model framework. Rectangular boxes represent factors containing at least a cost module (red diamond shaped boxes), and possibly, a transformation module (blue boxes). The encoder takes as input Y and produces a prediction of the latent code Z. The discrepancy between this prediction and the actual code Z is measured by the Prediction Cost module. Likewise, the latent code Z is the input to the decoder that tries to reconstruct the input Y. The discrepancy between this reconstruction and the actual Y is measured by the Reconstruction Cost module. Additional cost modules can be applied to the code and to the input. This is like a factor graph representation allowing to "zoom in" inside the nodes. The goal of *inference* is to determine the value of the latent code Z for a given input Y. The energy of the system is the sum of the terms produced by the cost modules. The goal of learning is to adjust the parameters of both Encoder and Decoder in order to make the energy lower in correspondence of the training samples, e.g., to make the predicted codes very close to the actual code Z, and to produce good reconstructions from Z when the input is similar to a training vector. After training, the encoder can be used for fast feed-forward feature extraction.

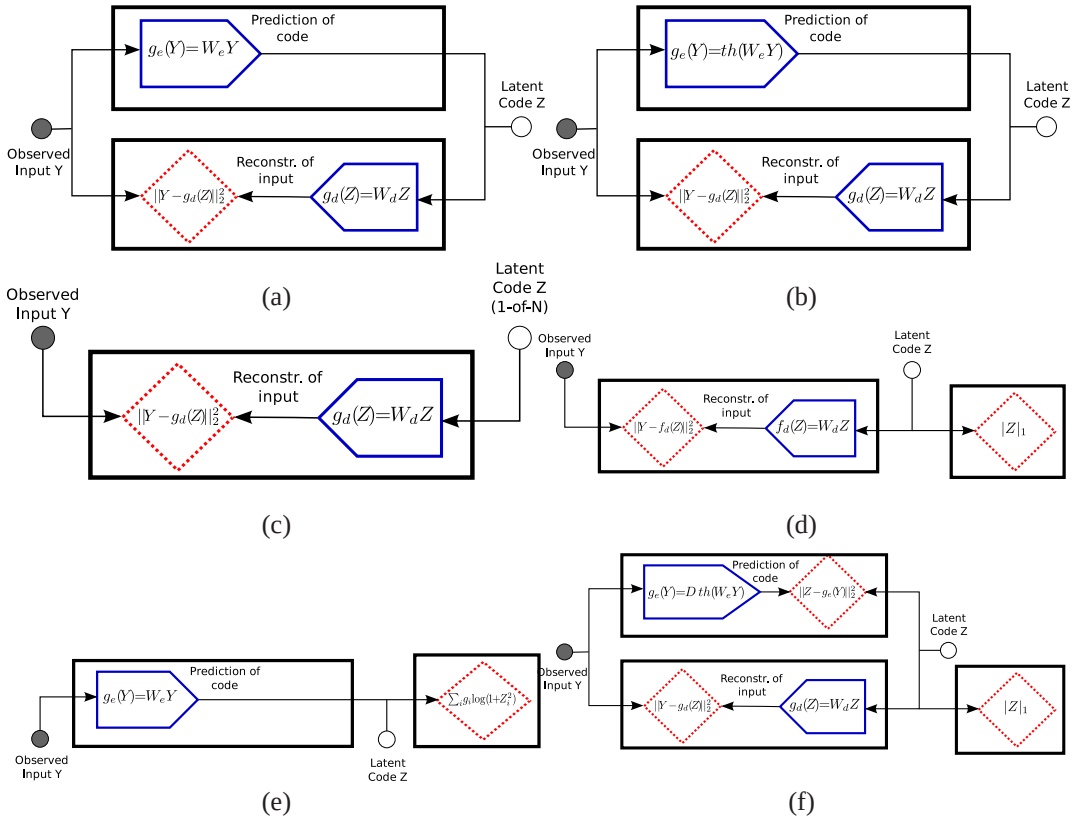


Figure 1.4: Instances of the graphical representation of fig. 1.3. (a) PCA the encoder and decoder are linear; (b) autoencoder neural network; (c) K-Means and other clustering algorithms: the code is constrained to be a binary vector with only one non-zero component; (d) sparse coding methods, including basis pursuit, Olshausen-Field models, and generative noisy ICA in which the decoder is linear and the code subject to a sparsity penalty; (e) encoder-only models, including Product of Experts and Field of Experts; (f) Predictive Sparse Decomposition method.

The energy-based graphical representation of an unsupervised model is derived from the graphical representation of a factor graph and it is shown in fig. 1.3. This is a

more operational representation where the transformations applied to the variables Y and Z, and the compatibility tests are made explicit and visible inside the factor nodes. In particular, there is a cost measuring the discrepancy between the code Z and its prediction given by the *encoder*. The encoder is a deterministic function mapping the input Y into an approximation of the latent code Z; this is denoted by $g_e(Y; W)$, where W are trainable parameters. Likewise, there is a cost measuring the discrepancy between the input Y and its reconstruction produced by the *decoder*. The decoder is another deterministic function that maps the latent code Z into a approximation of the input Y; this is denoted by $g_d(Z; W)$. Additional costs might take into account constraints applied to the input and latent variables as well. The overall energy of the system is the sum of all the terms produced by these cost modules.

Given a training set $T = \{Y^i, \quad i \in 1 \dots p\}$ and a set of trainable parameters W, we must define a parameterized family of energy functions $F(Y; W)$ in the form of an *architecture*, and a *loss functional* $L(F(\cdot; W), T)$ whose role is to measure the "quality" (or badness) of the energy surface $F(\cdot; W)$ on the training set T. An energy surface is "good" if it gives lower energies to areas around the training samples, and higher energies to all other areas.

Since the model depends not only on the input Y but also on the latent code Z, we must introduce another energy function $E(Y, Z; W)$ and an *inference* procedure to compute Z and $F(Y; W)$. With reference to fig. 1.3, $E(Y, Z; W)$ is the sum of the decoder reconstruction error, the encoder prediction error, and the error in satisfying the constraints on the input and latent code. In particular, we denote by $E_{\text{dec}}(Y, Z; W)$ and $E_{\text{enc}}(Y, Z; W)$ the error terms produced by the encoder and decoder's cost modules.

Before describing inference procedures, we first establish the link between energy-based models and probabilistic models. Among all possible distributions, we consider a *Boltzmann distribution* because it is the maximum entropy distribution satisfying an expected constraint on the average energy (Jaynes, 1957; Zhu et al., 1997). In other words, this is the distribution that makes less assumptions while being compatible with the observations. In a Boltzmann distribution the probability density function and the energy relate by:

$$P(Y, Z; W) = \frac{e^{-\beta E(Y,Z;W)}}{\Gamma_{Y,Z}}, \text{ with} \qquad (1.1)$$
$$\Gamma_{Y,Z} = \int_{y,z} e^{-\beta E(y,z;W)}, \quad \beta \in R^+$$

The denominator $\Gamma_{Y,Z}$ is called partition function and makes sure the distribution normalizes to one. Although energy-based models do not require $\Gamma_{Y,Z}$ to be finite in general (and therefore, there might be no probabilistic model that can be associated to an energy-based model), we assume $\Gamma_{Y,Z}$ finite when we refer to "the probabilistic model associated to" a given energy-based model. Note that any probabilistic model can be written in the energy-based model framework by defining $E(Y, Z; W) = -\log P(Y, Z; W)$. It is also useful to introduce the marginal distribution over the input Y:

$$P(Y; W) = \frac{e^{-\beta F(Y;W)}}{\Gamma_Y} = \frac{\int_z e^{-\beta E(Y,z;W)}}{\Gamma_{Y,Z}}, \text{ with} \qquad (1.2)$$
$$\Gamma_Y = \int_y e^{-\beta F(y;W)}$$

where $F(Y; W)$ is derived through marginalization, $F(Y; W) = -\frac{1}{\beta} \log \int_z e^{-\beta E(Y,z;W)}$.

The most common inference procedure defines Z and $F(Y)$ as follows:

$$Z = \arg\min_{z \in \mathcal{Z}} E(Y, z; W) \tag{1.3}$$

$$F(Y; W) = \min_{z \in \mathcal{Z}} E(Y, z; W) \tag{1.4}$$

In probabilistic terms, the proposed inference corresponds to finding the maximum a posteriori (MAP) estimate for the latent variables Z, that are treated as a *deterministic* latent variables. This is easy to show because $\arg\max P(Z|Y; W)$ is the same as $\arg\max P(Y, Z; W)$ which is equal to $\arg\min E(Y, Z; W)$ (see eq. 1.1). Instead, probabilistic models infer a *distribution* of latent codes by marginalizing the joint distribution of eq. 1.1. In terms of energies we have already seen that this corresponds to the following log sum of exponentials:

$$F(Y; W) = -1/\beta \log \int_z e^{-\beta E(Y,z;W)}, \beta \in \mathcal{R}^+ \tag{1.5}$$

which is intractable to compute, in general. Note that F can be interpreted as (the minimum of) the *free energy* from an analogy to statistical mechanics. For simplicity in this paper, we refer to both functions $E(Y, Z; W)$ and $F(Y; W)$ as "energy" since it will be clear from the context if we refer to one or the other. Also, note that if we let β go to infinity the log-sum in eq. 1.5 reduces to the minimization of eq. 1.4. Finally, some methods set the code through a deterministic mapping of the input. PCA and auto-encoder neural networks are the most popular example of machines using this kind of inference procedure. This limit case of inference procedure can also be seen as a particular instance of the minimization of eq. 1.4:

$$F(Y; W) = \min_{Z} E(Y, Z; W), \text{ with}$$

$$E(Y, Z; W) = \max_{\nu} E_{\text{dec}}(Y, Z; W) + \nu(Z - g_e(Y; W)) \tag{1.6}$$

where $E_{\text{dec}}(Y, Z; W)$ is the error term measuring the discrepancy between the output of the decoder and the input, $g_e(Y; W)$ is the value assigned to Z by the encoder, and ν is a Lagrange multiplier. While this is a dummy optimization problem setting $F(Y; W)$ equal to $E_{\text{dec}}(Y, g_e(Y); W)$, it directly links to the formulation of eq. 1.4.

Specializations of the model of fig. 1.3 include cases where either the encoder or the decoder are missing, as well as cases in which the code prediction error is constrained to be zero, i.e., where inference of the code is done through a deterministic mapping of the input. Fig. 1.4 and chapter 2 re-interpret several classical unsupervised methods in this framework, and elucidate this point. It is important to keep in mind that the general architecture of fig. 1.3 has several advantages over simpler architectures that lack either the encoder or the decoder. The decoder makes learning easier because it allows to check the fitting of the training data by comparing it with its reconstruction from the code. On the other hand, the encoder is trained to approximate the latent code Z allowing very fast and direct inference after its parameters are learned.

Devising an EBM consists of (1) choosing the architecture, i.e. the particular form of encoder, decoder and cost modules that will contribute to the energy function $E(Y, Z; W)$, (2) choosing an inference procedure that determines $F(Y; W)$ and Z, and (3) choosing a loss functional. A model can be trained with many different loss functionals. The simplest loss functional that one can devise, called the *energy loss*, is simply the average energy over the training set $T = \{Y^i, \ \ i \in 1 \dots p\}$:

$$L_{\text{energy}}(W, T) = \frac{1}{p} \sum_{i=1}^{p} F(Y^i; W) \tag{1.7}$$

In general, minimizing this loss *does not* produce good energy surfaces because, unless $F(Y; W)$ has a special form, nothing prevents the energy surface from becoming *flat*.

No term increases the loss if the energy of unobserved vectors is low, hence minimizing this loss will not ensure that the energies of unobserved vectors are higher than the energies of training vectors. This is undesired because it makes the model *unable to discriminate* between data vectors that are similar and data vectors that are dissimilar to training samples. An example of such learning failure is learning a set of random projections to simply rotate the input space, for instance. Clearly, all points in input space are perfectly reconstructed, even those that are very different from training samples, and the feature space is useless because it is just a rotation of the input space. To prevent this *catastrophic collapse*, we discuss two solutions. The first one is to add a contrastive term to the loss functional which has the effect of "pulling up" on the energies of selected unobserved points. The second solution, which is implicitly used by many classical unsupervised methods, is to construct the architecture in such a way that only a suitably small subset of the points can have lower energy. The region of lower energy can be designed to be a manifold with a given dimension, or a discrete set of regions around which the energy is lower. With such architectures, there is no need to explicitly pull up on the energies of unobserved points, since placing low energy areas near the training samples will automatically cause other areas to have higher energies.

1.2 Two Strategies to Avoid Flat Energy Surfaces

The trained model has to assign lower energy to vectors observed during training, and higher energy to unobserved vectors. This is achieved by designing a suitable energy and loss functional. All loss functionals have a term minimizing the energy over the training samples, while different strategies are employed to increase the energy of other

data vectors. First, we consider loss functionals that explicitly pull up on the energy of suitably chosen points. Then, we present a second family of energy functions that increase the energy of unobserved vectors indirectly by adding constraints to the code, and we demonstrate the equivalence of these two strategies.

1.2.1 Adding a Contrastive Term to the Loss

Learning to model the distribution of the input data, whether locally around the training samples or rather globally across the whole input space, can be achieved by minimizing a loss functional of the following form:

$$L(W;T) = \frac{1}{p}\sum_{i=1}^{p} f(F(Y^i;W)) - g(F(\bar{Y}^i;W)) \tag{1.8}$$

where Y^i is a training sample, $\bar{Y}^i$ is a data vector whose energy has to be increased, and f and g are monotonically increasing functions making sure that the energy of the training samples is lower than other points. The second term in the loss is called "contrastive term". Without this term the model could assign the same energy value to all points in input space.

An example of this loss functional is the so called *margin loss* (LeCun et al., 2006; Hadsell et al., 2006):

$$L(W,T) = \frac{1}{p}\sum_{i=1}^{p} F(Y^i;W)^2 + \max(0, m - F(\bar{Y}^i;W))^2 \tag{1.9}$$

where $m \in R^+$ is the margin. This loss tries to make the energy of the contrastive sample $\bar{Y}^i$ higher than the energy of the training sample by at least a margin m. Ideally, $\bar{Y}^i$ is chosen to be the "most offending incorrect answer" of the model (LeCun et al.,

2006). In other words, $\bar{Y}^i$ is the lowest energy point that lies outside a neighborhood containing the training data. If there is not prior knowledge about where such point can be picked, sampling methods such as Langevin dynamics can be used to select it.

Another example of this kind of loss functional can be derived by maximum likelihood learning in probabilistic models. Most probability densities can be written (or approximated as cloed as desired) in terms of an energy function through the Gibbs distribution:

$$P(Y^1, \ldots, Y^p; W) = \prod_{i=1}^{p} \frac{e^{-\beta F(Y^i;W)}}{\int_y e^{-\beta F(y;W)}} \tag{1.10}$$

where β is an arbitrary positive constant, and the denominator is the *partition function*. If a probability density is not explicitly derived from an energy function in this way, we simply define the energy as $F(Y;W) = -\log P(Y;W)$. Training a probabilistic density model is generally performed by finding the W that maximizes the likelihood of the training data under the model given in the previous equation. Equivalently, we can minimize a loss functional $L(W;T)$ that is proportional to the negative log probability of the data. Using the Gibbs expression for $P(Y;W)$, we obtain:

$$L(W;T) = -\frac{1}{\beta} \log P(Y^1, \ldots, Y^p; W) = \frac{1}{p} \sum_{i=1}^{p} F(Y^i;W) + \frac{1}{\beta} \log \int_y e^{-\beta F(y;W)} \tag{1.11}$$

Note that the same objective function can be derived using the dual formulation of the maximum entropy principle (Jaynes, 1957; Zhu et al., 1997). The gradient of $L(W,T)$ with respect to W is:

$$\begin{aligned} \frac{\partial L(W;T)}{\partial W} &= \frac{1}{p} \sum_{i=1}^{p} \frac{\partial F(Y^i;W)}{\partial W} - \int_y P(y;W) \frac{\partial F(y;W)}{\partial W} \\ &= < \frac{\partial F(Y;W)}{\partial W} >_{Y \sim T} - < \frac{\partial F(Y;W)}{\partial W} >_{Y \sim P(Y;W)} \end{aligned} \tag{1.12}$$

where $Y \sim T$ and $Y \sim P(Y;W)$ mean that Y is drawn from the training set and from the model distribution, respectively. In other words, minimizing the first term in eq. 1.11 with respect to W has the effect of making the energy of observed data points as small as possible, while minimizing the second term (the log partition function) has the effect of "pulling up" on the energy of unobserved data points to make it as high as possible, particularly if their energy is low (their probability under the model is high). Naturally, evaluating the derivative of the log partition function (the second term in eq. 1.12) may be intractable when Y is a high dimensional variable and $F(Y,W)$ is a complicated function for which the integral has no analytic solution. This is known as the *partition function problem*. A considerable amount of literature is devoted to this problem. The intractable integral is often evaluated through Monte-Carlo sampling methods, variational approximations, or dramatic shortcuts, such as Hinton's *contrastive divergence* method (Carreira-Perpignan and Hinton, 2005). The basic idea of contrastive divergence is to avoid pulling up on the energy of *every possible point* Y, and to merely pull up on the energy of randomly generated points located near the training samples. These points are found by using a Markov Chain that starts at training samples and that runs for only a few steps. This process is likely to pick low energy points that are nearby the training samples. This ensures that training points will become *local minima* of the energy surface, which is sufficient in many applications of unsupervised learning.

To summarize, we can interpret the log of the partition function as a very complicated instance of the contrastive term in eq. 1.8. This term increases the energy of all the points in input space making sure that the distribution normalizes to one. Even though the loss functional in eq. 1.11 is the only one maximizing the likelihood of the data, it is

generally intractable to compute and it requires approximations.

In general, one of the main issues when training unsupervised models is finding ways to prevent the system from producing flat energy surfaces. Probabilistic models explicitly pull up on the energies of unobserved points by using the partition function as a contrastive term in the loss. Other methods using a margin loss identify candidate points where the energy has to be pulled up by running an optimization to find a mode of the distribution, for instance. However, if the parameterization of the energy function makes the energy surface highly malleable or flexible, it may be necessary to pull up on a very large number of unobserved points to make the energy surface take a suitable shape. This problem is particularly difficult in high dimensional spaces where the volume of unobserved points is huge due to the curse of dimensionality.

1.2.2 Limiting the Information Content of the Internal Representation

One solution to the previously mentioned problem is to make the energy surface a little "stiff", so that pulling down on a small number of well-chosen points will automatically pull up the energies of many points (LeCun et al., 2006). One way to achieve this is by limiting the number of parameters or by constraining the parameters through a regularization term in the loss.

Another solution is to design the energy function in such a way that only a small subset of points can have low energies. This method is used (albeit implicitly) by prenormalized probabilistic models such as Gaussian models or Gaussian mixture models. In such models, only the points around the modes of the Gaussians can have low energy (or high

probability). Every other point has high energy by construction. It is important to note that this property is not exclusive to normalized density models. For example, a simple vector quantization model in which the energy is $F(Y;W) = \min_{i\in[1,...,N]} ||Y - W_i||^2$ where W_i is the i-th prototype, can only produce low energy values (good reconstructions) around each of the N prototypes and higher energies everywhere else.

Building on this idea, the encoder-decoder architecture has an interesting property that can be exploited to avoid flat energy surfaces. The architecture should be designed so that each training sample can be properly represented by a unique code, and therefore can be assigned a low energy value $F(Y;W)$ (e.g., good reconstruction). The architecture should also be designed so that unobserved points are assigned codes similar to those associated with training samples, so that their energies (e.g., reconstruction error) are higher. Satisfying this property can be done in a number of ways, but the simplest way is reduce the number of available codes while forcing them to represent well the training samples (by minimizing the energy over the training set). In short, we can minimize the simple energy loss in eq. 1.7 if we *limit the information content of the code*. This can be done by allowing the code to take only a finite number of different values (as with the example of the previous paragraph), or by making the code have a lower dimensionality than the input, or by having a term in the energy function $E(Y,Z;W)$ that forces the code to be a "sparse" vector in which most components are zero. Many classical unsupervised learning methods use this principle implicitly as described in the next chapter.

We provide a range of results that formalize the link between the information content of the code and the volume of data that can be assigned low energy in a number

of cases corresponding to different model assumptions, such as the kind of decoder and cost modules used. Lemma 1.1 is a set of general results that relate entropy to the shape of the energy function. Theorem 1.1 establishes a monotonic link between the entropy of the distribution of the input and the entropy of the distribution of the code in a linear generative model. Theorem 1.2 shows that in a sparse coding model, increasing the sparsity of the code decreases the volume of the input space with low energy. Theorem 1.3 shows that in a reconstructive dimensionality reduction model, reducing the dimensionality of the code decreases the volume of the input space with low energy.

Lemma 1.1. *Let us assume the energy $E(Y, Z)$ defines a joint probability distribution $P(Y, Z)$ from which we can derive conditional and marginal distributions. Let the marginal be $P(Y) = e^{-\beta F(Y)} / \int_y e^{-\beta F(y)}$ (omitting the parameters W for clarity of notation).*

(1) *The distribution maximizing the entropy over all probability densities on a given support S of finite volume is the uniform distribution (denoted by U).*

(2) *For any distribution $P(Y)$ defined on S, the KL divergence between $P(Y)$ and U increases linearly as the entropy of $P(Y)$ decreases.*

(3) *For any distribution $P(Y)$ defined on S with $H(P(Y))$ smaller than the maximum (log of the volume of S), $F(Y)$ cannot be constant.*

(4) *If Y is a vector distributed according to a Gaussian distribution, then decreasing $H(P(Y))$ makes $\det(\frac{\partial^2 F}{\partial Y^2})$ increase.*

(5) *Let $Y \in \mathcal{R}$ a random variable with distribution $P(Y)$, and consider a sequence of l variables drawn from $P(Y)$. A* high probability set *B_δ^l is defined as a set in $\mathcal{R}^l$ whose probability is greater than δ, with $\delta > 0.5$. The most probable set is the smallest of such*

sets. If the entropy of $P(Y)$ decreases, then the volume of the most probable set also decreases. ■

Proof.

(1) Let $H(P(Y))$ denote the entropy of a distribution $P(Y)$ and $H(U)$ the entropy of the uniform distribution.

$$\mathrm{KL}(P,U) = \int_S P(Y)\log(P(Y)Vol(S)) = -H(P(Y)) + H(U) \tag{1.13}$$

As the KL divergence of P and U is nonnegative with equality if and only if $P = U$,

$$H(P(Y)) \leq H(U), \tag{1.14}$$

with equality if and only if $P = U$.

(2) As seen above,

$$\mathrm{KL}(P,U) = H(U) - H(P(Y)) \tag{1.15}$$

(3) By defining the energy as minus the log of the probability, we have that the uniform distribution is the only one that is associated a flat energy surface $F(Y) = c, c \in \mathcal{R}$. Any other distribution with lower entropy, i.e. non-uniform, must have a non-flat energy surface. By contradiction, if $F(Y)$ is constant and not associated to a uniform distribution then we have that $P(Y) = \exp -\beta F(Y)/\int_y \exp -\beta F(y) = 1/Vol(S) = U$.

(4) The entropy of a random Gaussian vector with covariance matrix Σ is:

$$H(P(Y)) = \frac{1}{2}\log((2\pi e)^M |\det(\Sigma)|). \tag{1.16}$$

The entropy decreases iff $|\det(\Sigma)|$ decreases as well. On the other hand, if we define the energy as $F(Y) = -\log P(Y)$ (up to a constant) then we find that the Hessian of the energy is:

$$\frac{\partial^2 F}{\partial Y^2} = (|\det(\Sigma)|)^{-1}. \tag{1.17}$$

Hence, as the entropy of the distribution of the random vector is decreased the curvature of the energy surface increases. In particular, it increases at the mode of the distribution. If we think of $F(Y)$ as a trainable model, then decreasing the entropy makes the difference between the lowest and the highest energy values larger, and the model more discriminative (less uniform). A similar property can be demonstrated for any other uni-modal distribution as well.

(5) This is an interesting result from (Cover and Thomas, 1991). Let $Y \in \mathcal{R}$, and $\{Y^i\}_l$ a sequence of l independent samples drawn from the distribution $P(Y)$. Then, up to the first order in the exponent:

$$\text{Vol}(B_\delta^l) = 2^{lH(P(Y))} \tag{1.18}$$

where $\delta > 0.5$ and B_δ^l is the most probable set with probability at least δ estimated from l samples. In other words, the volume of the smallest set containing most of the probability is about $2^{lH(P(Y))}$ with (loosely speaking) average side length equal to $2^{H(P(Y))}$.

