A Unified Energy-Based Framework for Unsupervised Learning

Marc'Aurelio Ranzato **Y-Lan Boureau** **Sumit Chopra** **Yann LeCun**
Courant Insitute of Mathematical Sciences
New York University, New York, NY 10003

Abstract

We introduce a view of unsupervised learning that integrates probabilistic and non-probabilistic methods for clustering, dimensionality reduction, and feature extraction in a unified framework. In this framework, an energy function associates low energies to input points that are similar to training samples, and high energies to unobserved points. Learning consists in minimizing the energies of training samples while ensuring that the energies of unobserved ones are higher. Some traditional methods construct the architecture so that only a small number of points can have low energy, while other methods explicitly "pull up" on the energies of unobserved points. In probabilistic methods the energy of unobserved points is pulled by minimizing the log partition function, an expensive, and sometimes intractable process. We explore different and more efficient methods using an energy-based approach. In particular, we show that a simple solution is to restrict the amount of information contained in codes that represent the data. We demonstrate such a method by training it on natural image patches and by applying to image denoising.

1 Introduction

The main goal of unsupervised learning is to capture regularities in data for the purpose of extracting useful representations or for restoring corrupted data. For example, probabilistic density models can be used for deriving efficient codes (for compression), or for restoring corrupted data (e.g. denoising images). Many unsupervised methods explicitly produce representations or *codes*, or feature vectors, from which the data is to be reconstructed. For example in clustering methods such as K-Means, the code is the index of the prototype in the codebook that is closest to the data vector. Similarly in Principal Component Analysis, the code is the projection of the data vector on a linear subspace. In auto-encoder neural nets, the code is the state of a hidden layer with a small number of units, from which a vector on a low-dimensional non-linear manifold is reconstructed. In Restricted Boltzmann Machines [4], the code is a vector of stochastic binary variables from which the input can be reconstructed. Finally, in sparse-overcomplete representations methods such as Non-negative Matrix Factorization [2], Olshausen and Field's sparse overcomplete coding method [7], or the Energy-Based Model for learning sparse over-complete features [11], the code is a high-dimensional vector with most of the components at zero. In general, these models are trained in such a way that training data can be accurately reconstructed from the codes, while unobserved data vectors cannot. By contrast, most density estimation methods (e.g. Gaussian Mixture Models) or the Product of Experts methods [3] do not produce explicit codes from which the data can be reconstructed, but merely compute a function (the negative log probability) that produces low values around training data, and high values everywhere else.

Unsupervised methods appear very diverse, and based on very different principles. One of the main objectives of this paper is to show that most unsupervised learning methods are based on a common underlying principle. At a high level, many unsupervised models can be viewed as a scalar-valued energy function $E(Y)$ that operates on input data vectors Y. The function $E(Y)$ is designed to produce low energy values when Y is similar to some training data vectors, and high energy values when Y is dissimilar to any training data vector. The energy function is subject to learning. Training an unsupervised model consists in searching for an energy function within a family $\{E(Y,W),\ \ W \in \mathcal{W}\}$ indexed by a parameter W, that gives low energy values on input points that are similar to the training samples, and large values on dissimilar points.

We argue that the various unsupervised methods merely differ on three points: how $E(Y, W)$ is parameterized, how the energy of observed points is made small, and how the energy of unobserved points is made large. Common probabilistic and non-probabilistic methods use particular combinations of model architecture and loss functions to achieve this. This paper discusses which combinations of architectures and loss functions 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 the energy of unobserved points that turns out to be very inefficient in many cases. We propose new loss functions 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 the energy of unobserved points by limiting the information content of the code. This principle is illustrated by a new model that can learn low-entropy, sparse and overcomplete codes. The model is trained on natural image patches and applied to image denoising at very high noise levels with state-of-the-art performance.