Although this results is valid only for (any) one-dimensional distribution, it gives a nice intuition of how the entropy relates to the energy in this case. Given a distribution and its corresponding energy, we can set a threshold to individuate the (possibly not connected) set with lowest energy (most probable set). By decreasing the entropy of the distribution, the volume of this set decreases meaning that the energy becomes more peaked (the distribution has more mass) around the lowest energy (most likely) areas. □

In the case of a linear generative model, links between the entropy of the input distribution and the entropy of the code distribution can be described more precisely:

Theorem 1.1. *In a linear generative (decoder-only) model that represents the input as $Y = WZ + \eta$, where $P(\eta) = N(0, \sigma_Y^2)$ and $P(Z) = L(\lambda)$ (Laplace distribution with zero mean and scale parameter λ), decreasing the entropy $H(P(Y))$ of the marginal dis-*

tribution over Y also decreases the average number of partitions of the input space that are induced by the code Z. Under a Gaussian variational approximation to the posterior and 0-th order Taylor approximations, decreasing $H(P(Z))$ reduces $H(P(Y))$ too.
■

Proof. This is derived from the Blahut-Arimoto and Information Bottleneck methods (Tishby et al., 1995). The model assumptions are the same as in (Olshausen and Field, 1997)'s algorithm. The code is distributed according to a Laplace prior $P(Z)$, and the conditional likelihood $p(Y|Z)$ is a Gaussian with mean WZ and fixed spherical covariance matrix. Hence, $H(P(Y|Z))$ is fixed, while $H(P(Y))$ depends on $H(P(Z))$ and the parameters W. The algorithm implicitly partitions the input space into (soft) regions, where each region is assigned a code Z. Since the average volume of the input space is $2^{H(P(Y))}$ and the average volume of all the points that are mapped into the same code Z is $2^{H(Y|Z)}$, the average number of partitions induced by the algorithm is given by the ratio: $2^{H(P(Y))}/2^{H(P(Y|Z))} = 2^{I(Y,Z)}$, where $I(Y,Z)$ is the mutual information between Y and Z. Therefore, by reducing $H(P(Y))$ we reduce the number of bits required to identify a partition, i.e. the complexity of the model. By using a Gaussian variational approximation to the posterior (refer to the appendix A) and exploiting the relation $H(P(Y)) = H(P(Z)) - H(P(Z|Y)) + H(P(Y|Z))$, we can show that reducing $H(P(Z))$ reduces $H(P(Y))$ since the entropy of the posterior never decreases more than $H(P(Z))$. □

In a sparse coding model, the sparsity of the code can be controlled through a hyperparameter; we show that varying this hyperparameter has an effect on the shape of the

energy surface:

Theorem 1.2. *In a sparse coding model whose energy is* $F(Y;W) = \min_Z ||Y - WZ||^2$ *s.t.* $||Z||_0 \leq d$*, with* $W \in \mathcal{R}^{M \times N}$*,* $M \leq N$*,* $d < M$*, and rank(W) =* $m \in (d, M]$*, and assuming the data centered at the origin and contained in a ball of radius* R*, and for a sufficiently small* $\epsilon \in \mathcal{R}^+$*, the volume of the input space whose energy is below* ϵ *decreases as the code is made sparser, that is, as* d *is decreased.* ■

Proof. The model assumptions are again the same as in (Olshausen and Field, 1997)'s algorithm, but expressed in the log domain in terms of energies and using the L_0 norm instead of the L_1 norm . The "L_0 norm" is not a norm and it counts the number of non-zero elements of a vector. By relaxing the L_0 norm into an L_1 norm we turn an NP-hard optimization problem into a convex one. The L_1 relaxation gives the same solution as the L_0 norm provided that the solution is sparse enough (Donoho and Elad, 2003). The proof is by induction. When $d = 0$ the only admissible code is 0, and the region that can have energy smaller than ϵ is the sphere of radius $\sqrt{\epsilon}$ centered at the origin. We denote the volume of this sphere with $S_0^\epsilon = \frac{4}{3}\pi\epsilon^{\frac{3}{2}}$. When $d = 1$, the set of admissible codes consists of all codes with at most one non-zero coordinate. This creates N additional sets along the columns of W with energy smaller than ϵ. For small ϵ, each such set can be approximated by a "tube" of length $2R$ and radius $\sqrt{\epsilon}$ going through the origin along the direction given by the columns of W. The total volume of these sets S_1^ϵ is lower bounded by the volume of a single "tube" which is $2R\pi\epsilon$. This volume is larger than S_0^ϵ provided ϵ is sufficiently small to guarantee that $\epsilon < \frac{9}{4}R^2$. The assumption on small ϵ is also required to approximate the low energy areas with the tube when $d = 1$ and to make sure that sub-spaces generated by the columns of W have little overlap when

$d > 1$. Assuming that the property holds for $d-1$, we are going to show that it also holds for $d < m$. This is trivial because the volume of the sets that have energy below ϵ using at most $d-1$ code units, S^{ϵ}_{d-1}, is strictly contained in S^{ϵ}_{d} since (a) the condition $||Z||_0 \leq d$ implies $||Z||_0 \leq d-1$, and (b) $d < m$. The former condition implies that $S^{\epsilon}_{d-1} \subseteq S^{\epsilon}_{d}$. The latter condition guarantees that the space spanned by at least one d-dimensional sub-space generated by picking d columns of W does not coincide with the sub-spaces generated by taking into account only $d-1$ columns since the rank of W is greater than d, and therefore, $S^{\epsilon}_{d-1} \subset S^{\epsilon}_{d}$. □

Reducing the dimensionality of the code is another way to control the shape of the energy surface:

Theorem 1.3. *In a model whose energy is $F(Y;W) = \min_Z ||Y - WZ||^2$, with $W \in \mathcal{R}^{M \times N}$, $N < M$, rank(W) = N, and assuming the data centered at the origin and contained in a ball of radius R, and for a sufficiently small $\epsilon \in \mathcal{R}^+$, the volume of the input space whose energy is below ϵ decreases as the code is made lower dimensional, that is, as N is decreased.* ■

Proof sketch. The proof is similar to the one of the previous theorem. The volume generated by any set of $N-1$ linearly independent vectors S^{ϵ}_{N-1} is going to be always smaller than the volume S^{ϵ}_{N} generated by taking into account N linearly independent vectors (columns of W), provided that ϵ is sufficiently small. □

So far we have taken into account "decoder-only" models that linearly reconstruct the input without a module directly producing the representation. As we mentioned earlier,

there are also “encoder-only” models (Teh et al., 2003; Bell and Sejnowski, 1995) that directly produce the representation without explicitly reconstructing the input. In particular, the work by (Scholkopf et al., 2001) addresses the question of how to devise an encoder-only algorithm that assigns lower energy to high data density areas. The algorithm is a single class SVM that maps the input into feature space through a fixed non-linear transformation followed by a linear adaptive projection. The algorithm is trained in such a way that this function is positive when points are drawn from high data density areas, and negative otherwise. This is essentially akin to learn an energy function and setting a threshold to identify regions of high data density. In order to train the parameters, the authors propose to minimize a loss functional that trades off two terms: the empirical error and the model complexity. The empirical error is formulated in terms of the separation of the points mapped in feature space from the origin, while the model complexity term is the squared L_2 norm of the parameter vector. Since the algorithm can be reduced to a standard binary SVM, it inherits many theoretical properties. In particular, the authors show that the trade-off parameter between the two error terms in the loss effectively controls the volume over which the algorithm assigns high probability and the fitting of the data.

The aim of their work is very similar to ours because (1) they want to learn a function that is above threshold over the most probable set (lower energy around training samples) without trying to solve the more difficult density estimation problem tackled by probabilistic models, and (2) they use a loss function that trades off the fitting of the training data to the model complexity. While they control the model complexity by penalizing the parameters, we suggest to control it by constraining the internal represen-

tation. Theorem 1.1 used yet another way to control the model complexity, i.e. by minimizing the mutual information between input and code. While quantities like entropy and mutual information are difficult to compute and estimate, we argue that constraining the representation might be better than bounding the parameters of the model if we are interested in using the unsupervised algorithm to learn a representation of the input. The framework we propose is particularly useful not just to identify high data density areas, but also to learn adaptive representations of the input with some properties, as opposed to use fixed non-linear mappings.

To summarize, we have shown that the energy surface can be made less flat by either having a contrastive term in the loss pulling up on the energy of suitably chosen data vectors, or by constraining the code. The first strategy might become inefficient in high dimensional spaces because the volume that has to be considered is very large, and because sampling and optimization methods become too expensive. Constraining the internal representation is a more global strategy to pull up on the energy. Only few regions in input space can be assigned low energy values. Since the energy is made small around the training samples, it must be higher elsewhere. This strategy might overcome the inefficiency of the first class of methods that use a contrastive term in the loss, and it can actually be used in combination with it (Lee et al., 2007).

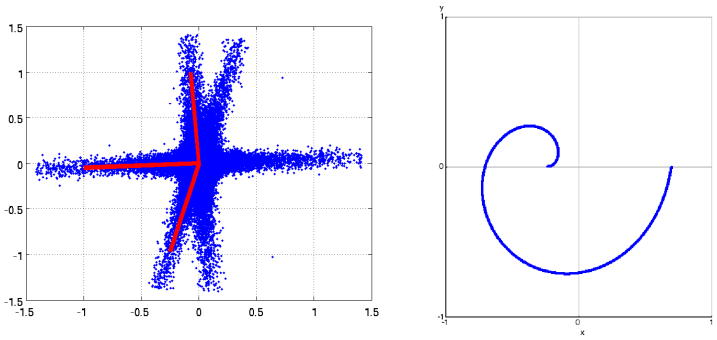
2

CLASSICAL METHODS IN THE LIGHT OF THE ENERGY-BASED MODEL FRAMEWORK

In this chapter we review a number of classical unsupervised learning models in the light of the energy-based framework. The most popular methods and PSD, the algorithm that we are going to introduce in chapter. 3, are summarized in table 2.1. Most of these algorithms use special cases of the encoder-decoder architecture described above. They differ in the specifics of the architecture, the constraints on the code, the inference procedure and the loss function used for training. To illustrate how each method works, we will use toy problems, designed to facilitate the visualization of the energy surface.

The first training dataset is shown in fig. 2.1(a) and it consists of 10,000 points in the 2D plane (y_1, y_2) . The training points are generated by a mixture of three Cauchy distributions along three random vectors. Points outside the circle of radius $\sqrt{2}$ have been left out. The second training dataset is shown in fig. 2.1(b) and it also consists of 10,000 points in the 2D plane, but the points are generated along a spiral that fits in the square with opposite corners (-1,1), (1,-1). The goal of learning is to learn an energy surface $F(Y; W)$ with lower values around regions of high data density and higher values everywhere else. The energy can be as simple as the squared reconstruction error, or it can include additional terms such as a sparsity constraint. The data is designed so that there

is no function that can predict a single value of y_2 from y_1 or vice versa. It is important to remain cautious about over-interpreting results obtained with this low-dimensional toy problem: real problems in unsupervised learning occur in high dimensional tasks. Nevertheless, these toy examples are a useful didactic tool.



(a) (b)

Figure 2.1: Toy datasets: 10,000 points generated by (a) a mixture of 3 Cauchy distributions (the red vectors show the directions of generation), and (b) points drawn from a spiral.

2.1 Principal Component Analysis

PCA is an encoder-decoder architecture that minimizes an energy loss equal to the mean squared reconstruction error. In PCA the optimal code is constrained to be equal to the value predicted by the encoder, and the encoder and decoder are symmetric and use a weight matrix W whose columns are orthogonal; see table 2.1 for more details. PCA avoids flat energy surfaces by using a code with a lower dimensionality than the input. Only those vectors Y that are in the space spanned by the columns of W will

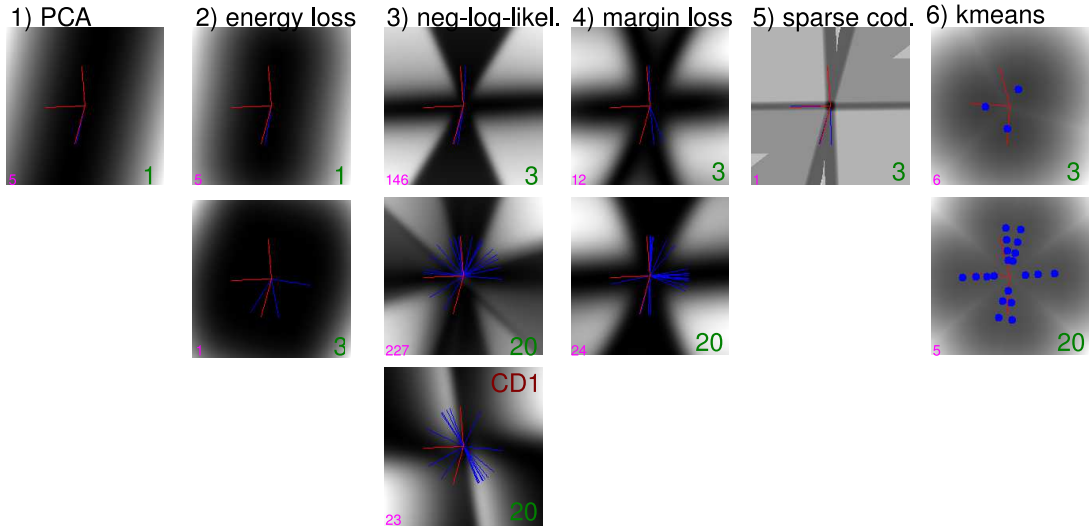


Figure 2.2: Toy dataset (a) - Energy surface $F(Y;W)$ for (by column): 1) PCA, 2) auto-encoder trained using the energy loss (minimization of mean squared reconstruction error), 3) auto-encoder trained using as loss the negative of the log-likelihood, 4) auto-encoder trained by using the margin loss, 5) a sparse coding algorithm (Lee et al., 2006), and 6) K-Means. The red vectors are the vectors along which the data was generated (mixture of Cauchy distributions). The blue lines are the directions learned by the decoder, the magenta numbers on the bottom left are the largest values of the energy (the smallest is zero), and the green numbers on the bottom right are the number of code units. Black is small and white is large energy value.

be exactly reconstructed (with zero energy). Therefore, learning can be carried out by simply minimizing the average energy of the training samples, without having to worry about pulling up on the energies of unobserved points: their energies will automatically become higher (except if they happen to be on the linear subspace).

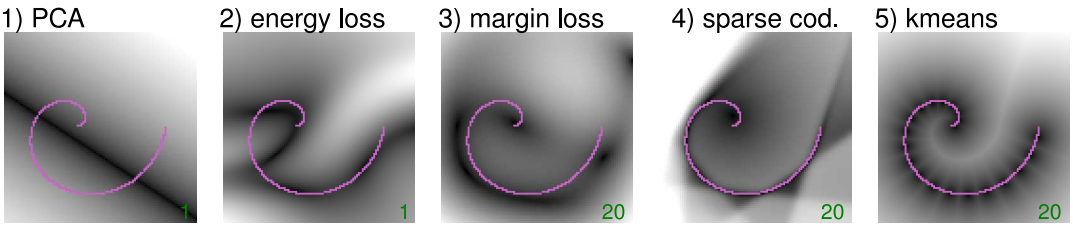


Figure 2.3: Toy dataset (b) - Energy surfaces. The magenta points are training samples along which the energy surface should take smaller values.

Table 2.1: Popular unsupervised algorithms in the EBM framework of fig. 1.3. N and M are the dimensionalities of the code Z and the input Y, σ is the logistic non-linearity and σ' its derivative, and g_e is the mapping produced by the encoder. In the Mixture of Gaussians (MoG) we denote the inverse of the covariance matrix of the i-th component with A_i, for $i = 1..N$.

Methods	Encoder	Decoder	Inp. Rec. Cost	Code Pred. Cost	Code Cost	pull-up
PCA	$W^T Y$	WZ	$\|\|Y - WZ\|\|_2^2$	$Z = g_e(Y; W)$	–	$N < M$
autoenc.	$\sigma(W_e Y)$	$W_d Z$	$\|\|Y - W_d Z\|\|_2^2$	$Z = g_e(Y; W_e)$	–	$N < M$
RBM	$\sigma(W^T Y)$	$\sigma(WZ)$	$-\frac{1}{2} Y^T WZ$	$-\frac{1}{2} Y^T WZ$	–	part. func.
ICA-IM	$W_e Y$	–	–	$Z = g_e(Y; W_e)$	$\sigma'(Z)$	$\log \|W_e\|$
sparse cod.	–	WZ	$\|\|Y - WZ\|\|_2^2$	–	$\|\|Z\|\|_1$	sparsity
PSD	$D\sigma(W_e Y)$	$W_d Z$	$\|\|Y - W_d Z\|\|_2^2$	$\|\|Z - D\sigma(W_e Y)\|\|_2^2$	$\|\|Z\|\|_1$	sparsity
K-Means	–	W_i	$\|\|Y - W_i\|\|_2^2$	–	–	1-of-N code
MoG	–	W_i	$\|\|Y - W_i\|\|_{A_i}^2$	–	–	part. func.
PoE	$W_e Y$	–	–	$Z = g_e(Y; W_e)$	$\sum g_i \log(1 + z_i^2)$	part. func.

PCA performs very poorly on these toy datasets. With one code unit, the region of low energy is a horizontal straight line as shown in fig. 2.2 1) and fig. 2.3 1), while with two code units, the entire input space has zero energy. Every point in the plane gets reconstructed perfectly (the system simply computes the identity function), but the

Table 2.2: Strategies used by common algorithms to avoid flat energies.

Pull-up factor	methods
$N < M$	PCA, autoencoder, factor analysis
sparsity	sparse coding, K-Means
exact max. likelihood (part. func.)	square ICA, MoG
approx. max. likelihood (CD, score matching, etc.)	RBM, PoE, overcomplete ICA
constant part. func.	Basis Rotation (Weiss and Freeman, 2007)
param. regularization	1 class SVM, KPCA

model is essentially useless as it does not discriminate between areas of high and low data density.

2.2 Autoencoder

Similarly to PCA, an autoencoder neural net with a small hidden layer learns low-dimensional representations. Unlike with PCA, that manifold may be non-linear if the encoder and decoder have multiple layers. Still, the limitation on the dimensionality of the code allows us to simply pull down on the energy of the training samples, without having to pull up on unobserved points. Fig. 2.2 2) shows the energy surface (squared reconstruction error) generated by using a one-hidden layer autoencoder with one and three code units (see table 2.1). While the former one produces an energy surface very similar to PCA, the latter one fails to produce a non-flat energy[1] because neither a contrastive term is added to the loss nor the code is constrained. Fig. 2.3 2) shows the energy surface on the spiral data when the encoder has a first hidden layer with 100 units and a second

[1]The energy is almost flat because of the saturation due to the hyperbolic tangent.

hidden layer with just one unit (the code). Similarly, the decoder has a hidden layer with 100 units and an output layer with two units. The lower dimensional manifold of zero energy is non-linear, but does not fit the data perfectly.

2.3 Negative Log Probability Loss

In order to train a "wide" autoencoder successfully, i.e. an autoencoder whose code is higher dimensional than the input, we can add a contrastive term to the loss. Here we consider a probabilistic approach and minimize the negative log likelihood loss. The energy of the system is again the squared reconstruction error between input and output of the network:

$$F(Y; W_e, W_d) = ||Y - W_d \text{tanh}(W_e Y)||_2^2$$

Naturally, there is no analytic expression for the log partition function, since it involves a non-linear function in the exponential, $\log \int_y \exp(-F(y; W_e, W_d))$. Then, we must resort to approximate methods to evaluate the derivative of the log partition function as shown in eq 1.12. Since the toy problem is two-dimensional, we can get away with replacing the integral by a discrete sum of about 10,000 data vectors at regularly-spaced grid points. Fig. 2.2 3) shows the energy surface produced by such a wide autoencoder with only 3 and 20 hidden units. The bottom plot shows the result of approximate maximum likelihood using Contrastive Divergence (with persistent Markov Chain sampling (Salakhutdinov, 2008)) in order to approximate a sample from the model distribution. Notice how in this case the energy surface takes the correct shape only locally around regions of high density of training data. Interestingly, the model *warps* the input

space in order to produce such energy surface. For instance, when the autoencoder has only 3 code units the directions learned by the decoder (blue vectors) are not the directions along which the data was generated (red vectors), and the encoder and decoder vectors are not pointing along the same directions. The machine assigns lower energy to the training samples, and much higher energy to the other points by properly rotating and stretching the mapping. A solution with symmetrical encoder and decoder where the vectors recover the directions of data generation would allow good reconstruction of all points in the plane, instead.

An important characteristics of the negative log probability loss is that its value only depends on the *difference* of energies between training points and unobserved points. Shifting the entire surface upwards, i.e. adding a constant to the energy, does not change the value of the loss. This means that there is no pressure for training points to produce low energy values. Since the energy is equal to the squared reconstruction error, an important consequence of this fact is that *minimizing the negative log probability loss does not necessarily lead to good reconstruction of training samples*. In general, the negative log probability loss is the only one maximizing the likelihood of the data. This is the loss that should be used for density estimation problems that might arise in compression applications, for instance. However, most of the times we only need to identify regions of high density data or to extract features. Unless the probabilistic model is simple and the input space is low dimensional, training using this loss might require expensive approximations, such as the use of variational and sampling methods.

2.4 Restricted Boltzmann Machines

In a Restricted Boltzmann Machine (Hinton, 2002), the code Z is binary and it can have any dimensionality. When the input Y is binary as well, we have that the encoder prediction error is the same as the input reconstruction error, and they are equal to $-\frac{1}{2}Z^TW^TY$. The overall energy (disregarding the biases) is: $E(Y, Z; W) = -Z^TW^TY$. There is no error associated to the code variables. Inference of the code is not performed by minimizing the energy, but by sampling the code according to the distribution defined by the energy. Once Z has been picked, the reconstruction is chosen by sampling as well. When Y is continuous, the code is still chosen by sampling, but the reconstruction is set equal to $\sigma(WZ)$, where $\sigma(.)$ is logistic. This corresponds to taking the average over the distribution of binary vectors Y. Weights are updated according to contrastive divergence (Carreira-Perpignan and Hinton, 2005), an approximation to the gradient of the log-likelihood of the data.

2.5 Product of Experts

In the Product of Experts (PoE) method proposed by (Teh et al., 2003) the decoding is missing and the code is a deterministic function of the input. The encoder consists of a set of linear filters, rows of matrix W_e, and the energy is defined as: $F(Y) = \sum g_i \log(1 + z_i^2)$, with $Z = W_eY$ and $g_i, i \in [1, .., N]$ set of coefficients that are subject to learning as well. Training uses the negative log probability loss with a gradient step approximated by contrastive divergence. While producing the code for a given

input is a simple matrix multiplication, generating data from the model (which is necessary in the energy pull-up phase of contrastive divergence learning) requires the use of expensive sampling methods, such as Markov Chain Monte Carlo and Hybrid Monte Carlo methods (Neal, 1993), whose convergence is slow and difficult to assess. More recently, Salakhutdinov (Salakhutdinov, 2008) introduced a faster variant of these methods dubbed "*persistent Markov Chains*". The basic idea is to maintain a set of "particles" (input vectors) to compute the model expectations, and to update these particles by using Gibbs sampling. Sampling methods are used whenever the learning algorithm needs to compute expectations over the model distribution or to draw samples from it, like in contrastive divergence learning.

2.6 Contrastive Margin Loss

As with contrastive divergence, we concentrate our effort on pulling up the energies of unobserved points that are in the *vicinity* of training samples. To generate one of those points (let's call it $\bar{Y}$), we use two strategies: (a) use Langevin dynamics, and (b) sample a point at random from a region that lies outside a small neighborhood of the training samples. In the first case, we start from the training sample and run a few steps of a Langevin dynamics by updating the current estimate of $\bar{Y}$ in the following way:

$$\bar{Y} \leftarrow \bar{Y} - \eta \frac{\partial E}{\partial Y}|_{Y=\bar{Y}} + \epsilon \tag{2.1}$$

where ϵ is a random sample from a zero-mean multivariate Gaussian distribution with a predefined isotropic covariance, and η is a step size. The process is biased towards picking unobserved points with low energy. Unlike contrastive divergence, we use a

contrastive loss function that does not attempt to approximate the negative log probability loss, but merely attempts to pull up on samples $\bar{Y}$ up to a given energy level m, called the *margin*:

$$L(Y;W) = F(Y;W) + \max(0, m - F(\bar{Y};W)) \tag{2.2}$$

For this experiment, we set $m = 1$. The energy surface is the squared reconstruction error and again we consider a wide autoencoder. The result using the sampling strategy (b) is shown in figure 2.2 4), while the energy surface obtained using the sampling strategy (a) is shown in fig. 2.3 3). The contrastive term prevents the energy from being flat everywhere. The margin prevents unobserved points whose energy is already high from being pushed even further.

2.7 Sparse codes

In this sparse coding model (Lee et al., 2006; Olshausen and Field, 1997), the code has most of its components constrained to be zero. The architecture does not have an encoder. The error on the code measures the L_1 norm of Z, while the reconstruction error of the input measures the squared Euclidean distance between input and linear reconstruction from the code. Since Z can be higher dimensional than the input, an energy loss equal to the squared reconstruction error would produce an energy surface equal to 0 everywhere (the linear system is underdetermined). Instead, the code is constrained to be sparse and only few low dimensional subspaces can have low energy. Loss and the

energies are equal to:

$$E(Y, Z; W) = \frac{1}{2}||Y - WZ||_2^2 + \lambda||Z||_1$$
$$L(Y; W) = F(Y; W) = min_Z E(Y, Z; W) \tag{2.3}$$

The model is also summarized in table 2.1. Inference is performed by minimizing the convex non-quadratic energy $E(Y, Z; W)$ over Z. By applying the algorithm to the first toy dataset, a staircase energy surface can be retrieved and the minima are tightly set around the training samples as shown in fig. 2.2 5) (the directions of data generation are actually perfectly recovered). We have also applied another sparse coding algorithm (Ranzato et al., 2006) to the spiral toy dataset and got similar results as shown in fig. 2.3 4).

2.8 K-Means Clustering

The architecture for K-means clustering is very similar to the one described for sparse coding. Indeed, the coding produced by K-Means can be interpreted as the ultimate sparse code because only one unit is active (non-zero) in the code. K-Means has no encoder, only a decoder and a reconstruction cost module, see table 2.1. The only points that are reconstructed with zero energy are the prototypes. Every other point has higher energy.

Figure 2.2 6) shows the energy surface obtained by training K-means with 3 and 20 prototypes on the first toy dataset, and fig. 2.3 5) shows the same on the spiral data with 20 prototypes. The minima of the energy are very localized, and many more prototypes would have been necessary to give the right shape to the energy surface. The problem

becomes even more serious in high dimensional spaces.

2.9 Mixture of Gaussians

The energy-based model view of a mixture of Gaussians and K-Means are very similar. A mixture of Gaussians is a "decoder-only" model lacking an encoder. Since the joint probability is $P(Y, Z = i; W) = P(Y|Z = i)P(Z = i)$, with $P(Y|Z = i) = N(W_i, A_i^{-1})$, where W_i is the mean of the i-th component, A_i is its inverse covariance matrix, and W denotes all the parameters of the model (means and covariances), the energy is: $E(Y, Z; W) = \sum_i \delta(Z, i)(\frac{1}{2}||Y - W_i||^2_{A_i} - \log\det(A_i) - \log P(Z = i))$. We can also introduce a constant $c_i = -\log\det(A_i) - \log P(Z = i)$ to define $E(Y, Z; W) = \sum_i \delta(Z, i)(\frac{1}{2}||Y - W_i||^2_{A_i} + c_i)$. If $P(Z = i)$ and the covariances are the same for all components (i.e., $c_i = c, \forall i$), then this is much like the squared reconstruction error used in K-Means. However, inference and learning are different. In a mixture of Gaussians inference is done through marginalization, $P(Y; W) = \sum_i P(Y|Z = i)P(Z = i)$, which corresponds to setting $F(Y; W) = -\log \sum_Z e^{-E(Y,Z;W)}$, and the loss is the negative log of the data likelihood. Training a mixture of Gaussians in the energy-based model framework would proceed by minimizing the loss by gradient descent over the training set. After training, the constant c_i would converge to minus the sum of the log of $P(Z = i)$ and the log determinant of A_i. Usually, a mixture of Gaussian is trained using EM algorithm instead (Dempster et al., 1977).

2.10 Other Algorithms

There are other popular unsupervised algorithms that can be interpreted in the energy-based model framework. In Factor Analysis (Hinton et al., 1997) the internal representation is modelled by a multivariate isotropic Gaussian distribution and the conditional distribution of the input given the code is another Gaussian distribution with a diagonal covariance matrix. Factor analysis is a linear generative model whose decoder matrix is learned by maximizing the data likelihood. The information content of the code is reduced by the choice of the prior on the code (penalizing its L2 norm), and by using a code with smaller dimensionality than the input.

Minimum Description Length (Hinton and Zemel, 1994) method minimizes a loss that takes into account the number of bits that are needed to encode the reconstruction error, the number of bits required to represent the code and the number of bits to encode the parameters of the model. These last two terms limit the information content of the code and the complexity of the model, ensuring that the energy is not constant.

(Doi et al., 2006) proposed an encoder-decoder model to learn sparse overcomplete representations. Their model prevents flat energy surfaces by injecting noise in the representation, yet another way to limit the information content of the code.

Finally, Score Matching (Hyvarinen, 2005) makes sure that training samples are local minima of the energy surface by minimizing a loss functional that is the difference between the square of the first derivative and the second derivative of the energy with respect to the input training data. In other words, training samples are minima of the energy surface and they are placed in regions of high curvature, thus preventing flat

energy surfaces. (Hyvarinen, 2007) also showed that score matching is a deterministic variant of Contrastive Divergence using Langevin dynamics for sampling.

2.11 What is not an EBM?

The EBM framework can be used to describe most unspervised learning algorithms. However, there are some methods that are based on other principles than learning an energy surface with training samples placed at the local minima. One such method is called *denoising autoencoder* (LeCun, 1987; Ranzato et al., 2007a; Vincent et al., 2008). These autoencoders are trained using as input a noisy data vector (e.g., an image patch corrupted by additive Gaussian noise), and as target the corresponding "clean" version of the input. The machine rather than learning to place the training samples at the minima of the energy surface, it learns *vector fields* whose rotational is not necessarily zero. Hence, these vector fields may not be derived or underlie any potential (or energy).

3

LEARNING SPARSE FEATURES

In the previous chapters we have seen that restricting the information content of the code is an efficient strategy to prevent flat energy surfaces in high dimensional spaces. A sparse coding algorithm represents the input data with a code that has only few significantly non-zero components. Our interest in sparse coding algorithms is justified by their computational efficiency, and by their ability to learn a possibly overcomplete set of features. In such high-dimensional spaces features are more likely to become linearly separable, making simpler a recognition system based on these representations.

Here we show how the energy-based model framework can be used to devise an efficient sparse coding algorithm. This algorithm is called *Predictive Sparse Decomposition* (PSD). First, we decide on the architecture. We are interested in systems that produce features through a (1) direct and (2) non-linear mapping. A direct mapping, as opposed to an iterative inference procedure, is desirable for its computational efficiency. The mapping has to be non-linear because linear projections can capture at most second order correlations. Moreover, a linear mapping would not be able to produce sparse overcomplete representations because of the non-orthogonality of the filters. Therefore, we use an architecture like in fig. 3.1 that has an encoder implementing a non-linear function such as:

$$g_e(Y; W_e, D) = D \tanh(W_e Y) \tag{3.1}$$

where $Y \in \mathcal{R}^M$ is the input, $W_e \in \mathcal{R}^{N \times M}$ is a filter matrix, $\tanh$ is the hyperbolic tangent non-linearity, and $D \in \mathcal{R}^{N \times N}$ is a diagonal matrix of coefficients. Other encoding

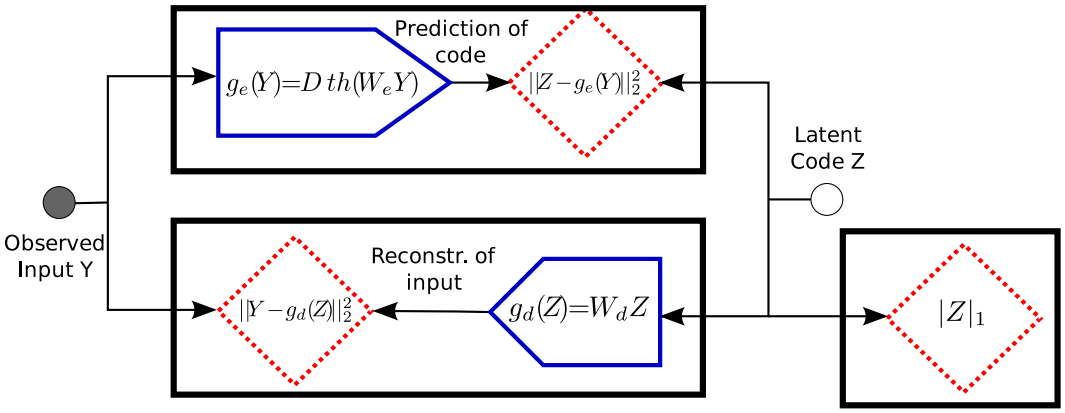


Figure 3.1: Graphical representation of PSD algorithm learning sparse representations.

functions will be studied in the appendix B. Since the encoder is already non-linear, the decoder can be linear and parameterized by a set of basis functions, columns of matrix $W_d \in \mathcal{R}^{M\times N}$. The most natural encoding and decoding cost modules simply compute the squared distance of their inputs. Since we are interested in sparse codes, we also enforce an L1 penalty on the code. We define the energy as the sum of all these error terms, and the loss can be safely set equal to the energy since the code is constrained to be sparse:

$$E(Y,Z;W_d,W_e,D) = \|Y - W_dZ\|_2^2 + \alpha\|Z - g_e(Y;W_e,D)\|_2^2 + \lambda\|Z\|_1$$
$$L(Y;W_d,W_e,D) = F(Y;W_d,W_e,D) = \min_Z E(Y,Z;W_d,W_e,D) \qquad (3.2)$$

3.1 Inference

Inferring the representation Z can be done in two ways.

- **Optimal inference** consists of setting the representation to

$$Z^* = \arg\min_z E(Y, Z; W_d, W_e, D)$$

, by running an iterative gradient descent algorithm involving two possibly large matrix-vector multiplications at each iteration (one for computing the value of the objective, and one for computing the derivatives through W_d). Note that the loss is convex with respect to Z, and other faster optimization algorithms could be used (Lee et al., 2006).

- **Approximate inference** sets the representation to $g_e(Y; W_e, D)$ as given in eq. 3.1, involving only a forward propagation through the encoder, i.e. a single matrix-vector multiplication.

The optimal inference is used during training, while the approximate inference is used after training when the encoder is able to produce values very close to the optimal ones.

3.2 Learning

The goal of learning is to find the optimal value of the parameters in both encoder and decoder: $\{W_e, D, W_d\}$. Learning proceeds by an on-line block coordinate gradient descent algorithm, alternating the following two steps for each training sample Y:

1. *(optimal inference step)* keeping the parameters fixed, minimize the energy $E(Y, Z; W_d, W_e, D)$ of eq. 3.2 with respect to Z, starting from the initial value provided by the encoder $g_e(Y; W_e, D)$.

2. *(parameter update step)* using the optimal value of the code Z found at the previous step, update the parameters by one step of stochastic gradient descent on the loss. The update is: $U \leftarrow U - \eta \frac{\partial \mathcal{L}}{\partial U}$, where U collectively denotes $\{W_e, D, W_d\}$ and η is the step size. The columns of W_d are then re-scaled to unit norm.

Minimizing this energy with respect to Z produces a representation that simultaneously reconstructs the input, is sparse, and is not too different from the predicted representation. If multiple solutions to the original loss (without the code prediction term) exist, minimizing this compound loss will drive the system towards producing basis functions and optimal representations that are easily predictable. After training, the function $g_e(Y; W_e, D)$ will provide good and smooth approximations to the optimal sparse representations. Also, the columns of W_d are rescaled to unit norm after every update because otherwise the loss can be made smaller by multiplying and dividing W_d and Z by the same constant. Since W_d is normalized and the $\tanh$ saturates, the encoder needs the trainable diagonal matrix D of coefficients in order to make the system able to adapt to different scaling of the input data.

Interestingly, we recover different algorithms depending on the value of the hyperparameter α of eq. 3.2:

- $\alpha = 0$. The loss of eq. 3.2 reduces to the one in eq. 2.3. The learning algorithm becomes similar to (Olshausen and Field, 1997)'s sparse coding algorithm. The encoder can be trained *separately* from the set of basis functions W_d.

- $\alpha \in (0, +\infty)$. The parameters are updated taking into account also the predictability constraint on the representation.

- $\alpha \rightarrow +\infty$. The additional constraint on the representation (the second term in eq. 3.2) becomes an equality, i.e. $Z = g_e(Y; W_e, D)$, and the model becomes similar to an auto-encoder neural network with a sparsity regularization term acting on the internal representation Z instead of a regularization acting on the parameters W_e and W_d.

In this paper, we always set $\alpha = 1$. Sec. 3.3.2 shows that training the encoder after training the set of bases W_d yields similar performance in terms of recognition accuracy. When the encoder is trained afterwards, the approximate representation is usually less sparse and the overall training time increases considerably. Finally, additional experiments (see appendix B) show that training the system as an auto-encoder ($\alpha \rightarrow +\infty$) provides a very fast and efficient algorithm that can produce good representations when the dimensionality of the representation is not much greater than the input dimensionality, i.e. $N \simeq M$. When the sparse representation is highly overcomplete the block-coordinate descent algorithm with $\alpha \in (0, +\infty)$ provides better features.
In the appendix A we give yet another interpretation of the encoder, as an approximation to the mean parameter of a Gaussian variational approximation to the true posterior distribution.

3.3 Experiments

First, we demonstrate that the proposed algorithm (PSD) is able to produce good features for recognition by comparing to other unsupervised feature extraction algorithms, principal components analysis (PCA), restricted Boltzmann machine (RBM) (Hinton,

2002), and sparse encoding symmetric machine (SESM) (Ranzato et al., 2007b). Then, we compare the recognition accuracy and inference time of PSD feed-forward approximation to feature sign (Lee et al., 2006), the fastest exact sparse coding algorithm, on the Caltech 101 dataset (Fei-Fei et al., 2004).

3.3.1 Comparing PSD to PCA, RBM, and SESM

The MNIST dataset (MNI,) has a training set with 60,000 handwritten digits of size 28x28 pixels, and a test set with 10,000 digits. Each image is preprocessed by normalizing the pixel values so that their standard deviation is equal to 1. In this experiment the sparse representation has 256 units. This internal representation is used as a global feature vector and fed to a linear regularized logistic regression classifier. Fig. 3.3.1 shows the comparison between PSD (using feed-forward approximate codes) and, PCA, SESM (Ranzato et al., 2007b), and RBM (Hinton, 2002). Even though PSD provides the **worst reconstruction error**, it can achieve the **best recognition accuracy** on the test set under different number of training samples per class.

3.3.2 Comparing PSD to Exact Sparse Coding Algorithms

In order to quantify how well our jointly trained encoder approximates the optimal representations obtained by minimizing the loss in eq. 3.2 and the optimal representations that are produced by an exact algorithm minimizing eq. 2.3 such as feature sign (Lee et al., 2006) (FS), we measure the average signal to noise ratio[1] (SNR) over a test dataset of 20,000 natural image patches of size 9x9. The dataset of images was constructed by

[1] $SNR = 10log_{10}(\sigma^2_{signal}/\sigma^2_{noise})$

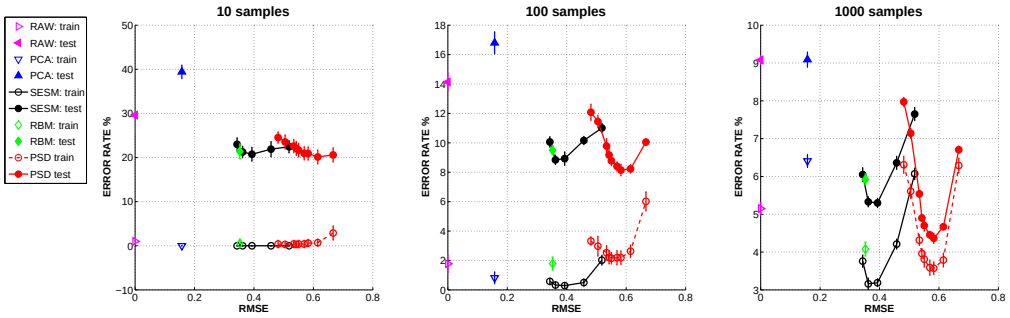


Figure 3.2: Classification error on MNIST as a function of reconstruction error using raw pixel values and, PCA, RBM, SESM and PSD features. Left-to-Right : 10-100-1000 samples per class are used for training a linear classifier on the features. The unsupervised algorithms were trained on the first 20,000 training samples.

randomly picking 9x9 patches from the images of the Berkeley dataset converted to gray-scale values, and these patches were normalized to have zero mean and unit standard deviation. The algorithms were trained to learn sparse codes with 64 units[2].

We compare representations obtained by "PSD Encoder" using the *approximate* inference, "PSD Optimal" using the *optimal* inference, "FS" minimizing eq. 2.3 with (Lee et al., 2006), and "Regressor" that is separately trained to approximate the exact optimal codes produced by FS. The results given in table 3.1 show that the PSD encoder achieves about the same SNR on the true optimal sparse representations produced by FS, as the Regressor that was trained to predict these representations.

Despite the lack of absolute precision in predicting the exact optimal sparse codes,

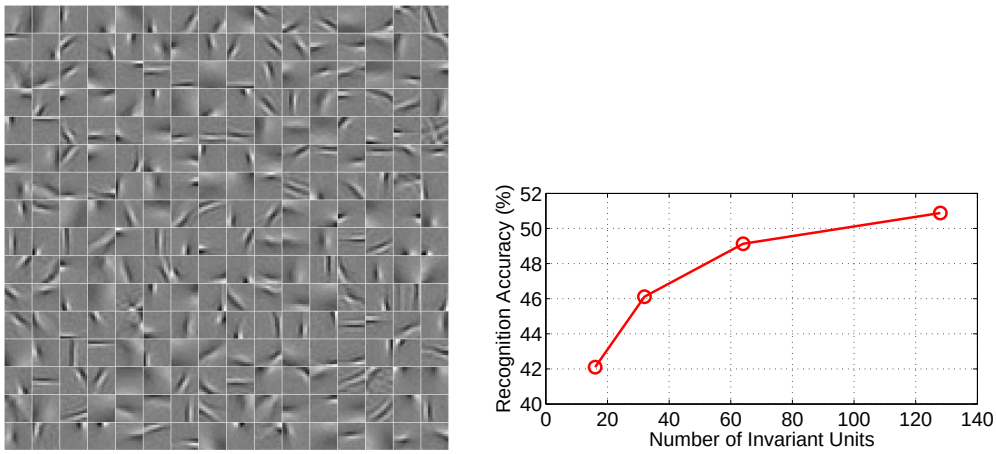
[2]Principal component analysis shows that the effective dimensionality of 9x9 natural image patches is about 47 since the first 47 principal components capture the 95% of the variance in the data. Hence, a 64-dimensional feature vector is actually an overcomplete representation for these 9x9 image patches.

Table 3.1: Comparison between representations produced by FS (Lee et al., 2006) and PSD. In order to compute the SNR, the noise is defined as $(Signal - Approximation)$.

Comparison (Signal / Approximation)	Signal to Noise Ratio (SNR)
1. PSD Optimal / PSD Encoder	8.6
2. FS / PSD Optimal	5.2
3. FS / PSD Encoder	3.1
4. FS / Regressor	3.2

PSD encoder achieves even better performance in recognition. The Caltech 101 dataset is pre-processed in the following way (Pinto et al., 2008): **1)** each image is converted to gray-scale, **2)** it is down-sampled so that the longest side is 151 pixels, **3)** the mean is subtracted and each pixel is divided by the image standard deviation, **4)** the image is locally normalized by subtracting the weighted local mean from each pixel and dividing it by the weighted norm if this is larger than 1 with weights forming a 9x9 Gaussian window centered on each pixel, and **5)** the image is 0-padded to 143x143 pixels. 64 feature detectors (either produced by FS or PSD Encoder) were plugged into an image classification system (Pinto et al., 2008) that **A)** used the sparse coding algorithms convolutionally to produce 64 feature maps of size 128x128 for each pre-processed image, **B)** applied an absolute value rectification, **C)** computed an average down-sampling to a spatial resolution of 30x30 and **D)** used a linear SVM classifier to recognize the object in the image (see fig. 3.3(b)). Using this system with 30 training images per class we can achieve 53% accuracy on Caltech 101 dataset.

It is important to observe that we must *rectify* the features in order to achieve good



(a) (b)

Figure 3.3: **a)** 256 basis functions of size 12x12 learned by PSD, trained on the Berkeley dataset. Each 12x12 block is a column of matrix W_d in eq. 3.2, i.e. a basis function. **b)** Object recognition architecture: linear adaptive filter bank, followed by abs rectification, average down-sampling and linear SVM classifier.

generalization. In other words, the features used for recognition are not the code units, but their sparsity errors (i.e., their absolute value). If we remove the absolute value rectification the accuracy on the test set drops to 16%, while all training samples are still correctly classified. We conjecture that a rectification is necessary for a twofold reason. First, it improves generalization by removing the polarity of edges which is irrelevant for recognition, since it is important to know there is an edge, but not if the object is brighter or darker than the background. Secondly, it avoids cancellations due to the band-pass nature of the learned filters that precede the anti-aliasing low-pass filter before the spatial down-sampling. These cancellations would propagate a very noisy signal to the classifier.

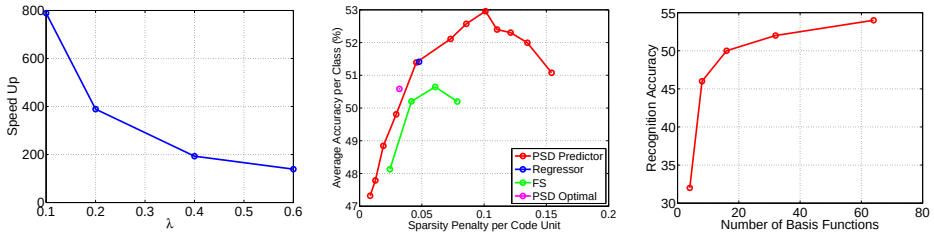
Since FS finds exact sparse codes, its representations are generally sparser than those found by the PSD encoder trained with the same value of sparsity penalty λ. Hence, we compare the recognition accuracy against the *measured* sparsity level of the representation as shown in fig. 3.4(b). PSD is not only able to achieve better accuracy than exact sparse coding algorithms, but also, it does it much more efficiently. Fig. 3.4(a) demonstrates that our feed-forward predictor extracts features more than 100 times faster than feature sign. In fact, the speed up is over 800 when the sparsity is set to the value that gives the highest accuracy shown in fig. 3.4(b).

Finally, we observe that these sparse coding algorithms are somewhat inefficient when applied convolutionally. Many feature detectors are the translated versions of each other as shown in fig. 3.3(a). Hence, the resulting feature maps are highly redundant. This might explain why the recognition accuracy tends to saturate when the number of filters is increased as shown in fig. 3.4(c).

3.3.3 Stability

In order to quantify the stability of PSD and FS, we investigate their behavior under naturally changing input signals. For this purpose, we train a basis set with 128 elements, each of size 9x9, using the PSD algorithm on the Berkeley (Ber,) dataset. This basis set is then used with FS on the standard "foreman" test video together with the PSD Predictor. We extract 784 uniformly distributed patches from each frame with a total of 400 frames.

For each patch, a 128 dimensional representation is calculated using both FS and the PSD predictor. The stability is measured by the number of times a unit of the representa-



(a) (b) (c)

Figure 3.4: **a)** Speed up for inferring the sparse representation achieved by the PSD encoder over FS for a code with 64 units. The feed-forward extraction is more than 100 times faster. **b)** Recognition accuracy versus measured sparsity (average ℓ^1 norm of the representation) of the PSD encoder compared to the to the representation of FS algorithm. A difference within 1% is not statistically significant. **c)** Recognition accuracy as a function of the number of basis functions.

tion changes its sign, either negative, zero or positive, between two consecutive frames. Since the PSD predictor does not generate exact zero values, we threshold its output units in such a way that the average number of zero units equals the one produced by FS (roughly, only the 4% of the units are non-zero). The transition probabilities are given in Figure 3.5. It can be seen from this figure that the PSD predictor generates a more stable representation of slowly varying natural frames compared to the representation produced by the exact optimization algorithm.

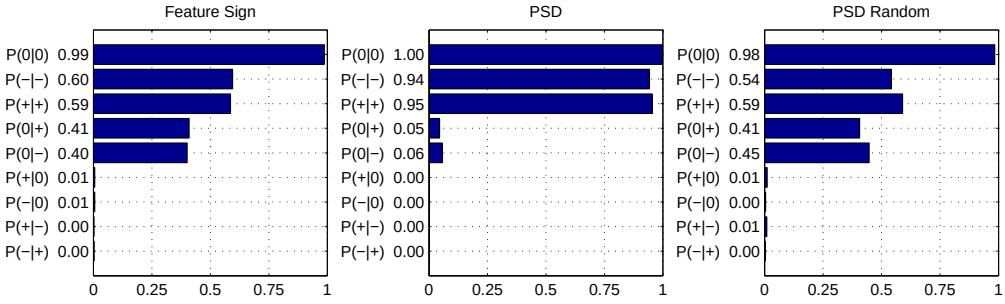


Figure 3.5: Conditional probabilities for sign transitions between two consecutive frames. For instance, $P(-|+)$ shows the conditional probability of a unit being negative given that it was positive in the previous frame. The figure on the right is used as baseline, showing the conditional probabilities computed on pairs of *random* frames.

4

LEARNING INVARIANT REPRESENTATIONS

In this chapter we study methods to learn invariant representations. We are interested in learning features that are invariant to transformations that are irrelevant for recognition. By mapping similar input data vectors into the same feature vector, we can make much easier the training of a subsequent supervised classifier. This is a typical problem in vision. For instance, consider the problem of recognizing generic object categories in images. Ideally, we would like to learn a representation that is invariant, or robust, to changes in illumination, position, scale, and orientation of objects. If we had such representation, then we could train a classifier using more compact representations and fewer labeled samples per class. Since most applications requiring invariant representations are in the computer vision field, we refer to vision problems in this chapter. However, the methods are general and can be applied to any other domain.

The most successful and most commonly-used (although not adaptive) invariant descriptors, such as SIFT and HoG (Lowe, 2004; Dalal and Triggs, 2005), are computed on patches extracted at a regularly spaced grid on the input image. Each patch is convolved with a filter bank (often consisting of oriented edge detectors), the outputs of which are rectified and often normalized and quantized. Then, the outputs of each filter are spatially pooled using a simple addition or a max operator, so as to build local bags of features. The pooling operation makes the descriptor robust to minor changes in the

position of individual features. This architecture is somewhat similar (and inspired by) that of the early areas of the mammalian visual cortex: simple cells detect oriented edges at various locations and scales (playing the same role as the filter bank). Highly-active simple cells inhibit other cells at neighboring locations and orientations (like the normalization), while complex cells spatially pool the rectified outputs of complex cells, so as to create a local invariance to small shifts (like the pooling operation).

The problem of *learning* invariant image features has become a topic of growing interest in recent years. Supervised learning methods have long been used in conjunction with Convolutional Networks to learn low-level, locally invariant features that are tuned to the task at hand (LeCun et al., 1998; LeCun et al., 2004), but these methods require large numbers of labelled samples. A number of different proposals have appeared for unsupervised learning of locally-invariant descriptors (Wiskott and Sejnowski, 2002; Foldiak, 1991), which also use sparsity criteria (Hyvarinen and Hoyer, 2001; Osindero et al., 2006; Hyvarinen and Koster, 2007; Ranzato et al., 2007c; Cadieu and Olshausen, 2008; Kavukcuoglu et al., 2009).

Some methods assume knowledge of the transformations to which the representation has to be invariant (Ranzato et al., 2007c). This is the simplest case and it will be described in section 4.1. There are two ways of learning such representations. The first one is to "hard-wire" the invariance in the architecture. For instance, a rotation invariant representation can be computed by applying a filter bank of edge detectors in different orientations, followed by a max operator across the corresponding feature maps. The output of the max operator is the same no matter the orientation of the input pattern, achieving invariance to rotation. The second way is to apply the transformation to each

training sample, and to train the system to produce the same representation when taking as input either the original or the transformed input.

A more ambitious and difficult problem is to learn representations that are invariant to *learned* transformations. At a very high level, these methods use the same principles as before. In slow feature analysis (Wiskott and Sejnowski, 2002) the system is trained on time-varying natural signals and the internal representation is forced to vary smoothly across time, achieving invariance (or robustness) to small distortions. Other methods learn transformations by relying on a carefully chosen architecture (Cadieu and Olshausen, 2008), or by trying to learn how to pool similar features (Kohonen, 1996; Hyvarinen and Hoyer, 2000; Hyvarinen and Koster, 2007). Section 4.2 will describe an algorithm that produces representations invariant to learned transformations by pooling features that are similar, while enforcing sparsity across pools of features (Kavukcuoglu et al., 2009). Overall, these algorithms are simple extensions of sparse coding algorithms, like the one described in the previous chapter.

4.1 Learning Locally-Shift Invariant Representations

An image patch can be modeled as a collection of features placed at particular locations within the patch. A patch can be reconstructed from the list of features that are present in the patch together with their respective locations. In the simplest case, the features are templates (or basis functions) that are combined additively to reconstruct a patch. If we assume that each feature can appear at most once within a patch, then computing a shift-invariant representation comes down to applying each feature detector at all locations in the patch, and recording the location where the response is the largest. Hence

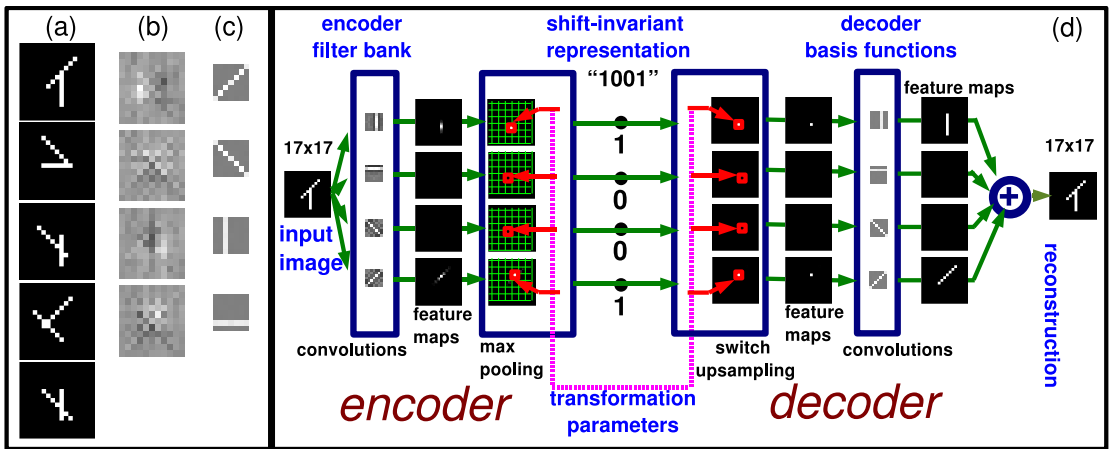


Figure 4.1: Left Panel: (a) sample images from the "two bars" dataset. Each sample contains two intersecting segments at random orientations and random positions. (b) Non-invariant features learned by an auto-encoder with 4 hidden units. (c) Shift-invariant decoder filters learned by the proposed algorithm. The algorithm finds the most natural solution to the problem. Right Panel (d): architecture of the shift-invariant unsupervised feature extractor applied to the two bars dataset. The encoder convolves the input image with a filter bank and computes the max across each feature map to produce the invariant representation. The decoder produces a reconstruction by taking the invariant feature vector (the "what"), and the transformation parameters (the "where"). The reconstructions is the sum of each decoder basis function at the position indicated by the transformation parameters, and weighted by the corresponding feature component.

the invariant feature vector records the presence or absence of each feature in the patch (*what* is in the image), while the so called *transformation parameters* record the location at which each feature output is the largest (*where* each feature appears in the image)[1][2][3]. In general, the feature outputs need not be binary. Before describing the learning algorithm, we show how a trained system operates using a toy example as an illustration. Each input sample is a binary image containing two intersecting bars of equal length, as shown in fig. 4.1(a). Each bar is 7 pixels long, has 1 of 4 possible orientations, and is placed at one of 25 random locations (on a 5×5 grid) at the center of a 17×17 image frame. The input image is passed through 4 convolutional filters of size 7×7 pixels. The convolution of each detector with the input produces an 11×11 feature map. A *max-pooling* layer finds the largest value in each feature map, recording the position of this value as the *transformation parameter* for that feature map. The invariant feature vector collects these max values, recording the presence or absence of each feature independently of its position. No matter where the two bars appear in the input image, the result of the *max-pooling* operation will be identical for two images containing bars of identi-

[1]This *matched filter* kind of approach is susceptible to failure because of interference from nearby features and because of non-zero responses of the filter to other patterns. However, the algorithm is very robust in practice and it seems that the minimization of the reconstruction error forces the system to resolve the interference problem.

[2]The representation might be invariant to other transformations than shift. This might be undesired depending on the application. It is an open research question how to achieve invariance exclusively to a given type of transformation preserving the information relevant to achieve a given task.

[3]In some applications, the information about the location of the features might carry useful information that cannot be discarded. One possibility is to use this algorithm on *overlapping* windows in order to implicitly retain the information about position in the representation through *cross-predictive coding*.

cal orientations at different locations. The reconstructed patch is computed by placing each code value at the proper location in the decoder feature map, using the transformation parameters 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 (which are essentially identical to the corresponding filters in the encoder) weighted by the feature map values at all locations.

A solution to this toy experiment is one in which the invariant representation encodes the information about which orientations are present, while the transformation parameters encode where the two bars appear in the image. The oriented bar detector filters shown in the figure are in fact the ones discovered by the learning algorithm described in the next section. In general, this architecture is not limited to binary images, and can be used to compute shift invariant features with any number of components.

4.1.1 Learning Algorithm

The encoder is given by two functions $Z = \mathrm{Enc}_Z(Y; W_e)$ and $U = \mathrm{Enc}_U(Y; W_e)$ where Y is the input image, W_e is the trainable parameter vector of the encoder (the filters), Z is the invariant feature vector, and U is the transformation parameter vector. Similarly, the decoder is a function $\mathrm{Dec}(Z, U; W_d)$ where W_d is the trainable parameter vector of the decoder (the basis functions). The input reconstruction error E_d measures the square Euclidean distance between the input Y and its reconstruction $E_d = ||Y - \mathrm{Dec}(Z, U; W_d)||^2$. Likewise, the code prediction error is the squared Euclidean distance between the invariant code produced by the encoder and the optimal code: $E_e = ||Z - \mathrm{Enc}_Z(Y; W_e)||^2$. Learning proceeds by block-coordinate descent over Z

and the parameters in encoder and decoder:

1. First, propagate the input Y through the encoder to produce the predicted code $Z_0 = \mathrm{Enc}_Z(Y; W_e)$ and the transformation parameters $U = \mathrm{Enc}_U(Y; W_e)$ that are then copied into the decoder. Keeping U fixed, and using Z_0 as initial value for the code Z, minimize the energy $E_d + E_e$ with respect to the code Z by gradient descent to produce the optimal code Z^*.

2. Update the weights in the encoder and decoder by one step of gradient descent so as to minimize the decoder energy.

Note that this is a *general learning algorithm* that is suitable for any encoder-decoder architecture, and not specific to a particular kind of feature or architecture choice. Any differentiable module can be used as encoder or decoder. In particular, we can use the encoder-decoder architecture described in chapter 3 by adding to the energy a sparsity penalty on the code, and by using a linear decoder and a simple non-linear encoder, for instance. This would produce feature that are not only sparse, but also shift-invariant.

We tested the proposed architecture and learning algorithm on the "two bars" toy example described in the previous section. In this experiment, both the encoder and the decoder are linear functions of the parameters (linear filters and linear basis functions) and the energy is simply the sum of encoder and decoder reconstruction error. The input images are 17×17 binary images containing two bars in different orientations: horizontal, vertical and the two diagonals as shown in fig. 4.1(a). The encoder contains four 7×7 linear filters, plus four 11×11 max-pooling units. The decoder contains four 7×7 linear basis functions. The parameters are randomly initialized. The learned basis functions

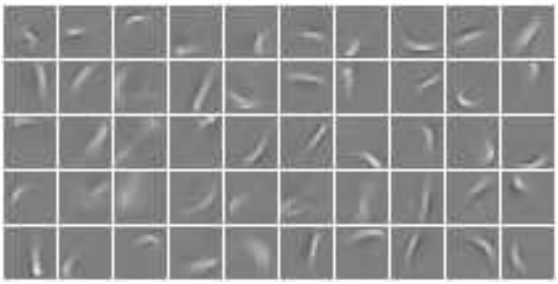


Figure 4.2: Fifty 20×20 filters learned in the decoder by the sparse and shift invariant learning algorithm after training on the MNIST dataset of handwritten digits of size 28×28 pixels. A digit is reconstructed as linear combination of a small subset of these features positioned at one of 81 possible locations (9×9), as determined by the transformation parameters produced by the encoder.

are shown in fig. 4.1(c), and the encoder filters in fig. 4.1(d). After training on a few thousand images, the filters converge as expected to the oriented bar detectors shown in the figure. The resulting 4-dimensional representation extracted from the input image is translation invariant. These filters and the corresponding representation differ strikingly from what can be achieved by PCA or an auto-encoder neural network. For comparison, an auto-encoder neural network with 4 hidden units was trained on whole images from this dataset. The filters (weights of the hidden units) are shown in fig. 4.1(b). There is no appearance of oriented bar detectors, and the resulting codes are not shift invariant.

In fig. 4.2 we show an example of sparse and shift invariant features (using an extension of the sparse coding algorithm described in (Ranzato et al., 2006), but similar results are achieved by using the algorithm proposed in chapter 3). The algorithm is applied to the handwritten digits of the MNIST dataset (MNI,), which consists of quasi-binary

images of size 28×28 pixels. We considered a set of fifty 20×20 filters in both encoder and decoder that are applied to the input at 81 locations (9×9 grid), over which the max-pooling is performed. Hence image features can move over those 81 positions while leaving the invariant feature vector unchanged. Because the feature vectors must be sparse, the learned features (shown in fig. 4.2) look like part detectors. Each digit can be expressed as a linear combination of a small number of these 50 parts, placed at one of 81 locations in the image frame. Unlike with the non-invariant method described in (Ranzato et al., 2006), no two filters are shifted versions of each other. Moreover, the reconstruction error is reduced by a factor of two compared to the non-invariant approach.

4.2 Learning Representations Invariant to Generic Transformations

The algorithm proposed in this section (Kavukcuoglu et al., 2009) is similar to the previous one in the fact that it combines feature detection and pooling during learning, but it does not assume anything about the transformation and the filters are not applied convolutionally.

Our aim is to learn the filter bank stage and the pooling stage simultaneously, in such a way that filters belonging to the same pool extract similar features. Our solution is to pre-wire (before learning) which filters are pooled together, and to let the filters learn. The main idea, borrowed from (Hyvarinen and Koster, 2007), is to minimize a sparsity criterion on the pooling units. As a result, filters that are pooled together end up

extracting similar features.

Several authors have proposed methods to learn pooled features (Kohonen, 1996; Hyvarinen and Hoyer, 2000; Hyvarinen and Koster, 2007). When the filters that are pooled together are organized in a regular array (1D or 2D), the filters form *topographic maps* in which nearby filters extract similar features (Osindero et al., 2006; Hyvarinen and Hoyer, 2001). In this work for the first time, a trainable topographically-organized feature map is also used for extracting locally invariant image descriptors for image recognition.

This is an extension to the sparse coding algorithm proposed in chapter 3 that overcomes one of the most well-known idiosyncrasy, namely the instability of the representation. If the input pattern is slightly distorted, the representation might drastically change. The use of a direct encoding regressor partially solves this issue because it makes the mapping smoother (see sec. 3.3.3), but the use of invariant features further improves the stability as demonstrated in sec. 4.2.2.

4.2.1 Modeling Invariant Representations

Let us consider the standard sparse coding algorithm introduced by Olshausen and Field (Olshausen and Field, 1997). The loss is equal to:

$$L(Y, Z; W_d) = \|Y - W_d Z\|_2^2 + \lambda \|Z\|_1 \tag{4.1}$$

We now describe how the sparsity term in eq. 4.1 can be modified to create coefficients that are invariant to perturbations of the input signal.

The overall idea (Hyvarinen and Koster, 2007) is to arrange the Z's into a 2D map (or some other topology) and then pool the squared coefficients of Z across overlapping

windows. Then, the square of the the filter outputs within each sub-window are summed, and its square root is computed. More formally, let the map of Z contain K overlapping neighborhoods P_i. Within each neighborhood i, we sum the squared coefficients Z_j (weighted by a fixed Gaussian weighting function centered in the neighborhood) and then take the square root. This gives the activation $v_i = \sqrt{\sum_{j \in P_i} w_j z_j^2}$, where w_j are the Gaussian weights. The overall sparsity penalty is the sum of each neighborhood's activation: $\sum_{i=1}^{K} v_i$. Figure 4.3(a) illustrates this scheme. Thus, the overall objective function is now:

$$L_I = \frac{1}{2}||Y - W_d Z||_2^2 + \lambda \sum_{i=1}^{K} \sqrt{\sum_{j \in P_i} w_j z_j^2} \tag{4.2}$$

The modified sparsity term has a number of subtle effects on the nature of Z that are not immediately obvious:

- The square root in the sum over i encourages sparse activations *across* neighborhoods since a few large activations will have lower overall cost than many small ones.

- *Within* each neighborhood i, the coefficients z_j are encouraged to be similar to one another due to the z_j^2 term (which prefers many small coefficients to a few large ones, see also fig. 4.4). This has the effect of encouraging similar basis functions in W_d to be spatially close in the 2D map.

- As the neighborhoods overlap, these basis functions will smoothly vary across the map, so that the coefficients z_j in any given neighborhood i will be similar.

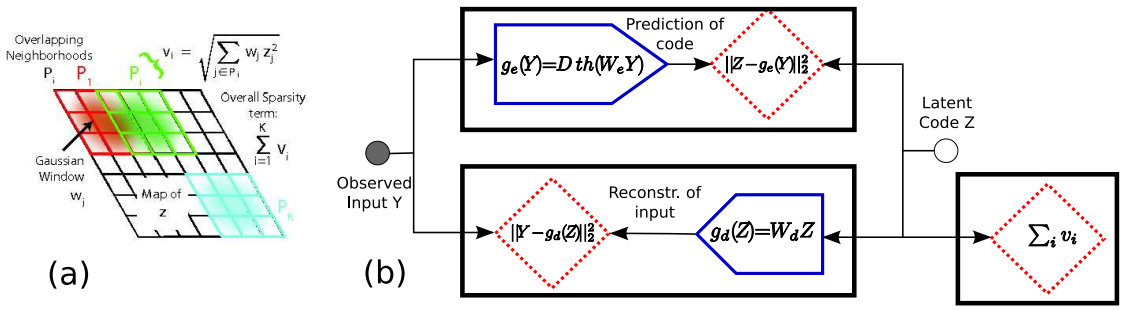


Figure 4.3: **(a)**: The structure of the block-sparsity term which encourages the basis functions in W_d to form a topographic map. See text for details. **(b)**: Overall architecture of the loss function, as defined in eq. 4.3. In the generative model, we seek a feature vector Z that simultaneously approximate the input Y via a dictionary of basis functions W_d and also minimize a sparsity term. Since performing the inference at run-time is slow, we train a prediction function $g_e(Y; W)$ (dashed lines) that directly predicts the optimal Z from the input Y. At run-time we use only the prediction function to quickly compute Z from Y, from which the invariant features v_i can be computed.

- If the size of the pooling regions is reduced to a single Z element, then the sparsity term is equivalent to that of eq. 4.1.

The modified sparsity term means that by minimizing the loss function L_I in eq. 4.2 with respect to both the coefficients Z and the dictionary W_d, the system learns a set of basis functions in W_d that are laid out in a *topographic map* on the 2D grid.

Since the nearby basis functions in the topographic map are similar, the coefficients z_j will be similar for a given input Y. This also means that if this input is perturbed slightly then the pooled response within a given neighborhood will be minimally affected, since a decrease in the response of one filter will be offset by an increased response in a nearby one. Hence, we can obtain a locally robust representation by taking the pooled activations v_i as features, rather than Z as is traditionally done.

Since invariant representations encode similar patterns with the same representation, they can be made more compact. Put another way, this means that the dimensionality of v can be made lower than the dimensionality of Z without loss of useful information. This has the triple benefit of requiring less computation to extract the features from an image, requiring less memory to store them, and requiring less computation to exploit them.

The 2D map over Z uses circular boundary conditions to ensure that the pooling wraps smoothly around at the edges of the map.

In order to make the algorithm efficient at test time, that is, to speed up inference of the invariant representation, we use an encoder to directly predict Z from Y and we add a corresponding penalty in the loss function as already done in the previous chapter.

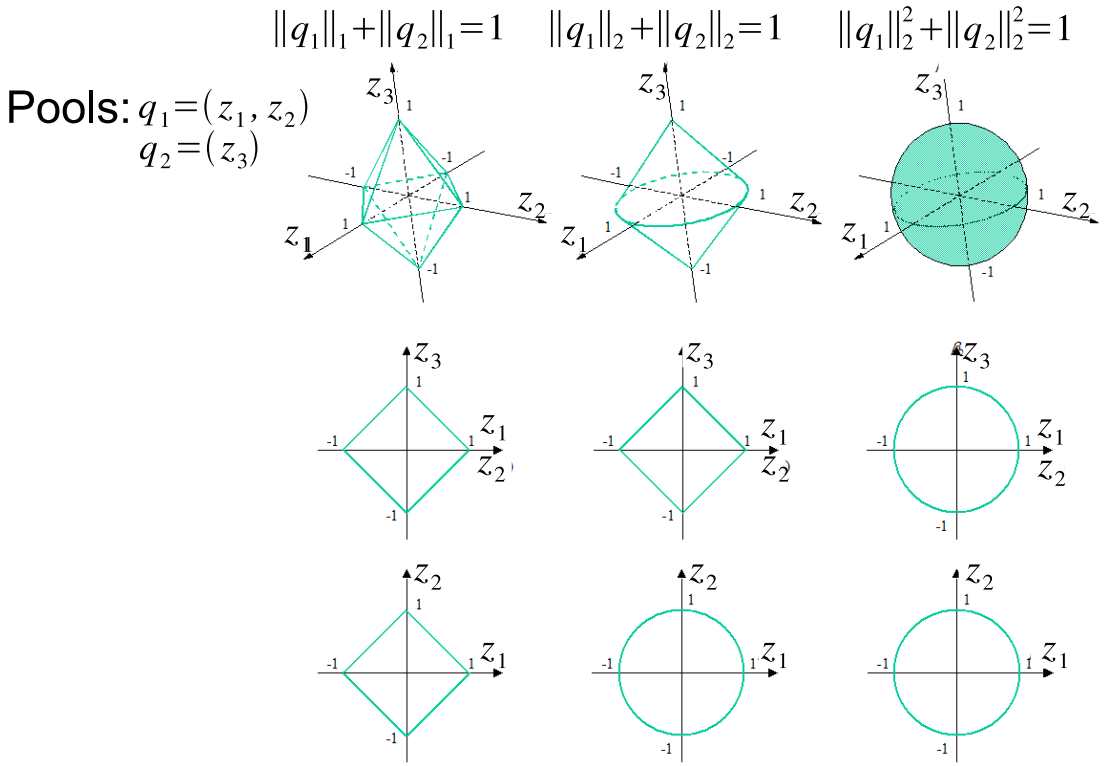


Figure 4.4: Level sets induced by different sparsity penalties (the figure was taken from Yuan and Lin's paper (Yuan and Lin, 2004)). There are two pools. The first one has two units (Z_1, Z_2), and the second one has only one unit (Z_3). The first row shows the level set in 3D, while the second and the third rows show the projections on the coordinate planes. The first column is the L1 norm of the units, the second column is the proposed sparsity penalty (grouped lasso), and the third one is the L2 norm of the units. The proposed sparsity penalty enforces sparsity across pools, but not within a pool.

Linking to the previous chapter, the overall energy function and loss are:

$$E(Y, Z; W_d, W_e, D) = \|Y - W_d Z\|_2^2 + \alpha \|Z - g_e(Y; W_e, D)\|_2^2 + \lambda \sum_{i=1}^{K} \sqrt{\sum_{j \in P_i} w_j Z_j^2}$$

$$L_{IPSD}(Y; W_d, W_e, D) = F(Y; W_d, W_e, D) = \min_Z E(Y, Z; W_d, W_e, D) \quad (4.3)$$

where the encoding mapping is the same one used before: $g_e(Y; W_e, D) = D \tanh(W_e Y)$.

The learning algorithm is unchanged, and it consists of a block coordinate gradient descent optimization alternating a minimization over the code Z, and over the parameters in both encoder and decoder (see sec. 3.2).

Once the parameters are learned, computing the invariant representation V can be performed by a simple feed-forward propagation through the encoder $g_e(Y; W_e, D)$, and then by mapping Z into V through $V_i = \sqrt{\sum_{j \in P_i} w_j Z_j^2}$. We will call this method Invariant Predictive Sparse Decomposition (IPSD).

4.2.2 Experiments

We first study the topographic map produced by our training scheme, before exploring the properties of the invariant features obtained. First, we make an empirical evaluation of the invariance achieved by these representations under translations and rotations of the input image. Second, we assess the discriminative power of these invariant representations on recognition tasks in three different domains: (i) generic object categories using the Caltech 101 dataset; (ii) generic object categories from a dataset of very low resolution images and (iii) classification of handwriting digits using the MNIST dataset. We show examples from the latter two datasets in Fig. 4.7. In these experiments we compare our learned representations with the SIFT descriptor (Lowe, 2004) that is con-

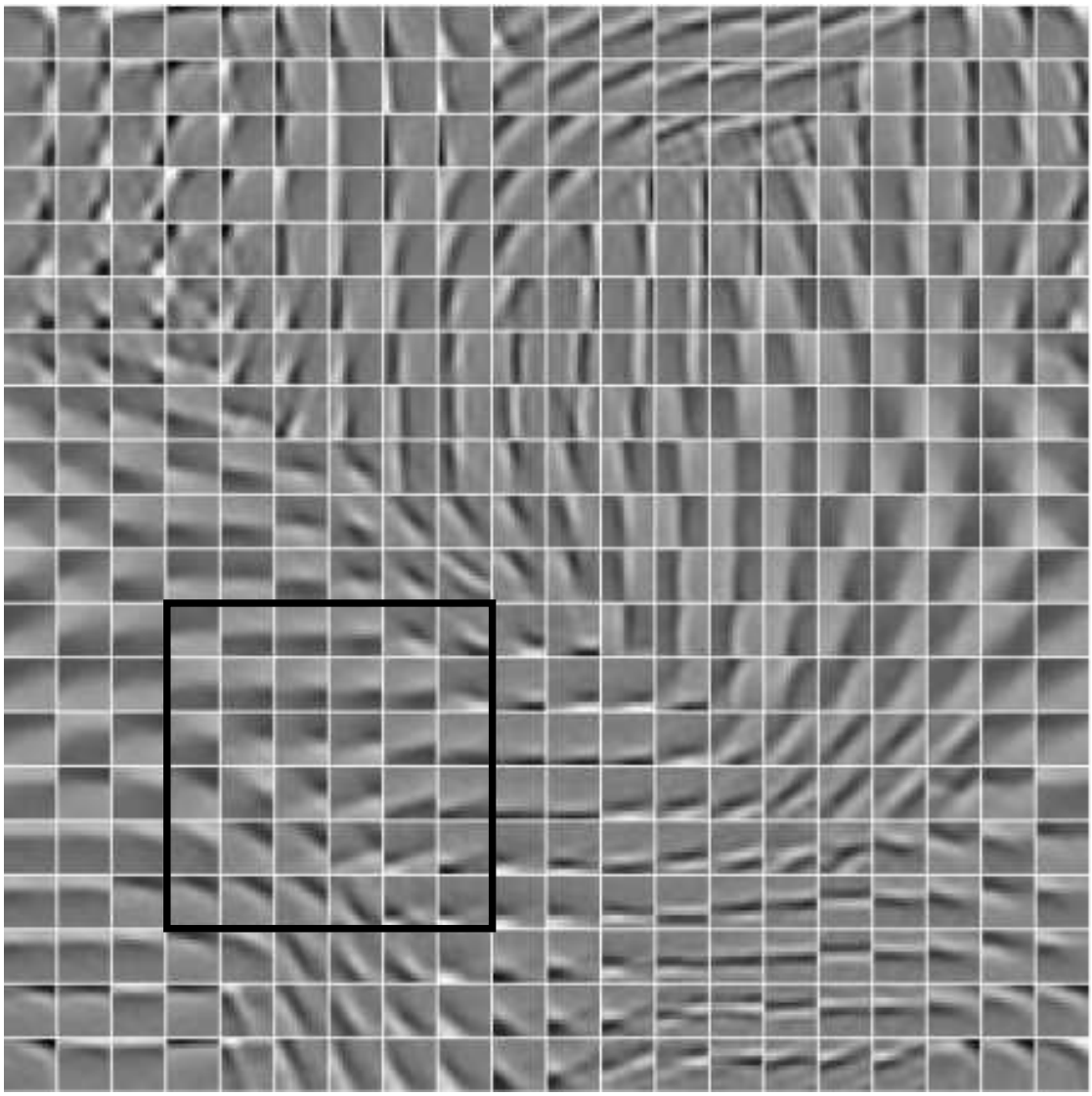


Figure 4.5: Topographic map of feature detectors learned from natural image patches of size 12x12 pixels by optimizing the loss in eq. 4.3. There are 400 filters that are organized in 6x6 neighborhoods. Adjacent neighborhoods overlap by 4 pixels both horizontally and vertically. Notice the smooth variation within a given neighborhood and also the circular boundary conditions.

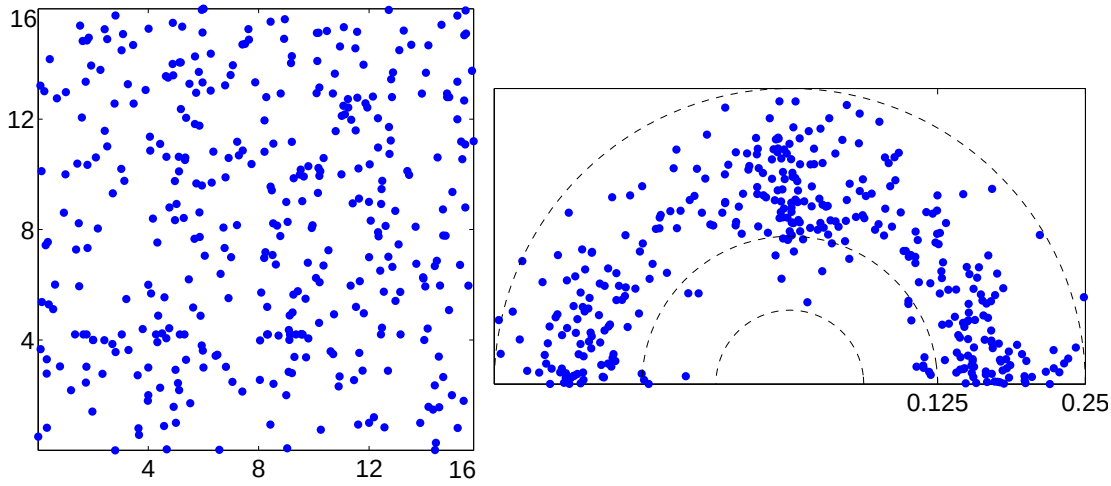


Figure 4.6: Analysis of learned filters by fitting Gabor functions, each dot corresponding to a filter. Left: Center location of fitted Gabor. Right: Polar map showing the joint distribution of orientation (azimuthally) and frequency (radially in cycles per pixel) of Gabor fit.

Figure 4.7: Left: Examples from the MNIST dataset. Right: Examples from the tiny images. We use gray-scale images in our experiments.

sidered a state-of-the-art descriptor in computer vision. Finally, we examine the computational cost of computing our features on an image.

Learning the topographic map

Fig. 4.5 shows a typical topographic map learned by the proposed method from natural image patches. Each tile shows the filter in W_d corresponding to a particular Z_i. In the example shown, the input images are patches of size 12x12 pixels, and there are 400 basis functions, and hence, 400 units Z_i arranged in a 20x20 lattice. The neighborhoods over which the squared activities of Z_i's are pooled are 6x6 windows, and they overlap by 4 in both the vertical and the horizontal direction. The properties of these filters are analyzed by fitting Gabor functions and are shown in Fig. 4.6.

By varying the way in which the neighborhoods are pooled, we can change the properties of the map. Larger neighborhoods make the filters in each pool increasingly similar. A larger overlap between windows makes the filters vary more smoothly across different pools. A large sparsity value λ makes the feature detectors learn less localized patterns that look like those produced by k-means clustering because the input has to be reconstructed using a small number of basis functions. On the other hand, a small sparsity value makes the feature detectors look like non-localized random filters because any random overcomplete basis set can produce good reconstructions (effectively, the first term in the loss of eq. 4.3 dominates).

The map in fig. 4.5 has been produced with an intermediate sparsity level of $\lambda = 3$. The chosen parameter setting induces the learning algorithm to produce a smooth map with mostly localized edge detectors in different positions, orientations, and scales. These features are nicely organized in such a way that neighboring units encode similar patterns. A unit V_i that connects to the sum of the squares of units Z_j in a pool is invariant because these units represent similar features, and small distortions applied to the input, while slightly changing the Z_j's within a pool, are likely to leave the corresponding V_i unaffected.

While the sparsity level, the size of the pooling windows and their overlap should be set by cross-validation, in practice we found their exact values does not significantly affect the kind of features learned. In other words, the algorithm is quite robust to the choice of these parameters, probably because of the many constraints enforced during learning.

Analyzing Invariance to Transformations

In this experiment we study the invariance properties of the learned representation under simple transformations. We have generated a dataset of 16x16 natural image patches under different translations and rotations. Each patch is presented to our predictor that produces a 128 dimensional descriptor (chosen to be the same size as SIFT) of V's. A representation can be considered invariant if it does not change significantly as the input is transformed. Indeed, this is what we observe in Fig. 4.8. We compare the mean squared difference between the descriptor of the reference patch and the descriptor of the transformed version, averaged over many different image patches. The figure shows ours against SIFT with a varying horizontal shift for 0 and 25 degrees rotation. Very similar results are found for vertical shifts and other rotation angles.

On the left panel, we can see that the mean squared error (MSE) between the representation of the original patch and its transformation increases linearly as we increase the horizontal shift. The MSE of our representation is generally lower than the MSE produced by features that are computed using SIFT, a non-rotation invariant version of SIFT, and a non-invariant representation produced by the proposed method (that was trained with pools of size 1x1, like PSD described in the previous chapter). A similar behavior is found when the patch is not only shifted, but also rotated. When the shift is small, SIFT has lower MSE. But as soon as the translation becomes large enough that the input pattern falls in a different internal sub-window, the MSE increases considerably. Instead our learned representations seem to be quite robust to shifts, with an overall lower area under the curve. Note also that traditional sparse coding algorithms are prone to produce unstable representations under small distortions of the input. Because each

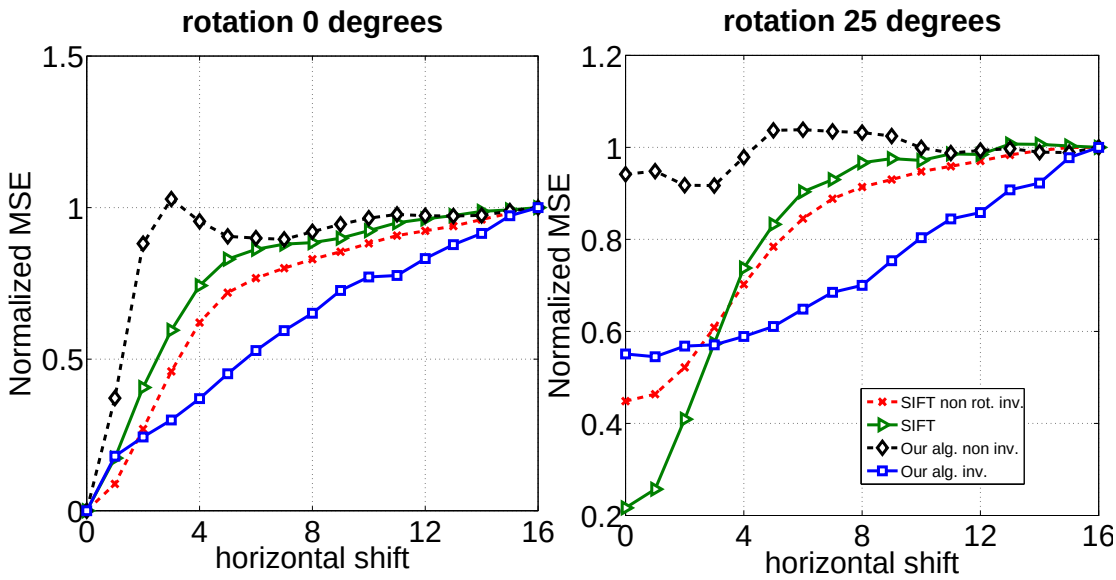


Figure 4.8: Mean squared error (MSE) between the representation of a patch and its transformed version. On the left panel, the transformed patch is horizontally shifted. On the right panel, the transformed patch is first rotated by 25 degrees and then horizontally shifted. The curves are an average over 100 patches randomly picked from natural images. Since the patches are 16x16 pixels in size, a shift of 16 pixels generates a transformed patch that is quite uncorrelated to the original patch. Hence, the MSE has been normalized so that the MSE at 16 pixels is the same for all methods. This allows us to directly compare different feature extraction algorithms: non-orientation invariant SIFT, SIFT, the proposed method trained to produce non-invariant representations (i.e. pools have size 1x1), and the proposed method trained to produce invariant representations. All algorithms produce a feature vector with 128 dimensions. Our method produces representations that are more invariant to transformations than the other approaches for most shifts.

input has to be encoded with a small number of basis functions, and because the basis functions are highly tuned in orientation and location, a small change in the input can produce drastic changes in the representation. This problem is partly alleviated by our approximate inference procedure that uses a smooth encoder function. However, this experiment shows that this representations is still fairly unstable under small distortions, when compared to the invariant representations produced by the invariant algorithm and SIFT.

Generic Object Recognition

We now use our invariant features for object classification on the Caltech 101 dataset (Fei-Fei et al., 2004) of 101 generic object categories. We use 30 training images per class and up to 20 test images per class. The images are randomly picked, and pre-processed as described in sec. 3.3.2.

We have trained our method on 50,000 16x16 patches randomly extracted from the pre-processed images. The topographic map used has size 32x16, with the pooling neighborhoods being 6x6 and an overlap of 4 coefficients between neighborhoods. Hence, there are a total of 512 units that are used in 128 pools to produce a 128-dimensional representation that can be compared to SIFT. After training our algorithm in an unsupervised way, we use the encoder function to infer the representation of one whole image by: (i) running the encoder on 16x16 patches spaced by 4 pixels to produce 128 maps of features of size 34x34; (ii) the maps of features are low-pass filtered with a boxcar filter to avoid aliasing; (iii) the maps are then projected along the leading 3060 principal components (equal to the number of training samples), and (iv) a supervised linear SVM is trained to recognize the object in each corresponding image. The overall scheme is

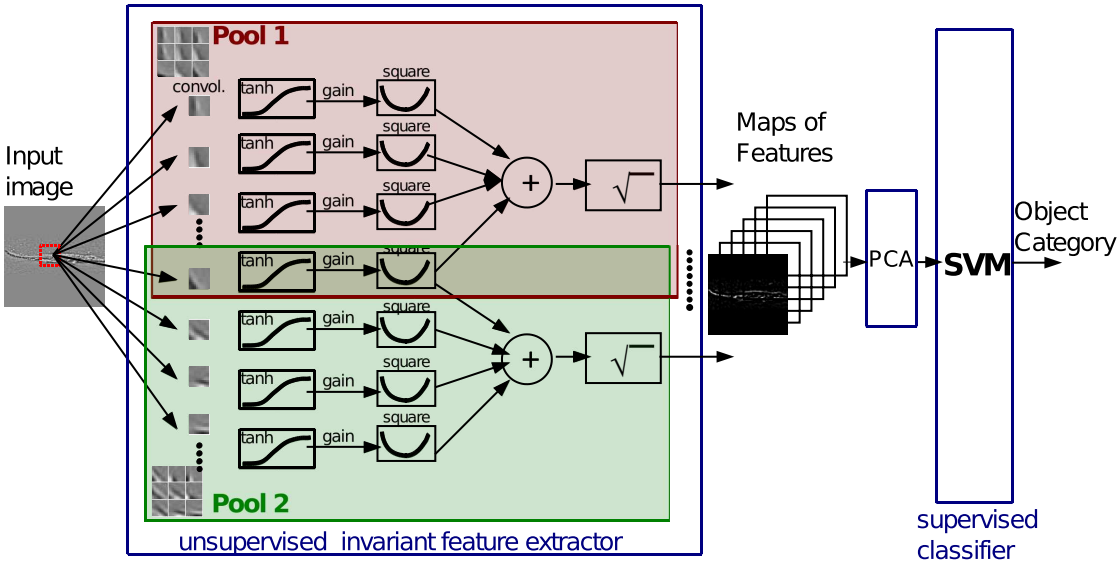


Figure 4.9: Diagram of the recognition system. This is composed of an invariant feature extractor that has been trained unsupervised, followed by a supervised linear SVM classifier. The feature extractor process the input image through a set of filter banks, where the filters are organized in a two dimensional topographic map. The map defines pools of similar feature detectors whose activations are first non-linearly transformed by a hyperbolic tangent non-linearity, and then, multiplied by a gain. Invariant representations are found by taking the square root of the sum of the squares of those units that belong to the same pool. The output of the feature extractor is a set of maps of features that can be fed as input to the classifier. The filter banks and the set of gains is learned by the algorithm. Recognition is very fast, because it consists of a direct forward propagation through the system.

shown in Fig. 4.9. Table 4.1 reports the recognition results for this experiment. Using a linear SVM classifier our features outperforms SIFT and the multi-scale Gabor system proposed by Serre and Poggio (Serre et al., 2005). However, if rotation invariance is removed from SIFT the performance becomes comparable to SIFT. With the more sophisticated Spatial Pyramid Matching Kernel SVM classifier (Lazebnik et al., 2006a), our features yield an average accuracy per class equal to 59.6%. By decreasing the stepping stride to 1 pixel, thereby producing 120x120 feature maps, our features achieve 65.5% accuracy as shown in table 4.1. This is comparable to Lazebnik's 64.6% accuracy on Caltech-101 (without background class) (Lazebnik et al., 2006a). For comparison, our re-implementation of Lazebnik's SIFT feature extractor, stepped by 4 pixels to produce 34x34 maps, yielded 65% average recognition rate. With 128 invariant features, each descriptor takes around 4ms to compute from a 16x16 patch. Note that the evaluation time of each region is a linear function of the number of features, thus this time can be further reduced if the number of features is reduced. Fig. 4.10 shows how the recognition performance varies the number of features is decreased.

Tiny Images classification

The proposed method was compared to SIFT on another recognition task using a tiny images dataset (Torralba et al., 2008). This was chosen as its extreme low-resolution provides a different setting to the Caltech 101 images. For simplicity, we selected 5 animal nouns (Abyssinia cat, angel shark, apatura iris (a type of butterfly), bilby (a type of marsupial), may beetle) and manually labeled 200 examples of each. 160 images of each class were used for training, with the remaining 40 held out for testing. All images are converted to gray-scale. Both our algorithm with 128 pooled units and SIFT

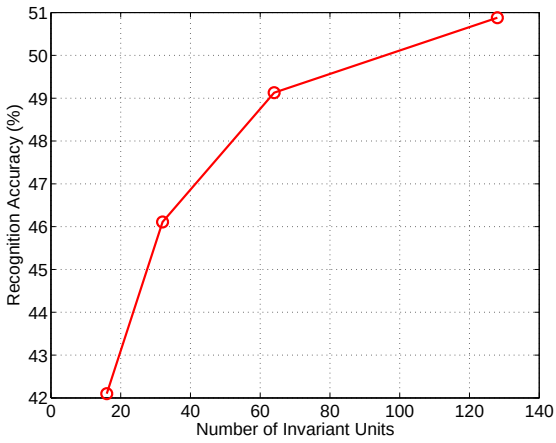


Figure 4.10: The figure shows the recognition accuracy on the Caltech 101 dataset as a function of the number of invariant units (and thus the dimensionality of the descriptor). Note that the performance improvement between 64 and 128 units is below 2%, suggesting that for certain applications the more compact descriptor might be preferable.

were used to extract features over 16x16 regions, spaced every 4 pixels over the 32x32 images. The resulting 5 by 5 by 128 dimensional feature maps are then fed into a linear SVM classifier, as before. Our features achieve 54% correct classification rate and SIFT features achieve 53% correct classification rate. Hence our learned features perform comparably to SIFT.

Handwriting Recognition

We use a very similar architecture to that used in the experiments above to train on the handwritten digits of the MNIST dataset (MNI,). This is a dataset of quasi-binary handwritten digits with 60,000 images in the training set, and 10,000 images in the test set. The algorithm was trained using 16x16 windows extracted from the original 28x28 pixel images. For recognition, 128-dimensional feature vectors were extracted at 25 locations regularly spaced over a 5x5 grid. A linear SVM trained on these features yields an error rate of **1.0%**. When 25 SIFT feature vectors are used instead of our invariant features, the error rate increases to **1.5%**. This demonstrates that, while SIFT seems well suited to natural images, our method produces features that can *adapt* to the task at hand.

Method	**Av. Accuracy/Class** (%)
local norm$_{5\times5}$ + boxcar$_{5\times5}$ + PCA$_{3060}$ + linear SVM	
IPSD (24x24)	**50.9**
SIFT (24x24) (non rot. inv.)	**51.2**
SIFT (24x24) (rot. inv.)	**45.2**
Serre et al. features (Serre et al., 2005)	**47.1**
local norm$_{9\times9}$ + Spatial Pyramid Match Kernel SVM	
SIFT (Lazebnik et al., 2006a)	**64.6**
IPSD (34x34)	**59.6**
IPSD (56x56)	**62.6**
IPSD (120x120)	**65.5**

Table 4.1: Recognition accuracy on Caltech 101 dataset using a variety of different feature representations and two different classifiers. The PCA + linear SVM classifier is similar to (Pinto et al., 2008), while the Spatial Pyramid Matching Kernel SVM classifier is that of (Lazebnik et al., 2006a). IPSD is used to extract features with three different sampling step sizes over an input image to produce 34x34, 56x56 and 120x120 feature maps, where each feature is 128 dimensional to be comparable to SIFT. Local normalization is **not** applied on SIFT features when used with Spatial Pyramid Match Kernel SVM.

Performance on Tiny Images Dataset	
Method	**Accuracy** (%)
IPSD (5x5)	**54**
SIFT (5x5) (non rot. inv.)	**53**
Performance on MNIST Dataset	
Method	**Error Rate** (%)
IPSD (5x5)	**1.0**
SIFT (5x5) (non rot. inv.)	**1.5**

Table 4.2: Results of recognition error rate on Tiny Images and MNIST datasets. In both setups, a 128 dimensional feature vector is obtained using either our method or SIFT over a regularly spaced 5x5 grid and afterwards a linear SVM is used for classification. For comparison purposes it is worth mentioning that a Gaussian SVM trained on MNIST images without any preprocessing achieves 1.4% error rate.

5
DEEP NETWORKS

A hierarchical feature extractor can be trained layer-by-layer using any of the unsupervised algorithms described in chapter 3 and 4, similarly to what was proposed by Hinton and collaborators (Hinton et al., 2006; Hinton and Salakhutdinov, 2006) for training deep belief networks. Indeed, the underlying principle is very simple and it consists of initializing the parameters of each layer by using an unsupervised algorithm, like an RBM or an auto-encoder neural network (Bengio et al., 2007; Ranzato et al., 2007b), and then to optimize the whole system by supervised gradient descent. The chain of non-linear layers can be used for feature extraction, or it can be topped by a supervised classifier for recognition. Recent works (Weston et al., 2008; Collobert and Weston, 2008; Ahmed et al., 2008) proposed to replace this two-steps training procedure with a more integrated one, where the whole system is optimized from random initial conditions, but auxiliary prediction tasks are added to the loss function, *de facto* injecting more gradients both at the top and at the internal states of the network to improve optimization and generalization.

In this chapter we report some experiments using deep networks. We demonstrate these methods on three very different datasets in order to show the ability of the learning algorithm to adapt to different domains. We report results on recognition of handwritten digits (Ranzato et al., 2006; Ranzato et al., 2007b; Ranzato et al., 2007c), on recognition of generic natural object categories (Ranzato et al., 2007c), and on classification and retrieval of text documents (Ranzato and Szummer, 2008).

5.1 Digit Recognition

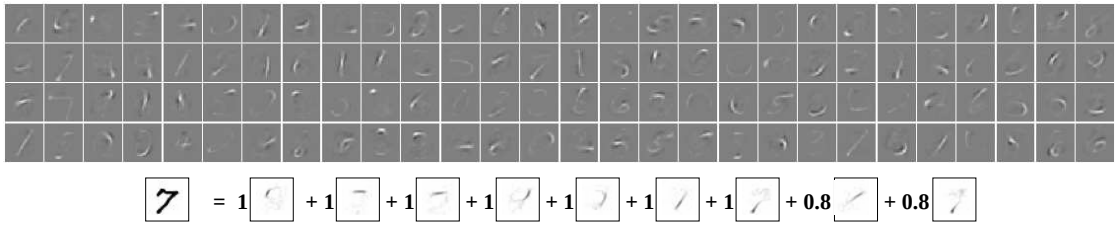


Figure 5.1: Top: A randomly selected subset of encoder filters learned by a sparse coding algorithm (Ranzato et al., 2006) similar to the one presented in chapter 3, when trained on the MNIST handwritten digit dataset. Bottom: An example of reconstruction of a digit randomly extracted from the test data set. The reconstruction is made by adding "parts": it is the *additive* linear combination of few basis functions of the decoder with positive coefficients.

The MNIST dataset (MNI,) of handwritten digits has 60,000 samples in the training set, and 10,000 samples in the test set. Digits are quasi-binary images of size 28×28 pixels (see fig. 4.7). The images are pre-processed by dividing each pixel value by 255.

In the experiment of fig. 5.1 the sparse coding algorithm described in (Ranzato et al., 2006), which is similar to PSD but with codes that are non-negative, was trained to produced feature vectors with 196 components using 196 filters of size 28×28 (i.e., filters are not convolutional).

Each one of the filters, shown in the top part of fig. 5.1, contains an elementary "part" of a digit. Reconstruction of most digits is achieved by a linear additive combination of a small number of filters since the the code is sparse and positive. The bottom part of

fig. 5.1 illustrates this reconstruction by parts.

In the next experiment, the same sparse coding algorithm is used to initialize the first layer of a large convolutional network. We used an architecture essentially identical to *LeNet-5* as described in (LeCun et al., 1998). However, because the model produces sparse features, our network had a considerably larger number of feature maps: 50 for layer 1 and 2, 50 for layer 3 and 4, 200 for layer 5, and 10 for the output layer. The numbers for LeNet-5 were 6, 16, 100, and 10 respectively. We refer to our larger network as the 50-50-200-10 network. We trained this networks on 55,000 samples from MNIST, keeping the remaining 5,000 training samples as a validation set. When the error on the validation set reached its minimum, an additional five sweeps were performed on the training set augmented with the validation set (unless this increased the training loss). Then the learning was stopped, and the final error rate on the test set was measured. When the weights are initialized randomly, the 50-50-200-10 achieves a test error rate of 0.7%, to be compared with the 0.95% obtained by (LeCun et al., 1998) with the 6-16-100-10 network.

Next, the sparse feature learning method was trained on 5×5 image patches extracted from training images. The model had a 50-dimensional code. The encoder filters were used to initialize the first layer of the 50-50-200-10 net. The network was then trained in the usual way, except that the first layer was kept fixed for the first 10 epochs through the training set. The 50 filters after training are shown in fig. 5.2. The test error rate was 0.6%. To our knowledge, this is the best results ever reported with a method trained on the original MNIST set, without deskewing nor augmenting the training set with distorted samples.

The training set was then augmented with samples obtained by elastically distorting the original training samples, using a method similar to (Simard et al., 2003). The error rate of the 50-50-200-10 net with random initialization was 0.49% (to be compared to 0.40% reported in (Simard et al., 2003)). By initializing the first layer with the filters obtained with the proposed method, the test error rate dropped to 0.39%. While this is the best numerical result ever reported on MNIST, it is not statistically different from (Simard et al., 2003).

Figure 5.2: Filters in the first convolutional layer after training when the network is randomly initialized (top row) and when the first layer of the network is initialized with the features learned by the sparse unsupervised algorithm (bottom row).

Architecture	**Training Set Size**					
	20K		60K		60K + Distortions	
6-16-100-10 (LeCun et al., 1998)	-	-	0.95	-	0.60	-
5-50-100-10 (Simard et al., 2003)	-	-	-	-	0.40	-
50-50-200-10	1.01	**0.89**	0.70	**0.60**	0.49	**0.39**

Table 5.1: Comparison of test error rates on MNIST dataset using convolutional network architectures with various training set size: 20,000, 60,000, and 60,000 plus 550,000 elastic distortions. For each size, results are reported with randomly initialized filters, and with first-layer filters initialized using the proposed algorithm (bold face).

5.1.1 What does the top-layer represent?

Here we report an experiment that was done using yet another sparse coding algorithm, dubbed SESM (Ranzato et al., 2007b). SESM is also similar to PSD, but encoder and decoder share the same weight matrix like in RBM's. The encoder and decoder compute a weighted sum of the input followed by a logistic non-linearity. The loss function and the training algorithm are the same as in PSD.

Training SESM on the whole digits produces filters that look like digit strokes, similarly to what is shown in fig. 5.1. By using the first layer representation, we train a second stage using SESM. While the first layer representation has 200 components, the second layer representation has only 10 components. Since we aim to find a 1-of-10 code we increase the sparsity level when training the second stage machine. Despite the completely *unsupervised* training procedure, the feature detectors in the second stage machine look like digit prototypes, as can be seen in fig. 5.3. The hierarchical unsupervised feature extractor is able to capture higher order correlations among the input pixel intensities, and to discover the highly non-linear mapping from raw pixel data to the class labels. While the first layer captures local correlations among the input variables (strokes), the second layer models longer range dependencies by putting together the strokes that frequently occur together.

Changing the random initialization can sometimes lead to the discover of two different shapes of "9" without a unit encoding the "4", for instance. Nevertheless, results are qualitatively very similar to this one. For comparison, when training a two-layers DBN, prototypes are not recovered because the learned code is distributed among units.

Figure 5.3: Back-projection in image space of the filters learned in the second stage of the hierarchical feature extractor. The second stage was trained on the non linearly transformed codes produced by the first stage machine. The back-projection has been performed by using a 1-of-10 code in the second stage machine, and propagating this through the second stage decoder and first stage decoder. The filters at the second stage discover the class-prototypes (manually ordered for visual convenience) even though no class label was ever used during training.

5.1.2 Using Sparse and Locally Shift Invariant Features

In this section we report experiments using the sparse and locally-shift invariant feature extractor described in sec. 4.1. We constructed a deep network and trained it on subsets of various sizes, with three different learning procedures. In all cases the feature extraction is performed by the four bottom layers (two levels of convolution/pooling). The input is a 34×34 image obtained by evenly padding the 28×28 original image with zeros. The first layer is a convolutional layer with fifty 7×7 filters, which produces 50 feature maps of size 28×28. The second layer performs a max-pooling over 2×2 neighborhoods and outputs 50 feature maps of size 14×14. The third layer is a convolutional layer with 1,280 filters of size 5×5, that connect the subsets of the 50 layer-two feature maps to the 128 layer-three maps of size 10×10. Each layer-three feature map is connected to 10 layer-two feature maps according to a fixed, randomized connectivity

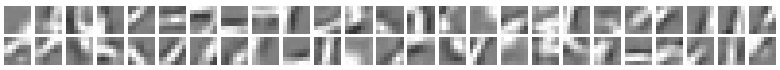


Figure 5.4: Fifty 7×7 sparse shift-invariant features learned by the unsupervised learning algorithm on the MNIST dataset. These filters are used in the first convolutional layer of the feature extractor.

table. The fourth layer performs a max-pooling over 2×2 neighborhoods and outputs 128 feature maps of size 5×5. The layer-four representation has $128 \times 5 \times 5 = 3,200$ components that are fed to a two-layer neural net with 200 hidden units, and 10 output units (one per class). There is a total of about 10^5 trainable parameters in this network.

The **first training procedure** trains the four bottom layers of the network unsupervised over the whole MNIST dataset, following the method presented in the previous sections. In particular the first stage module was learned using 100,000 8×8 patches extracted from the whole training dataset (see fig.5.4), while the second stage module was trained on 100,000 50×6×6 patches produced by the first stage extractor. The second-stage features are receptive fields of size 18×18 when backprojected on the input. In both cases, these are the smallest patches that can be reconstructed from the convolutional and max-pooling layers. Nothing prevents us from using larger patches if so desired. The top two layers are then trained supervised with features extracted from the labeled training subset. The **second training procedure** initializes the whole network randomly, and trains supervised the parameters in all layers using the labeled samples in the subset. The **third training procedure** randomly initializes the parameters in both stages of the feature extractor, and only trains (in supervised mode) the top two layers

on the samples in the current labeled subset, using the features generated by the feature extractor *with random filters*.

For the supervised portion of the training, we used labeled subsets of various sizes, from 300 up to 60,000. Learning was stopped after 50 iterations for datasets of size bigger than 40,000, 100 iterations for datasets of size 10,000 to 40,000, and 150 iterations for datasets of size less than 5,000.

The results are presented in fig.5.5. For larger datasets ($>$ 10,000 samples) there is no difference between training the bottom layer unsupervised or supervised. However for smaller datasets, networks with bottom layers trained unsupervised perform consistently better than networks trained entirely supervised. Keeping the bottom layers random yields surprisingly good results (less than 1% classification error on large datasets), and outperforms supervised training of the whole network on very small datasets ($<$ 1,000 samples). This counterintuitive result shows that it might be better to freeze parameters at random initial values when the paucity of labeled data makes the system widely over-parameterized. Conversely, the good performance with random features hints that the lower-layer weights in fully supervised back-propagation do not need to change much to provide good enough features for the top layers. This might explain why overparameterization does not lead to a more dramatic collapse of performance when the whole network is trained supervised on just 30 samples per category. For comparison, the best published testing error rate when training on 300 samples is 3% (Amit and Trouve, 2005), and the best error rate when training on the whole set is 0.60% (Ranzato et al., 2006) as reported in the previous section. Note that in that case the *whole* network was fine-tuned by supervised gradient descent after the unsupervised training

stage, yielding slightly better results.

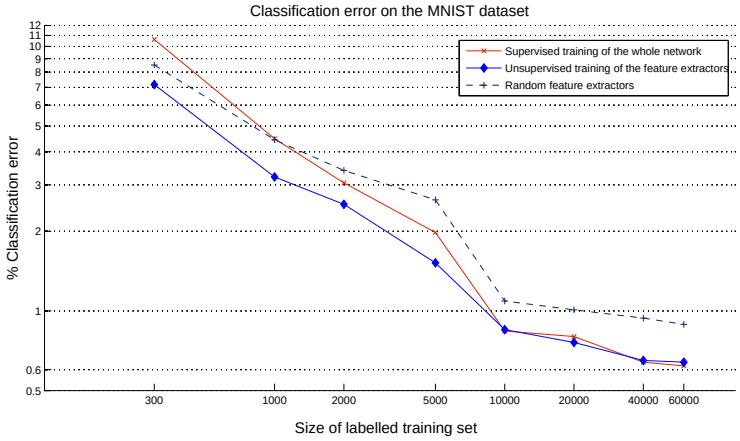
5.2 Recognition of Generic Object Categories

In this section we report another experiment using the sparse and locally shift-invariant feature extractor of sec. 4.1. We used the features to recognize object categories in the Caltech-101 dataset (Fei-Fei et al., 2004). The Caltech 101 dataset has images of 101 different object categories, plus a background category. It has various numbers of samples per category (from 31 up to 800), with a total of 9,144 samples of size roughly 300×300 pixels. The common experiment protocol adopted in the literature is to take 30 images from each category for training, use the rest for testing, and measure the average recognition rate per class.

This dataset is particularly challenging for learning-based systems, because the number of training sample per category is exceedingly small. An end-to-end supervised classifier such as a convolutional network would need a much larger number of training samples per category, lest over-fitting would occur. In the following experiment, we demonstrate that extracting features with the proposed unsupervised method leads to considerably higher accuracy than pure supervised training.

Before extracting features, the input images are preprocessed. They are converted to gray-scale, resized so that the longer edge is 140 pixels while maintaining the aspect ratio, high-pass filtered to remove the global lighting variations, and evenly zero-padded to a 140×140 image frame.

The feature extractor has the following architecture. In the first stage feature extractor (layer 1 and 2) there are 64 filters of size 9×9 that output 64 feature maps of size



Labeled training samples	Unsupervised training for bottom layers, supervised training for top layers	Supervised training from random initial conditions	Random bottom layers, supervised training for top layers
60,000	0.64	0.62	0.89
40,000	0.65	0.64	0.94
20,000	0.76	0.80	1.01
10,000	0.85	0.84	1.09
5,000	1.52	1.98	2.63
2,000	2.53	3.05	3.40
1,000	3.21	4.48	4.44
300	7.18	10.63	8.51

Figure 5.5: Error rate on the MNIST test set (%) when training on various number of labeled training samples. With large labeled sets, the error rate is the same whether the bottom layers are learned unsupervised or supervised. The network with random filters at bottom levels performs surprisingly well (under 1% classification error with 40K and 60K training samples). With smaller labeled sets, the error rate is lower when the bottom layers have been trained unsupervised, while pure supervised learning of the whole network is plagued by over-parameterization. Despite the large size of the network the effect of over-fitting is surprisingly limited.

132×132. The next max-pooling layer takes non overlapping 4×4 windows and outputs 64 feature maps of size 33×33. Unsupervised training was performed on 100,000 patches randomly sampled from the subset of the Caltech-256 dataset (Griffin et al., 2006) that does not overlap with the Caltech 101 dataset (the Caltech 101 categories were removed). The first stage was trained on such patches of size 12×12. The second stage of feature extraction (layer 3 and 4) has a convolutional layer which outputs 512 feature maps and has 2048 filters. Each feature map in layer 3 combines 4 of the 64 layer-2 feature maps. These 4 feature maps are picked at random. Layer 4 is a max-pooling layer with 5×5 windows. The output of layer 4 has 512 feature maps of size 5×5. This second stage was trained unsupervised on 20,000 samples of size $64 \times 13 \times 13$ produced by the first stage feature extractor. Note that the top level representation is invariant in windows of approximate size 20×20 pixels in input space, because of the combined effect of pooling at the first and second stage. Example of learned filters are shown in fig. 5.6.

After the feature extractor is trained, it is used to extract features on a randomly picked Caltech-101 training set with 30 samples per category. To test how a baseline classifier fares on these 512×5×5 features, we applied a k-*nearest neighbor* classifier which yielded about 20% overall average recognition rate for $k = 5$.

Next, we trained an SVM with Gaussian kernels in the one-versus-others fashion for multi-class classification. The overall recognition system is shown in fig. 5.7. The two parameters of the SVM's, the Gaussian kernel width γ^{-1} and the softness C, are tuned with cross validation, with 10 out of 30 samples per category used as the validation set. The parameters with the best validation performance, $\gamma = 5.6 \cdot 10^{-7}$, $C = 2.1 \cdot$

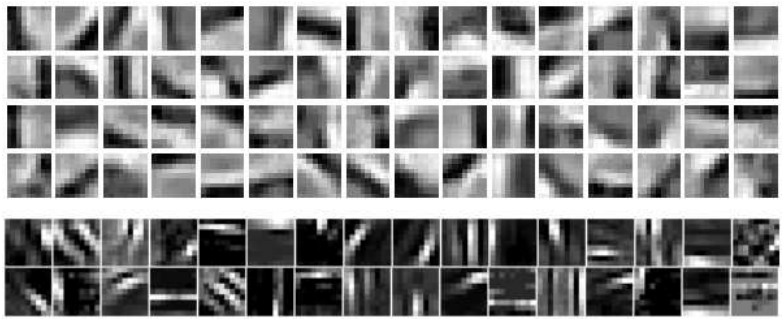


Figure 5.6: Caltech 101 feature extraction. Top Panel: the 64 convolutional filters of size 9×9 learned by the first stage of the invariant feature extraction. Bottom Panel: a selection of 32 (out of 2048) randomly chosen filters learned in the second stage of invariant feature extraction.

10^3, were used to train the SVM. More than 90% of the training samples are retained as support vectors. This is an indication of the complexity of the classification task due to the small number of training samples and the large number of categories. We report the average result over 8 independent runs, in each of which 30 images of each category were randomly selected for training and the rest were used for testing. The average recognition rate over all 102 categories is **54%**($\pm$ 1%). Examples of images and recognition rates on a few categories are given in fig. 5.8.

For comparison, we trained an essentially identical architecture in supervised mode using back-propagation (except the penultimate layer was a traditional dot-product and sigmoid layer with 200 units instead of a layer of Gaussian kernels). Supervised training from a random initial condition over the whole net achieves 100% accuracy on the training dataset (30 samples per category), but only 20% average recognition rate on the

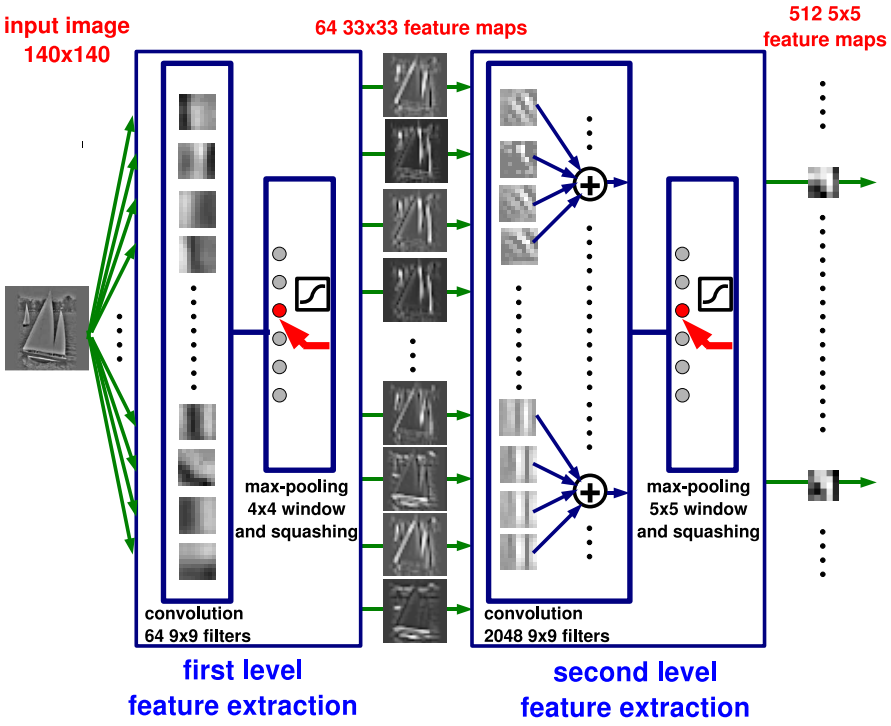


Figure 5.7: Example of the computational steps involved in the generation of two 5×5 shift-invariant feature maps from a pre-processed image in the Caltech101 dataset. Filters and feature maps are those actually produced by our algorithm.

test set. This is only marginally better than the simplest baseline systems (Fei-Fei et al., 2004; Berg et al., 2005), and considerably worse than the above result.

In our experiment, the categories that have the lowest recognition rates are the background class and some of the animal categories (wild cat, cougar, beaver, crocodile), consistent with the results reported in (Lazebnik et al., 2006b) (their experiment did not include the background class).

Our performance is similar to that of similar multi-stage Hubel-Wiesel type architec-

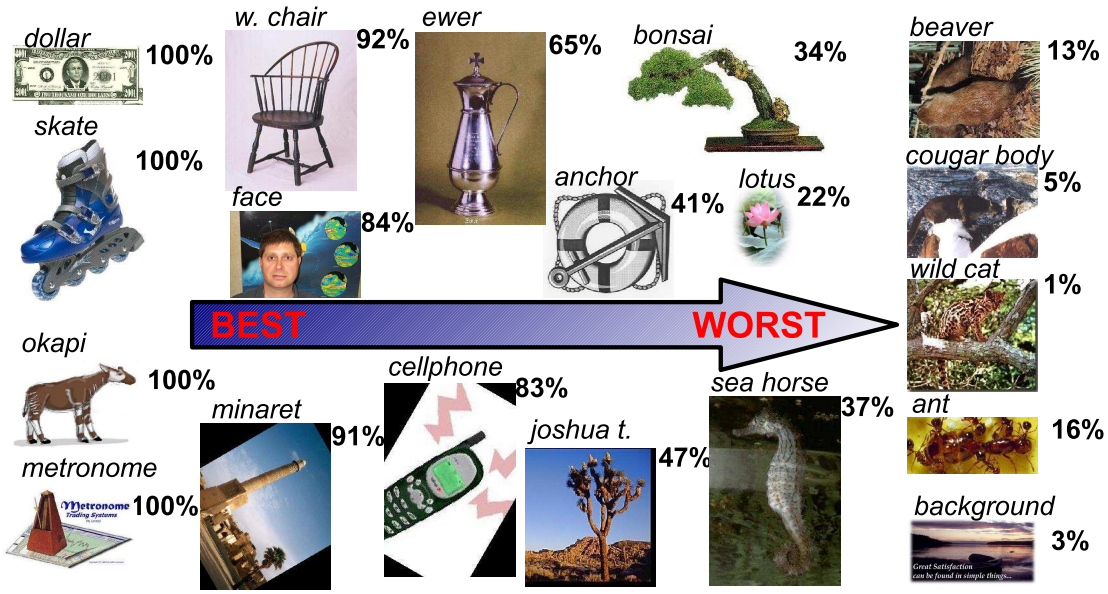


Figure 5.8: Recognition accuracy on some object categories of the Caltech 101 dataset. The system is more accurate when the object category has little variability in appearance, limited occlusion and plain background.

tures composed of alternated layers of filters and max pooling layers. Serre et al. (Serre et al., 2005) achieved an average accuracy of 42%, while Mutch and Lowe (Mutch and Lowe, 2006) improved it to 56%. Our system is smaller than those models, and does not include feature pooling over scale. It would be reasonable to expect an improvement in accuracy if pooling over scale were used. More importantly, our model has several advantages. First, our model uses no prior knowledge about the specific dataset. Because the features are learned, it applies equally well to natural images and to digit images (and possibly other types). This is quite unlike the systems in (Serre et al., 2005; Mutch and Lowe, 2006) which use fixed Gabor filters at the first layer. Second, using trainable filters at the second layer allows us to get away with only 512 feature maps. This is to be compared to Serre et al's 15,000 and Mutch et al's 1,500.

For reference, the best reported performance of 66.2% on this dataset was reported by Zhang et al. (Zhang et al., 2006), who used a geometric blur local descriptor on interest points, and matching distance for a combined nearest neighbor and SVM. Lazebnik et al. (Lazebnik et al., 2006b) report 64.6% by matching multi-resolution histogram pyramids on SIFT. While such carefully engineered methods have an advantage with very small training set sizes, we can expect this advantage to be reduced or disappear as larger training sets become available. As evidence for this, the error rate reported by Zhang et al. on MNIST with 10,000 training samples is over 1.6%, twice our 0.84% on the same, and considerably more than our 0.64% with the full training set.

Our method is very time efficient in recognition. The feature extraction is a feed-forward computation with about $2 \cdot 10^8$ multiply-add operations for a 140×140 image and 10^9 for 320×240. Classifying a feature vector with the Caltech-101 SVM takes

another $4 \cdot 10^7$ operations. An optimized implementation of our system could be run on a modern PC at several frames per second.

5.3 Text Classification and Retrieval

Document representations are a key ingredient in all information retrieval and processing systems. The goal of the representation is to make certain aspects of the document readily accessible, e.g. the document topic. To identify a document topic, we cannot rely on specific words in the document, as it may use other synonymous words or misspellings. Likewise, the presence of a word does not warrant that the document is related to it, as it may be taken out of context, or polysemous, or unimportant to the document topic.

The most widespread representations for document classification and retrieval today are based on a vector of counts. These include various term-weighting retrieval schemes, such as tf-idf and BM25 (Robertson and Walker, 1994), and bag-of-words generative models such as naive Bayes text classifiers. The pertinent feature of these representations is that they represent individual words. A serious drawback of the basic tf-idf and BM25 representations is that all dimensions are treated as independent, whereas in reality word occurrences are highly correlated.

There have been many attempts at modeling word correlations by rotating the vector space and projecting documents onto principal axes that expose related words. Methods include LSI (Deerwester et al., 1990) and pLSI (Hofmann, 1999). These methods constitute a linear re-mapping of the original vector space, and while an improvement, still can only capture very limited relations between words. As a result they need a large

number of projections in order to give an appropriate representation.

Other models, such as LDA (Blei et al., 2003), have shown superior performance over pLSI and LSI. However, inferring the representation is computationally expensive because of the "explaining away" effect that plagues all directed graphical models.

More recently, a number of authors have proposed undirected graphical models that can make inference efficient at the cost of more complex learning due to a global (rather than local) partition function whose exact gradient is intractable. These models build on RBM's by adapting the conditional distribution of the input visible units to model discrete counts of words (Hinton and Salakhutdinov, 2006; Gehler et al., 2006; Salakhutdinov and Hinton, 2007a; Salakhutdinov and Hinton, 2007b). These models have shown state-of-the-art performance in retrieval and clustering, and can be easily used as a building block for deep multi-layer networks (Hinton et al., 2006). This might allow the top-level representation to capture high-order correlations that would be difficult to efficiently represent with similar but shallow models (Bengio and LeCun, 2007).

Seeking an algorithm that can be trained efficiently, and that can produce a representation with just a few matrix multiplications, we propose a deep network whose building blocks are autoencoders, with a specially designed first layer for modeling discrete counts of words.

Previously, deep networks have been trained either from fully labeled data, or purely unlabeled data. Neither method is ideal, as it is expensive to label large collections, whereas purely unsupervised learning may not capture the relevant class information in the data. Inspired by the experiments by Bengio et al. (Bengio et al., 2007), we learn the parameters of the model by using *both a supervised and an unsupervised objective*. In

other words, we require the representation to produce good reconstructions of the input documents and, at the same time, to give good predictions of the document class labels. Besides demonstrating better accuracy in retrieval, we also extend the deep network framework to a *semi-supervised* setting where we deal with partially labeled collections of documents. This allows us to use relatively few labeled documents yet leverage language structure learned from large corpora (Ranzato and Szummer, 2008).

5.3.1 Modelling Text

The input to the system is a bag of words representation of each text document in the form of a count vector. The length of the vector equals the number of unique words in the collection, and its i-th entry stores the number of times the corresponding word occurs in the document. The goal of the system is to extract a *compact* representation from this very high-dimensional but sparse input vector. A compact representation is good because it requires less storage, and allows fast index lookup.

Since we want to extract compact representations, we use at each layer a simple auto-encoder neural network compressing the input into a code with fewer units. The auto-encoder is trained by minimizing the square distance between input and output of the network. The encoder computes a weighted sum of the input followed by a logistic non-linearity, while the decoder is linear. The only exception is the auto-encoder at the first stage that has to predict a vector of positive word counts. For this stage, we use a Poisson regressor in the decoder, that is, we exponentiate the weighted sum produced by the decoder and treat it as the mean firing rate of the Poisson distribution. The parameters of the model are trained by minimizing this unsupervised objective and also

the supervised error term coming from the prediction of the input document topic. The supervised error term is simply the cross-entropy between network prediction and target vector encoding the class of the training sample.

More formally, at the first stage the encoder computes:

$$Z = \sigma(W_e \log(X + 1) + b_e) \tag{5.1}$$

where X is the input vector of counts, Z is the representation at the first stage, W_e and b_e are the weight matrix and the bias of the first stage encoder, and σ is the logistic non-linearity. This encoder tries to mirror the computations done in the decoder. The decoder at the first stage computes a reconstruction by taking $e^{W_d Z + b_d}$, where W_d and b_d are the weight matrix and the bias of the decoder. In the upper stages, the encoder computes the code as $Z = \sigma(W_e X + b_e)$, and the decoder computes the reconstruction as $W_d Z + b_d$.

The layer-wise training is done by minimizing a loss that is the weighted sum of the reconstruction error and the classification error $L = E_R + \alpha E_C$, where the hyper-parameter α is set to zero if the sample does not have a label. Denoting with $(W_C)_i$ the i-th row of the classifier weight matrix, with b_{Ci} the i-th bias of the classifier, and with H_j the j-th output unit of the classifier passed through a soft-max:

$$H_j = \frac{\exp((W_C)_j \cdot Z + b_{Cj})}{\sum_i \exp((W_C)_i \cdot Z + b_{Ci})}, \tag{5.2}$$

we define $E_C = -\sum_i Y_i \log H_i$, where Y is a 1-of-N encoding of the target class label. Finally, in the upper layers the reconstruction error is:

$$E_R = \|X - W_d \sigma(W_e X + b_e) - b_d\|_2^2, \tag{5.3}$$

while at the first layer it is derived from minus the log probability of the data under the Poisson model (instead of using a Gaussian model as before):

$$E_R = \sum_i (\beta e^{((W_d)_i \cdot Z + b_{di})} - X_i (W_d)_i \cdot Z - X_i b_{di} + \log X_i!), \tag{5.4}$$

where β is a constant proportional to the document length, and Z is the encoder output.

Since the layer-wise training takes already into account the labels of the documents (if available), no global "fine-tuning" of the whole system is necessary according to our experiments. This saves a lot of time because it is expensive to do forward and backward propagation through a large and deep network. In the following experiments, the deep network is trained layer-by-layer by stochastic gradient descent.

5.3.2 Experiments

In our experiments we considered three standard datasets: 20 Newsgroups, Reuters-21578, and Ohsumed[1]. The 20 Newsgroups dataset contains 18845 postings taken from the Usenet newsgroup collection. Documents are partitioned into 20 topics. The dataset is split into 11314 training documents and 7531 test documents. Training and test articles are separated in time. Reuters has a predefined ModApte split of the data into 11413 training documents and 4024 test documents. Documents belong to one of 91 topics. The Ohsumed dataset has 34389 documents with 30689 words and each document might be assigned to more than one topic, for a total of 23 topics. The dataset is split into training and test by randomly selecting the 67% and the 33% of the data.

[1]These corpora were downloaded from `http://people.csail.mit.edu/jrennie/20Newsgroups`, and `http://www.kyb.mpg.de/bs/people/pgehler/rap`

Rainbow[2] was used to pre-process these datasets by stemming the documents, removing stop words and words appearing less than three times or in only a single document, and retaining between 1000 and 30,000 words with the highest mutual information.

Unless stated otherwise, we trained each layer of the network for only 4 epochs over the whole training dataset. Convergence took only a couple of epochs, and was robust to the choice of the learning rate. This was set to about 10^{-4} when training the first layer, and to 10^{-3} when training the layers above. The learning rate was exponentially decreased by multiplying it by 0.97 every 1000 samples. A small L1 regularizer on the parameters was added to the loss. Each weight was randomly initialized, and was updated by taking a gradient step with a regularizer given by the value of the learning rate times $5 \cdot 10^{-4}$ the sign of the weight. The value of α_c in eq. 3.2 was set to the ratio between the number of input units in the layer and the number of classes in order to make the two error terms E_R and E_C comparable. Its exact value did not affect the performance as long as it had the right order of magnitude.

The Value of Labels

In order to assess whether semi-supervised training was better than purely unsupervised training, we trained the deep model on the 20 Newsgroup dataset using only 2, 5, 10, 20 and 50 samples per class. During training we showed the system 10 labeled samples every 100 examples by sweeping more often over the labeled data. This procedure was repeated at each layer during training. We trained 4 layers for 10 epochs with an architecture of 2000-200-100-50-20, denoting 2000 inputs, 200 hidden units at the first layer, 100 at the second, 50 at the third, and 20 at the fourth. Then, we trained a Support

[2]Rainbow is available at `http://www.cs.cmu.edu/~mccallum/bow/rainbow`

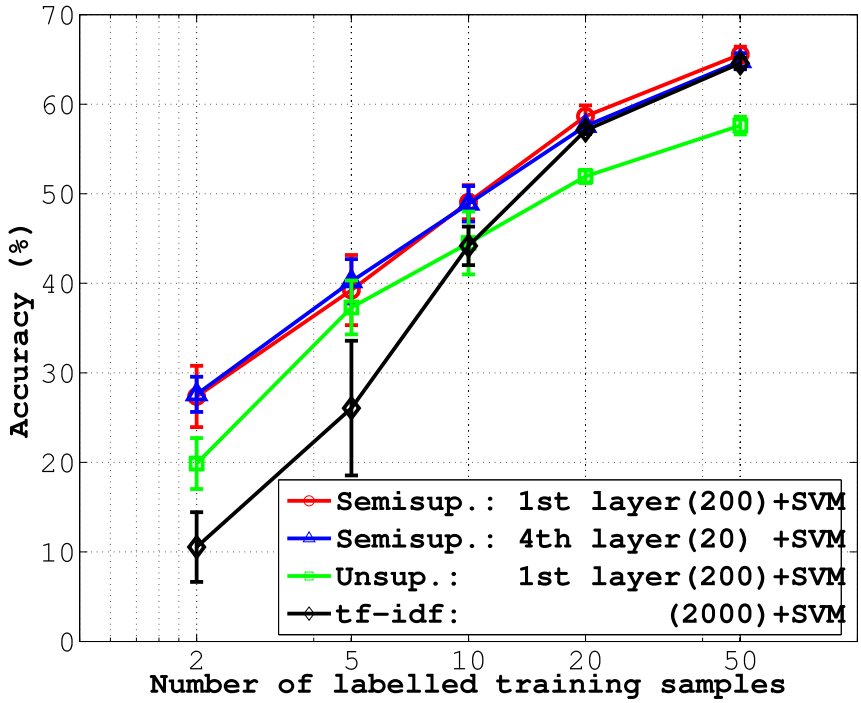


Figure 5.9: SVM classification of documents from the 20 Newsgroups dataset (2000 word vocabulary) trained with between 2 and 50 labeled samples per class. The SVM was applied to representations from the deep model trained in a semi-supervised or unsupervised way, and to the tf-idf representation. The numbers in parentheses denote the number of code units. Error bars indicate one standard deviation. The fourth layer representation has only 20 units, and is much more compact and computationally efficient than all the other representations.

Vector Machine[3] (SVM) with a Gaussian kernel on (1) the codes that corresponded to the labeled documents, and we compared the accuracy of the semi-supervised model to the one achieved by a Gaussian SVM trained on the features produced by (2) the same model but trained in an unsupervised way, and by (3) the tf-idf representation of the

[3]We used libsvm package available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`

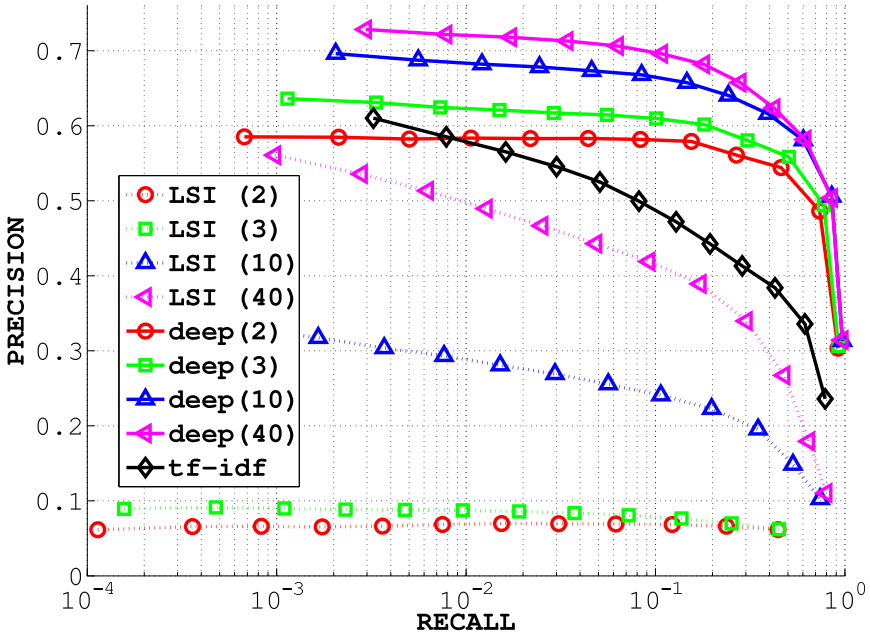


Figure 5.10: Precision-recall curves for the Reuters dataset comparing a linear model (LSI) to the nonlinear deep model with the same number of code units (in parentheses). Retrieval is done using the k most similar documents according to cosine similarity, with $k \in [1 \ldots 4095]$.

same labeled documents. The SVM was generally tuned by five-fold cross validation on the available labeled samples (but two-fold cross validation when using only two samples per class). Fig. 5.9 demonstrates that the learned features gave much better accuracy than the tf-idf representation overall when labeled data was scarce. The model was able to exploit the very few labeled samples producing features that were easier to discriminate. The performance actually improved when the dimensionality of the code was reduced and only 2 or 5 labeled samples per class were available, probably because a more compact code implicitly enforces a stronger regularization. Semi-supervised

training outperformed unsupervised training, and the gap widened as we increased the number of labeled samples, indicating that the unsupervised method had failed to model information relevant for classification when compressing to a low-dimensional space.

Interestingly, if we classify the data using the classifier of the feedback module we obtain a performance similar to the one achieved by the Gaussian SVM. For example, when *all* training samples are labeled the classifier at the first stage achieves accuracy of 76.3% (as opposed to 75.5% of the SVM trained either on the learned representation or on tf-idf), while the one on the fourth layer achieves accuracy of 74.8%. Hence, the training algorithm provides an accurate classifier as a side product of the training, reducing the overall learning time.

Deep or Shallow?

In all the experiments discussed in this section the model was trained using fully labeled data (still, training also includes an unsupervised objective as discussed earlier). In order to retrieve documents after training the model, all documents are mapped into the latent low-dimensional space, the cosine similarity between each document in the test dataset and each document in the training dataset is measured, and the k most similar documents are retrieved. k is chosen to be equal to 1, 3, 7, ..., 4095. Based on the topic label of the documents, we assess the performance by computing the *recall* and the *precision* averaged over the whole test dataset.

In the first experiment, we compared the linear mapping produced by LSI to the nonlinear mapping produced by our model. We considered the Reuters dataset with a 12317 word vocabulary and trained a network with 3 layers. The first layer had 100 code units, the second layer had 40 units in one experiment and 10 in another, the third

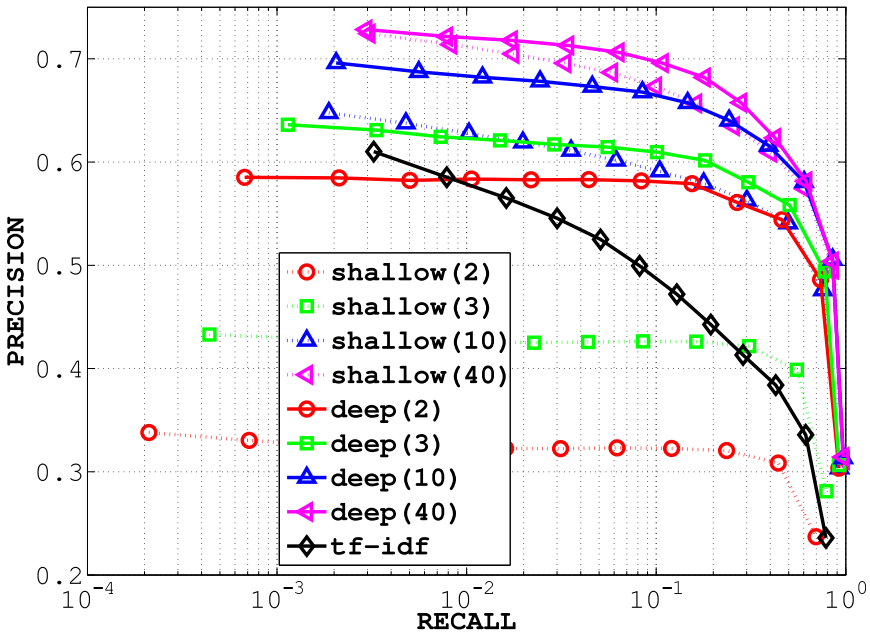


Figure 5.11: Precision-recall curves for the Reuters dataset comparing shallow models (one-layer) to deep models with the same number of code units. The deep models are more accurate overall when the codes are extremely compact. This also suggests that the number of hidden units has to be *gradually* decreased from layer to layer.

layer was trained with either 3 or 2 code units. As shown in Fig. 5.10, the nonlinear representation is more powerful than the linear one, when the representation is very compact.

Another interesting question is whether adding layers is useful. Fig. 5.11 shows that for a given dimensionality of the output latent space the deep architecture outperforms the shallow one. The deep architecture is capable of capturing more complex dependencies among the input variables than the shallow one, while the representation remains compact. The compactness allows us to efficiently handle very large vocabularies (more

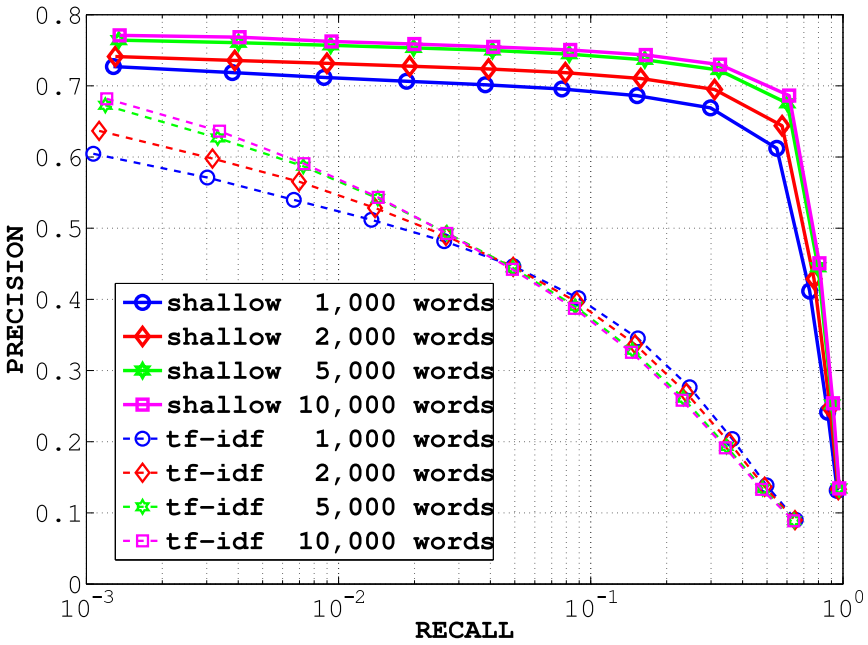


Figure 5.12: Precision-recall curves for the 20 Newsgroups dataset comparing the performance of tf-idf versus a one-layer shallow model with 200 code units for varying sizes of the word dictionary (from 1000 to 10000 words).

than 30,000 words for the Ohsumed, for instance). Fig. 5.12 shows that increasing the number of words (i.e. the dimensionality of the input) does give better retrieval performance.

Compact or Binary High-Dimensional?

The most popular representation of documents is tf-idf, a very high-dimensional and sparse representation. One might wonder whether we should learn a high-dimensional representation instead of a compact representation. Unfortunately, the autoencoder based learning algorithm forces us to map data into a lower-dimensional space at each layer, as without additional constraints the trivial identity function would be learned. We used the

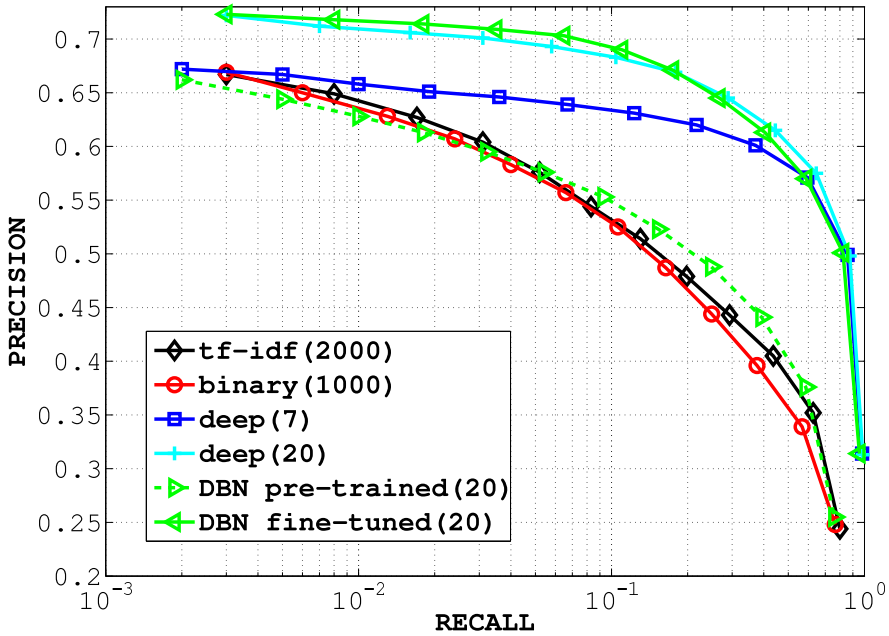


Figure 5.13: Precision-recall curves comparing compact representations vs. high-dimensional binary representations. Compact representations can achieve better performance using less memory and CPU time.

sparse encoding symmetric machine (SESM) (Ranzato et al., 2007b) as a building block for training a deep network producing sparse features. SESM is a symmetric autoencoder with a sparsity constraint on the representation, and it is trained without labels. In order to make the sparse representation at the final layer computationally appealing we thresholded it to make it binary. We trained a 2000-1000-1000 SESM network on the Reuters dataset. In order to make a fair comparison with our compact representation, we fixed the information content of the code in terms of precision[4] at $k = 1$. We

[4]The entropy of the representation would be more natural, but its value depends on the quantization level.

measured the precision and recall of the binary representation of a test document by computing its Hamming distance from the representation of the training documents. We then trained our model with the following number of units 2000-200-100-7. The last number of units was set to match the precision of the binary representation at $k = 1$. Fig. 5.13 shows that our compact representation outperforms the high-dimensional and binary representation at higher values of k. Just 7 continuous units are able to achieve better retrieval than 1000 binary units[5]! Storing the Reuters dataset with the compact representation takes less than half the memory space than using the binary representation, and comparing a test document against the whole training dataset is five times faster with the compact representation. The best accuracy for our model is given with a 20-unit representation. Fig. 5.13 shows the performance of a representation with the same number of units learned by a deep belief network (DBN) following Salakhutdinov and Hinton's constrained Poisson model (Salakhutdinov and Hinton, 2007b). Their model was greedily pre-trained for one epoch in an unsupervised way (200 pre-training epochs gave similar fine-tuned accuracy), and then fine-tuned with supervision for 100 epochs. While fine-tuning does not help our model, it significantly improves the DBN which eventually achieves the same accuracy as our model. Despite the similar accuracy, the computational cost of training a DBN (with our implementation using conjugate gradient on mini-batches) is several times higher due to this supervised training through a large and deep network. By looking at how words are mapped to the top-level feature space, we can get an intuition about the learned mapping. For instance, the code closest

[5]Note that the binarization has been achieved by thresholding the quasi-binary codes produced by SESM and this was not taken into account while training SESM. Therefore, the comparison is a bit unfavorable to the binary codes.

Table 5.2: Neighboring word stems for the model trained on Reuters. The number of units is 2000-200-100-7.

Word stem	**Neighboring word stems**
livestock	beef, meat, pork, cattle
lend	rate, debt, bond, downgrad
acquisit	merger, stake, takeov
port	ship, port, vessel, freight
branch	stake, merger, takeov, acquisit
plantat	coffe, cocoa, rubber, palm
barrel	oil, crude, opec, refineri
subcommitte	bill, trade, bond, committe
coconut	soybean, wheat, corn, grain
meat	beef, pork, cattl, hog
ghana	cocoa, buffer, coffe, icco
varieti	wheat, grain, agricultur, crop
warship	ship, freight, vessel, tanker
edibl	beef, pork, meat, poultri

to the representation of the word “jakarta” corresponds to the word “indonesia”, similarly,“meat” is closest to “beef” (table 5.2). As expected, the model implicitly clusters synonymous and related words.

Visualization

The deep model can also be used to visualize documents. When the top layer is two-dimensional we can visualize high-dimensional nonlinear manifolds in the space of bags of words. Fig. 5.14 shows how documents in the Ohsumed test set are mapped to the

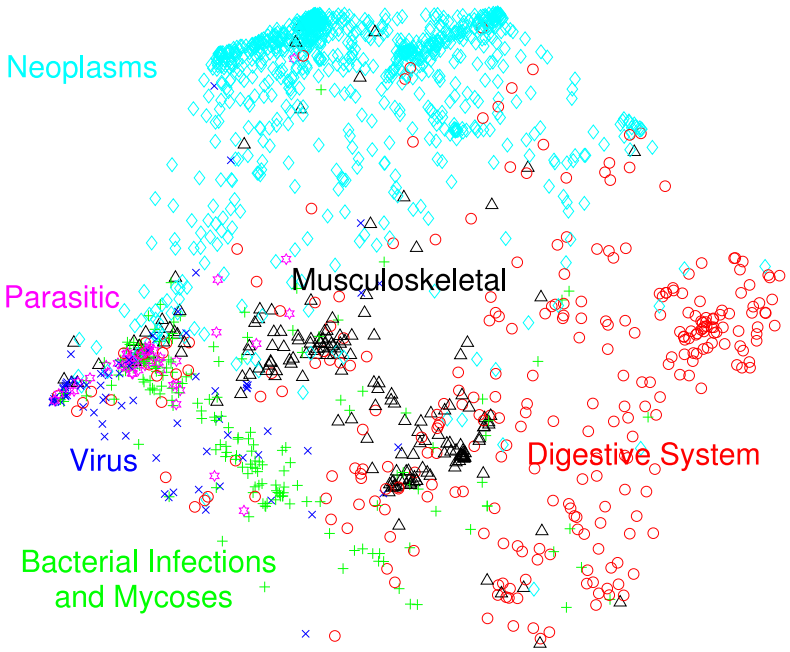


Figure 5.14: Two-dimensional codes produced by the deep model 30689-100-10-5-2 trained on the Ohsumed dataset (only the 6 most numerous classes are shown). The codes result from propagating documents in the test set through the four-layer network.

plane. The model exposes clusters of documents according to the topic class, and places similar topics next to each other. The dimensionality reduction is extreme in this case, from more than 30000 to 2.

CONCLUSION

The first contribution of this work is to introduce and develop the Energy-Based Model framework for unsupervised learning. This framework allows to view most unsupervised algorithms as pursuing the same "energy carving" task, that is, making the energy lower around areas of high data density. This bridges the conceptual gap between methods that maximize the data likelihood and methods that enforce constraints on the internal representation. In particular, it interprets sparsity as a particular way to constrain the code and regularize the learning process.

The second theme of this thesis is efficient inference. The models we propose are not very different from generative probabilistic models since we fit the data by reconstructing from an internal representation. However, inference is made efficient by training a direct mapping from input to latent representation. Generative models have straightforward interpretation, but are very expensive to use because inference consists of "inverting" the model, often requiring the use of iterative optimization procedures. We propose to train a feed-forward function to predict the latent code, and we suggest simple ways to jointly train this with the generative part of the model.

The main topic of this work has been how to train and apply deep networks. We have demonstrated these methods on a variety of tasks with experiments that gave intuitions about the encoding produced by these hierarchical models, as well as the importance to add non-linear layers to represent higher order dependencies among the input variables. These models can be trained in many different ways. We introduced several unsuper-

vised algorithms to train each layer in sequence. These algorithms produce compact representations, or even sparse overcomplete representations. Simple extensions allows the learning of representations that are invariant to either known or learned transformations. Learning invariant feature hierarchies is crucial to build a robust recognition system in vision, for instance. In this work, we pointed the reader to other training procedures that learn the parameters in an end-to-end fashion by adding additional constraints on the internal representations. This seems a more efficient strategy than the proposed layer-wise training, but it is avenue of future work to compare them and to devise even better strategies by exploiting semi-supervised and multi-task learning principles.

Another interesting question is how to exploit the feed-back connections we use during training, in order to improve recognition. The accuracy of recognition systems is poor when the input is corrupted by noise. For instance, visual object recognition is not very successful when objects are occluded. It seems conceivable that by using feed-back connections the network could fill-in the missing information and improve its prediction. Also, feed-back connections could be used in vision applications to implement attentional mechanisms, allowing to process higher resolution images.

Also, the quest for efficient coding has ample avenue for future investigation. In this work, we used sparsity as a mechanism to encourage uncorrelation, if not independence, among the units in the latent code. However, recent advances on models for lateral inhibition and pooling in computational neural science might suggest better ways to achieve coding efficiency in these bottom-up hierarchical models.

A
VARIATIONAL INTERPRETATION

In this section we show that a Gaussian variational approximation to the posterior reduces to a Laplace approximation in a sparse coding model like (Olshausen and Field, 1997). Under the assumption that the encoder of PSD is trained after the decoder has been optimized, the encoder is a least square approximation to the MAP estimate of the latent code, and it can then be interpreted as an approximation to the variational mean.

The results in this sections are used to justify theorem 1.1 as well as to derive architectures of encoders in appendix B.

A.1 The Fixed Point Solution of Lasso

Consider the problem of minimizing the following loss with respect to Z:

$$L(Z) \quad = \quad \frac{1}{2}\|Y - W_d Z\|^2 + \frac{\sigma_Y^2}{\lambda}\sum_{i=1}^{N}|z_i| \tag{A.1}$$

This loss is convex, but non-quadratic. The fixed point of this equation can be found by setting to zero the derivatives. The solution for the i-th component can be written as follows:

$$\begin{aligned} P_i &= \text{col}_i(W_d)\cdot(Y - \sum_{k\neq i}\text{col}_k(W_d)z_k) \\ z_i &= \frac{\{|P_i| - \frac{\sigma_Y^2}{\lambda}\}^+}{\|col_i(W)\|^2}\,\text{sgn}(P_i) \end{aligned} \tag{A.2}$$

where the operator $\{x\}^+ = max(0, x)$, i.e. is the positive part operator, and $\text{col}_i(W_d)$ is the i-th column of matrix W_d.

The fixed point equations provide us with an iterative algorithm where the code unit takes a value equal to the *shrinked* projection of the residual error along the corresponding basis function. The residual is computed by taking into account all code units except the current one. These update formulas reminds those used in Gaussian belief propagation to solve a system of linear equations. The sparsity enforced on Z adds the shrinkage on these update rules; different sparsity penalties yield different shrinkage non-linearities (Rozell et al., 2008). The convergence of this iterative algorithm has been demonstrated by (Daubechies et al., 2004).

A.2 Variational Approximation to the Posterior

Here, we study a Gaussian variational approximation to the posterior distribution. We assume that (1) the parameters W_d are given, (2) the likelihood $p(Y \mid Z)$ is $N(W_d Z, \sigma_Y^2)$, (3) the code Z is factorial with each component distributed according to a 0-mean Laplace distribution $L(\lambda)$, and (4) the samples are i.i.d. (hence, we can just consider a single sample Y in the following derivation). We seek the best factorial Gaussian $q(Z) = \prod_i q(z_i)$ that approximates the posterior distribution $p(Z \mid Y)$, with $q(z_i) \in N(m_i, \sigma^2)$. First, we assume that the standard deviation σ is given, and we optimize only for the value of the mean $m_i, i \in [1 \ldots N]$. In the next section, we will also derive the optimal value of σ. We minimize the KL-divergence between the approximate distribution q and the true posterior $p(Z \mid Y)$ over the mean parameters m_i.

We have that by definition and using our assumptions:

$$
\begin{aligned}
KL(q(z), p(Z \mid Y)) &= \int \prod q(z_i) \log \frac{\Pi q(z_i)}{p(Z \mid Y)} && \text{(A.3)} \\
&= \int \prod q(z_i) \log \frac{\Pi q(z_i)}{p(Y \mid Z) p(Z)} + \mathrm{c} \\
&= -\frac{N}{2} \ln(2\pi e \sigma^2) + \frac{1}{\lambda} \sum_i \int q(z_i)|z_i| \\
&\quad - \int \Pi q(z_i) \log p(Y \mid Z) + \mathrm{c} && \text{(A.4)}
\end{aligned}
$$

Since, we have that:

$$
\int q(z_i)|z_i| = m_i(1 - 2\Phi(-\frac{m_i}{\sigma})) + 2\sigma e^{-\frac{1}{2}(\frac{m_i}{\sigma})^2} \qquad \text{(A.5)}
$$

where Φ is the c.d.f. of the normal distribution, and also,

$$
\begin{aligned}
-\int \Pi q(z_i) \log p(Y \mid Z) = \frac{1}{2\sigma_y^2} [\sum_i \|\mathrm{col}_i(W)\|_2^2 (\sigma^2 + m_i^2) - 2 \sum_i m_i (\sum_l w_{li} y_l) + \\
+2 \sum_l \sum_k \sum_{h \neq k} (w_{lk} w_{lh} m_k m_h)] + c \qquad \text{(A.6)}
\end{aligned}
$$

then, we have expanded all terms in eq A.4; in other words, we can substitute eq. A.5 and A.6 into eq. A.4. By setting to zero the derivatives w.r.t. m_i we obtain:

$$
\begin{aligned}
\frac{\partial KL}{\partial m_i} = \frac{1}{\sigma_Y^2} (\|\mathrm{col}_i(W)\|^2 m_i - \sum_l w_{li} y_l + \sum_l \sum_{k \neq i} w_{lk} w_{li} m_k) + \\
+ \frac{1}{\lambda} (1 - 2\Phi(-\frac{m_i}{\sigma}) + 2\frac{m_i}{\sigma} e^{-\frac{1}{2}(\frac{m_i}{\sigma})^2} (\frac{1}{\sqrt{2\pi}} - 1)) = 0 \qquad \text{(A.7)}
\end{aligned}
$$

Unfortunately, this equation cannot be solved in closed form because it contains both polynomials and exponentials of m_i. However, we can make three cases depending on the value of the ratio m_i/σ.

Case 1: $\frac{m_i}{\sigma} \approx 0$ In this case, we can assume that $\Phi(-\frac{m_i}{\sigma}) \approx \frac{1}{2}$, and $\exp(-\frac{1}{2}(\frac{m_i}{\sigma})^2) \approx 1$. Then, the KL divergence is minimized for that m_i such that:

$$m_i = \frac{\text{col}_i(W_d) \cdot (Y - \sum_{h \neq i} \text{col}_h(W_d) m_h)}{\|\text{col}_i(W_d)\|^2 + \frac{2}{\sigma\lambda}(\frac{1}{\sqrt{2\pi}} - 1)} \tag{A.8}$$

Case 2: $\frac{m_i}{\sigma} \gg 0$ In this case, we can assume that $\Phi(-\frac{m_i}{\sigma}) \approx 0$, and $\exp(-\frac{1}{2}(\frac{m_i}{\sigma})^2) \approx 0$. Then, the KL divergence is minimized for that m_i such that:

$$m_i = \frac{\text{col}_i(W_d) \cdot (Y - \sum_{h \neq i} \text{col}_h(W_d) m_h) - \frac{\sigma_Y^2}{\lambda}}{\|\text{col}_i(W_d)\|_2^2} \tag{A.9}$$

Case 3: $\frac{m_i}{\sigma} \ll 0$ In this case, we can assume that $\Phi(-\frac{m_i}{\sigma}) \approx 1$, and $\exp(-\frac{1}{2}(\frac{m_i}{\sigma})^2) \approx 0$. Then, the KL divergence is minimized for that m_i such that:

$$m_i = \frac{\text{col}_i(W_d) \cdot (Y - \sum_{h \neq i} \text{col}_h(W_d) m_h) + \frac{\sigma_Y^2}{\lambda}}{\|\text{col}_i(W_d)\|_2^2} \tag{A.10}$$

These equations are essentially the same as those found for the MAP estimates of eq. A.2.

A.2.1 Optimizing the Variance

In the previous section we have found that the MAP estimate and the means of the variational distribution are the same under the condition $|m_i/\sigma| \gg 0$. This might trivially say that when σ is very small the Gaussian variational distribution reduces to a delta Dirac distribution, and the best way we can approximate a distribution with a delta Dirac is by placing the delta at the *mode* of the posterior distribution, i.e. at the most likely value. However, if we allow also the standard deviation σ of the variational distribution to be learned we found that this σ does not tend to zero when $|m_i/\sigma| \gg 0$.

By setting to 0 the derivative of the KL-divergenve w.r.t. σ, and by performing very similar reasonings, we have that:

$$\sigma^2 = \frac{N\sigma_Y^2}{\sum_i \|\text{col}_i(W_d)\|^2} \tag{A.11}$$

when $|m_i/\sigma| \gg 0$. By setting the norm of the basis functions to 1, then

$$\sigma^2 = \sigma_Y^2 \tag{A.12}$$

Therefore, when $|m_i/\sigma| \gg 0$ the optimal value of the variance is constant and depends on the noise level of the observed Y. Under these conditions, the mean of the Gaussian variational distribution coincides with the MAP estimate (the mode of the posterior distribution), while the variance is equal to the variance σ_Y^2 of the noise added to the observed Y. Hence, the encoder of PSD approximates the mean of a Gaussian variational distribution approximating the posterior.

When $|m_i/\sigma| \approx 0$, we can assume that $\exp(-\frac{1}{2}(\frac{m_i}{\sigma})^2 \approx 1$ and $\Phi(-\frac{m_i}{\sigma}) \approx \frac{1}{2}$. Then, the optimal σ minimizing the KL-divergence is given by:

$$\sigma = \frac{\sigma_y^2}{\sqrt{(\frac{\sigma_y^2}{\lambda})^2 + \sigma_y^2} + \frac{\sigma_y^2}{\lambda}} \tag{A.13}$$

If we use this variational approximation to the posterior in $H(Y) = H(Y|Z) + H(Z) - H(Z|Y)$, we have that when $|m_i/\sigma| \gg 0$ decreasing λ decreases both $H(Z)$ and $H(Y)$ because $H(Y|Z)$ and $H(Z|Y)$ are fixed (entropies of Gaussian distributions whose covariance does not depend on λ). When $|m_i/\sigma| \approx 0$, decreasing $H(Z)$ does not change $H(Y)$ because the decrease is compensated by an equal decrease of $H(Z|Y)$, instead. In fact, the difference $H(Z) - H(Z|Y)$ depends on the difference $(\log\lambda - \log\sigma)$ which tends to a constant as we decrease λ. Therefore, under the above mentioned

assumptions and assuming that there is at least on code unit satisfying the condition $|m_i/\sigma| \gg 0$ we have that decreasing $H(Z)$ actually decreases $H(Y)$ too.

B

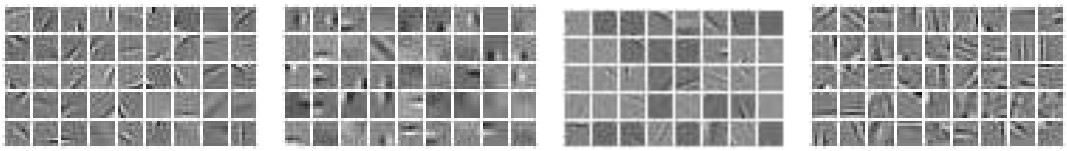
CHOOSING THE ENCODING FUNCTION

The choice of the encoding architecture is task dependent. In general, the encoder architecture as well as any other hyper-parameter of the system, like the sparsity level, have to be cross-validated.

We report experiments using PSD and other algorithms that differ from PSD mainly for the choice of the encoder architecture. The comparison is done qualitatively by visually inspecting the learned features, and quantitatively by measuring the reconstruction and the sparsity error, as well as the recognition rate on the Caltech 101 dataset using the simple recognition system described in sec. 3.3.2.

We consider the following algorithms:

1. PSD as described in sec. 3 using the output of the encoder as feature after training
2. PSD using the optimal features (i.e. those minimizing the energy) even after training
3. PSD using the output of the encoder as latent code even during training, akin to a standard autoencoder with a sparsity constraint on the internal representation
4. like 3, but the encoder is composed of a diagonal matrix followed by a linear matrix of filters (with unit norm rows) and by a thresholding non-linearity (see below)
5. like 4, but the linear matrix of filters in the encoder is the transpose of the decoder matrix of basis functions.



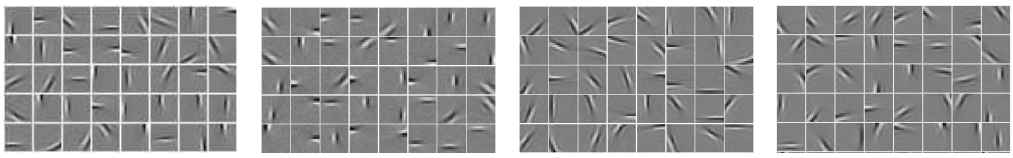
(a) (b) (c) (d)

Figure B.1: Random subset of the 405 filters of size 9x9 pixels learned in the encoder by different algorithms trained on patches from the Berkeley dataset: (a) PSD (case 1 and 2), (b) PSD without iterating for the code during training (case 3), (c) a sparse autoencoder with a thresholding non-linearity in the encoder (case 4), and (d) a sparse autoencoder with thresholding non-linearity and tied/shared weights between encoder and decoder (case 5).

By looking at the fixed point solution of lasso of eq. A.2 (i.e. inference in PSD without the code prediction error term) we can see that the non-linearity used to produce sparse representations is the *soft thresholding non-linearity*:

$$y = \begin{cases} x - \lambda & \text{if } x > \lambda \\ 0 & \text{if } -\lambda \leq x \leq \lambda \\ x + \lambda & \text{if } x < -\lambda \end{cases} \tag{B.1}$$

Unfortunately, this non-linearity does not work well in an encoder during training because the slope is zero in the interval $[\ \lambda, \lambda]$. As a result, some filters of the encoder might never be updated. Therefore in experiments number 4 and 5, (1) we also normalize the encoder filters to unit norm and (2) we consider the following smooth quadratic



(a) (b) (c) (d)

Figure B.2: Random subset of the 128 filters of size 16x16 pixels learned in the encoder by different algorithms trained on patches from the Caltech 101 pre-processed images: (a) PSD (case 1 and 2), (b) PSD without iterating for the code during training (case 3), (c) a sparse autoencoder with a thresholding non-linearity in the encoder (case 4), and (d) a sparse autoencoder with thresholding non-linearity and tied/shared weights between encoder and decoder (case 5).

approximation:

$$y = \begin{cases} x - \lambda & \text{if } x > \lambda + \alpha \\ \frac{\alpha}{(\alpha+\lambda)^2} x^2 \text{sgn}(x) & \text{if } -\lambda - \alpha \leq x \leq \lambda + \alpha \\ x + \lambda & \text{if } x < -\lambda - \alpha \end{cases} \tag{B.2}$$

where sgn is the sign function. Unlike an encoder using a hyperbolic tangent that is linear around zero, a code prediction function with a smooth thresholding non-linearity is able to produce representations that are sparser because weak activations are suppressed by this non-linearity. In our experiment, λ as well as α in eq. B.2 are set equal to the sparsity level.

In order to compute the reconstruction and the sparsity error we trained on image patches randomly sampled from the Berkeley dataset. We learn 405 filters of size 9x9 pixels. A random subset of encoder filters are shown in fig. B and the errors are reported in table B.1. The machines using the thresholding non-linearity yield sparser

Table B.1: Comparison between different encoding architectures and ways to train them. The sparsity level is set to 0.6 in all experiments, except case 5 which was set to 0.2.

Methods	SNR (rec. error) [dB]	Sparsity [L1 norm]	Recogn. Rate
1. PSD predicted codes	15.7	5.6	47%
2. PSD optimal codes	14.6	2.8	-
3. PSD not iter. for the code during training	12.9	3.4	45%
4. autoenc. with thres. non-lin.	14.3	3.1	43%
5. autoenc. with thres. non-lin., tied weights	16.1	5.1	46%

representations, comparable to those achieved by using optimal codes (case 2).

In the table we report also a recognition experiment using these different machines. We trained 128 filters of size 16x16 (see fig. B for a random subset of encoder filters) on Caltech 101 images. The pre-processing as well as the recognition system are the same as the ones described in sec. 3.3.2. Although filters might look quite different the recognition performance seems robust to the choice of the training algorithm used to learn the filters. Even though the encoder of PSD does not produce very sparse codes, it achieves the best recognition accuracy. Higher recognition rates could be achieved by (1) cross-validating the sparsity level, (2) using a better classifier (e.g. a spatial pyramid matching SVM (Lazebnik et al., 2006a)), and (3) extracting hierarchical features by using these features as intermediate representations of a deep network (Ranzato et al., 2007c).

Comparing the features learned by the machine number 3 in fig. B.1(b) and B.2(b) to the filters learned by PSD and looking at the corresponding results in table B.1, we can conclude that the minimization used to infer the code during training of PSD is crucial to improve the optimization when the representation is highly overcomplete, but

otherwise it is not necessary. In general, a better training procedure could be to start the optimization by minimizing in code space to break the symmetries and then, to complete the training by removing this extra optimization step (binding the code to the output of the decoder) to speed-up learning.

BIBLIOGRAPHY

http://yann.lecun.com/exdb/mnist/.

http://www.cs.berkeley.edu/projects/vision/grouping/segbench/.

Aharon, M., Elad, M., and Bruckstein, A. (2005). K-svd and its non-negative variant for dictionary design. In *Proc. of the SPIE conference wavelets*, volume 5914.

Ahmed, A., Yu, K., Xu, W., Gong, Y., and Xing, E. (2008). Training hierarchical feed-forward visual recognition models using transfer learning from pseudo tasks. In *European Conference on Computer Vision*.

Amit, A. and Trouve, A. (2005). Pop: Patchwork of parts models for object recognition. Technical report, The Univ. of Chicago.

Attneave, F. (1954). Some informational aspects of visual perception. *Psych. Rev.*, 61:183–193.

Barlow, H. (1961). *Possible principles underlying the transformation of sensory messages*. MIT Press, Cambridge, MA.

Bell, A. J. and Sejnowski, T. J. (1995). An information-xaximization approach to blind separation and blind deconvolution. *Neural Computation*, 7:1129–1159.

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

Bengio, Y. and LeCun, Y. (2007). Scaling learning algorithms towars ai. In Bottou, L., Chapelle, O., DeCoste, D., and Weston, J., editors, *Large-Scale Kernel Machines*. MIT Press.

Berg, A. C., Berg, T. L., and Malik, J. (2005). Shape matching and object recognition using low distortion correspondences. In *CVPR*.

Bethge, M. (2006). Factorial coding of natural images: how effective are linear models in removing higher-order dependencies? *J. Opt. Soc. Am. A*, 23(6):1253–1268.

Blei, D., Ng, A. Y., and Jordan, M. I. (2003). Latent Dirichlet allocation. *JMLR*.

Boser, B., Guyon, I., and Vapnik, V. (1992). A training algorithm for optimal margin classifiers. In *5th Annual ACM Workshop on COLT*, pages 144–152.

Cadieu, C. and Olshausen, B. (2008). Learning transformational invariants from time-varying natural images. In *NIPS*.

Carreira-Perpignan, M. A. and Hinton, G. E. (2005). On contrastive divergence learning. *Artificial Intelligence and Statistics*.

Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *International Conference on Machine Learning*, pages 160–167.

Cover, T. and Thomas, J. (1991). *Elements of information theory*. Wiley. Chap.3.

Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *Proc. of Computer Vision and Pattern Recognition*.

Daubechies, I., Defrise, M., and De Mol, C. (2004). An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Comm. Pure Appl. Math*, 57(11):1413–1457.

Deerwester, S., Dumais, S., Landauer, T., Furnas, G., and Harshman, R. (1990). Indexing by latent semantic analysis. *Journ. of American Society of Information Science*, 41:391–407.

Dempster, A., Laird, N., and D.B., R. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38.

Doi, E., Balcan, D. C., and Lewicki, M. S. (2006). A theoretical analysis of robust coding over noisy overcomplete channels. In *NIPS*. MIT Press.

Donoho, D. and Elad, M. (2003). Optimally sparse representation in general (nonorthogonal) dictionaries via ℓ^1 minimization. *Proc. Natl. Acad. Sci. USA*, 100(5):2197–2202.

Elad, M. and Aharon, M. (2006). Image denoising via learned dictionaries and sparse representation. In *CVPR*.

Fei-Fei, L., Fergus, R., and Perona, P. (2004). Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *CVPR Workshop*.

Foldiak, P. (1991). Learning invariance from transformation sequences. *Neural Computation*, 3:194–200.

Freund, Y. and Haussler, D. (1994). Unsupervised learning of distributions on binary vectors using two layer networks. Technical report, University of Calif. Santa Cruz.

Gehler, P. V., Holub, A. D., and Welling, M. (2006). The rate adapting Poisson model for information retrieval and object recognition. In *ICML*.

Griffin, G., Holub, A., and Perona, P. (2006). The caltech 256. Technical report, Caltech.

Hadsell, R., Chopra, S., and LeCun, Y. (2006). Dimensionality reduction by learning an invariant mapping. In *CVPR 2006*. IEEE Press.

Hadsell, R., Erkan, A., Sermanet, P., Scoffier, M., Muller, U., and LeCun, Y. (2008). Deep belief net learning in a long-range vision system for autonomous off-road driving. In *Intelligent Robots and Systems*.

Herault, J. and Jutten, C. (1986). Space or time adaptive signal processing by neural network models. In Denker, J., editor, *Neural Networks for Computing: AIP 151*.

Hinton, G. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800.

Hinton, G., Dayan, P., and Revow, M. (1997). Modeling the manifolds of images of handwritten digits. *IEEE Transactions on Neural Networks*, 8:65–74.

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

Hinton, G. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

Hinton, G. and Zemel, R. (1994). Autoencoders, minimum description length, and helmholtz free energy. In *NIPS*.

Hofmann, T. (1999). Probabilisitc latent semantic analysis. In *Proc. of Uncertainty in Artificial Intelligence*.

Hyvarinen, A. (2005). Estimation of non-normalized statistical models using score matching. *Journal of Machine Learning Research*, 6:695–709.

Hyvarinen, A. (2007). Connections between score matching, contrastive divergence, and pseudolikelihood, for continuous-valued variables. In *IEEE Trans. Neural Networks*.

Hyvarinen, A. and Hoyer, P. (2000). Emergence of phase- and shift-invariant features by decomposition of natural images into independent feature subspaces. *Neural Computation*, 12(7):1705–1720.

Hyvarinen, A. and Hoyer, P. (2001). A two-layer sparse coding model learns simple and complex cell receptive fields and topography from natural images. *Vision Research*, 41(18):2413–2423.

Hyvarinen, A., Karhunen, J., and Oja, E. (2001). *Independent Component Analysis*. John Wiley & Sons.

Hyvarinen, A. and Koster, U. (2007). Complex cell pooling and the statistics of natural images. *Network*, 18(2):81–100.

Jaynes, E. (1957). Information theory and statistical mechanics. *Physical review*, 106:620–630.

Jutten, C. and Herault, J. (1991). Blind separation of sources, part i: An adaptive algorithm based on neuromimetic architecture. *Signal Processing*, 24:1–10.

Kavukcuoglu, K., Ranzato, M., Fergus, R., and LeCun, Y. (2009). Learning invariant features through topographic filter maps. In *Proc. of Computer Vision and Pattern Recognition Conference*.

Kohonen, T. (1996). Emergence of invariant-feature detectors in the adaptive-subspace self-organizing map. *Biol. Cybern.*, 75:281–291.

Larochelle, H., Erhan, D., Courville, A., Bergstra, J., and Bengio, Y. (2007). An empirical evaluation of deep architectures on problems with many factors of variation. In *International Conference on Machine Learning*, pages 473–480.

Lazebnik, S., Schmid, C., and Ponce, J. (2004). Semi-local affine parts for object recognition. In *BMVC*.

Lazebnik, S., Schmid, C., and Ponce, J. (2006a). Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Proc. of Computer Vision and Pattern Recognition*, pages 2169–2178. IEEE.

Lazebnik, S., Schmid, C., and Ponce, J. (2006b). Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*.

LeCun, Y. (1987). *Modeles connexionnistes de l'apprentissage (connectionist learning models)*. PhD thesis, Université P. et M. Curie (Paris 6).

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

LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M., and Huang, F. (2006). A tutorial on energy-based learning. In Bakir, G. and al.., editors, *Predicting Structured Data*. MIT Press.

LeCun, Y., Huang, F.-J., and Bottou, L. (2004). Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of CVPR'04*. IEEE Press.

Lee, D. D. and Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791.

Lee, H., Battle, A., Raina, R., and Ng, A. (2006). Efficient sparse coding algorithms. In *NIPS*.

Lee, H., Chaitanya, E., and Ng, A. Y. (2007). Sparse deep belief net model for visual area v2. In *Advances in Neural Information Processing Systems*.

Lowe, D. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*.

Lyu, S. and Simoncelli, E. (2008). Reducing statistical dependencies in natural signals using radial gaussianization. In *NIPS*. MIT Press.

MacKay, D. (1999). Maximum likelihood and covariant algorithms for independent component analysis.

Mnih, A. and Hinton, G. (2007). Three new graphical models for statistical language modelling. In *International Conference on Machine Learning*, pages 641–648.

Mnih, A. and Hinton, G. (2008). A scalable hierarchical distributed language model. In *NIPS*.

Mutch, J. and Lowe, D. (2006). Multiclass object recognition with sparse, localized features. In *CVPR*.

Neal, R. (1993). Probabilistic inference using markov chain monte carlo methods. Technical Report CRG-TR-93-1, Dept. of Comp. Science, Univ. of Toronto.

Ning, F., Delhomme, D., LeCun, Y., Piano, F., Bottou, L., and Barbano, P. (2005). Toward automatic phenotyping of developing embryos from videos. *IEEE Transactions on Image Processing*, 14(9):1360–1371. Special issue on Molecular and Cellular Bioimaging.

Olshausen, B. A. and Field, D. J. (1997). Sparse coding with an overcomplete basis set: a strategy employed by v1? *Vision Research*, 37:3311–3325.

Osindero, S., Welling, M., and Hinton, G. (2006). Topographic product models applied to natural scene statistics. *Neural Computation*, 18(2):381–414.

Pinto, N., Cox, D. D., and DiCarlo, J. J. (2008). Why is real-world visual object recognition hard? *PLoS Computational Biology*, 4(1).

Portilla, J., Strela, V., Wainwright, M., and Simoncelli, E. (2003). Image denoising using scale mixtures of gaussians in the wavelet domain. *IEEE Trans. Image Processing*, 12(11):1338–1351.

Ranzato, M., Boureau, Y., Chopra, S., and LeCun, Y. (2007a). A unified energy-based framework for unsupervised learning. In *Proc. Conference on AI and Statistics (AI-Stats)*.

Ranzato, M., Boureau, Y., and LeCun, Y. (2007b). Sparse feature learning for deep belief networks. In *NIPS*.

Ranzato, M., Huang, F., Boureau, Y., and LeCun, Y. (2007c). Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Proc. Computer Vision and Pattern Recognition Conference*. IEEE Press.

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

Ranzato, M. and Szummer, M. (2008). Semi-supervised learning of compact document representations with deep networks. In *Internal Conference of Machine Learning*.

Rasmussen, C. and Williams, C. (2006). *Gaussian processes for machine learning*. MIT Press.

Robertson, S. and Walker, S. (1994). Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval. In *sigir*, pages 232–241.

Roth, S. and Black, M. (2005). Fields of experts: A framework for learning image priors. In *CVPR*.

Rozell, C., Johnson, D., R.G., B., and Olshausen, B. (2008). Sparse coding via thresholding and local competition in neural circuits. *Neural Computation*.

Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536.

Salakhutdinov, R. (2008). Learning and evaluating boltzmann machines. Technical report, University of Toronto.

Salakhutdinov, R. and Hinton, G. (2007a). Semantic hashing. In *Workshop on Information Retrieval and applications of Graphical Models*.

Salakhutdinov, R. and Hinton, G. (2007b). Using deep belief nets to learn covariance kernels for gaussian processes. In *NIPS 20*. MIT Press.

Salakhutdinov, R., Mnih, A., and Hinton, G. (2007). Restricted boltzmann machines for collaborative filtering. In *International Conference on Machine Learning*, pages 791–798.

Schmid, C. and Mohr, R. (1997). Local grayvalue invariants for image retrieval. *IEEE Trans. Pattern Analalysis and Machine Intelligence*, 19(5):530–535.

Scholkopf, B., Platt, J., Shawe-Taylor, J., Smola, A., and Williamson, R. (2001). Estimating the support of a high-dimensional distribution. *Neural Computation*, 13:1443–1471.

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

Simard, P., Steinkraus, D., and Platt, J. (2003). Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, pages 958–962.

Simoncelli, E. (1997). Statistical models for images: Compression, restoration and synthesis. In *31st Asilomar Conf on Signals, Systems and Computers*, volume 1, pages 673–678.

Simoncelli, E. P., Freeman, W. T., Adelson, E. H., and Heeger, D. J. (1998). Shiftable multi-scale transforms. *IEEE Trans. Information Theory*, 38(2):587–607.

Teh, Y. W., Welling, M., Osindero, S., and Hinton, G. E. (2003). Energy-based models for sparse overcomplete representations. *Journal of Machine Learning Research*, 4:1235–1260.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, 8:257–277.

Tishby, N., Pereira, F., and Bialek, W. (1995). The information bottleneck method. *Neural Computation*, 7:1129–1159.

Torralba, A., Fergus, R., and Weiss, Y. (2008). Small codes and large databases for recognition. In *Computer Vision and Pattern Recognition*.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P. (2008). Extracting and composing robust features with denoising autoencoders. In *ICML*.

Wegmann, B. and Zetzsche, C. (1990). Statistical dependence between orientation filter outputs used in an human vision based image code. In *Visual Comm. and Image Processing*, volume 1360, pages 909–922.

Weiss, Y. and Freeman, W. (2007). What makes a good model of natural images? In *CVPR*.

Weston, J., Ratle, F., and Collobert, R. (2008). Deep learning via semi-supervised embedding. In *International Conference on Machine Learning*.

Wiskott, L. and Sejnowski, T. (2002). Slow feature analysis: unsupervised learning of invariances. *Neural Computation*, 14(4):715–770.

Yuan, M. and Lin, Y. (2004). Model selection and estimation in regression with grouped variables. Technical report, Univ. of Winsconsin.

Zhang, H., Berg, A. C., Maire, M., and Malik, J. (2006). Svm-knn: Discriminative nearest neighbor classification for visual category recognition. In *CVPR*.

Zhu, S., Wu, Y., and Mumford, D. (1997). Minimax entropy principle and its application to texture modeling. *Neural Computation*, 9(8):1627–1660.