1.1 Probabilistic Density Models

A probabilistic density model defines a normalized density $P(Y)$ over the input space. Most probability densities can be written as deriving from an energy function through the Gibbs distribution:

$$P(Y, W) = \frac{e^{-\beta E(Y,W)}}{\int_y e^{-\beta E(y,W)}} \quad (1)$$

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 $E(Y, W) = -\log P(Y, W)$. Training a probabilistic density model from a training dataset $T = \{Y^i, \quad i \in 1 \ldots p\}$ is generally performed by finding the W that maximizes the likelihood of the training data under the model $\prod_{i=1}^{p} P(Y^i, W)$. Equivalently, we can minimize a loss function $L(W, T)$ that is proportional to the negative log probability of the data. Using the Gibbs expression for $P(Y, W)$, we obtain:

$$L(W, T) = \frac{1}{p}\sum_{i=1}^{p} E(Y^i, W) + \frac{1}{\beta}\log\int_y e^{-\beta E(y,W)} \quad (2)$$

The gradient of $L(W, T)$ with respect to W is:

$$\frac{\partial L(W,T)}{\partial W} = \frac{1}{p}\sum_{i=1}^{p}\frac{\partial E(Y^i,W)}{\partial W} - \int_y P(y,W)\frac{\partial E(y,W)}{\partial W} \quad (3)$$

In other words, minimizing the first term in eq. 2 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 equation 3) may be intractable when Y is a high dimensional variable and $E(Y, W)$ 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 [4]. The basic idea of contrastive divergence is to avoid pulling up the energy of *every possible point* Y, and to merely pull up on the energy of randomly generated points located near 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.

1.2 Energy-Based Models

While probabilistic approaches rely on the normalization of the density to guarantee that the energies of unobserved points are larger than that of the training points, other methods use different strategies. Energy-based approaches produce suitable energy surfaces by minimizing other loss functionals than the negative log likelihood (see [1] for a general introduction to energy-based learning in the supervised case). Given a training set $T = \{Y^i, \quad i \in 1 \ldots p\}$, we must define a parameterized family of energy functions $E(Y, W)$ in the form of an *architecture*, and a *loss functional* $L(E(\cdot, W), T)$ whose role is to measure the "quality" (or badness) of the energy surface $E(\cdot, W)$ on the training set T. An energy surface is "good" if it gives low energy to areas around the training samples, and high energies to all other areas. The simplest loss functional that one can devise, called the *energy loss*, is simply the average energy of the training samples

$$L_{\text{energy}}(W, T) = \frac{1}{p}\sum_{i=1}^{p} E(Y^i, W) \quad (4)$$

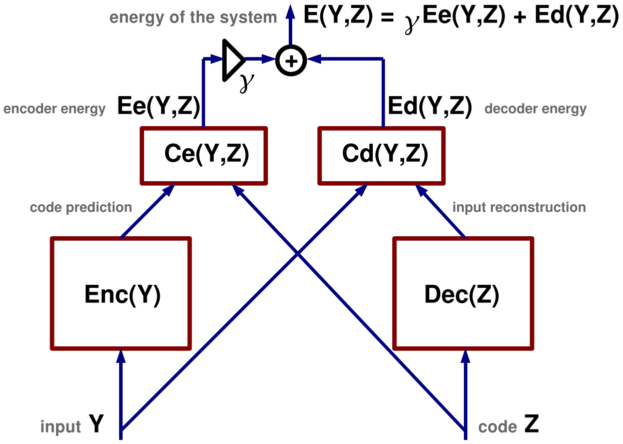

Figure 1: A general architecture for unsupervised learning.

In general, minimizing this loss *does not* produce good energy surfaces because, unless $E(Y, W)$ has a special form, nothing prevents the energy surface from becoming flat. No term increases the loss if the energy of unobserved points is low, hence minimizing this loss will not ensure that the energies of unobserved points are higher than the energies of training points. 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 negative log probability loss used in probabilistic models is merely a particular way of doing so. 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 low energies. The region of low energy can be designed to be a manifold with a given dimension, or a discrete set of regions around which the energy is low. 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 high energies.

We will now introduce the general form of architecture for unsupervised learning. We will then discuss how several classical unsupervised learning models can be interpreted in terms of the energy-based framework.

1.3 Common Architectures

For applications such as data restoration (e.g. image denoising), one merely requires a good energy surface over the input space. Since a properly trained model assigns low energies to areas of the input space that correspond to clean data, restoring a corrupted input vector may be performed by finding an area of low energy near that input vector [3, 10, 9]. If the energy surface is smooth, this can be performed using gradient descent. However, many applications require that the model extract *codes* (or representations) from which the training samples can be reconstructed. The code is often designed to have certain desirable properties, such as compactness, independence of the components, or sparseness. Such applications include dimensionality reduction, feature extraction, and clustering. The code vector, denoted by Z can be seen as a deterministic latent variable in the energy function. Most common models of unsupervised learning can be seen as using an energy function of the following form

$$E(Y) = \min_{Z \in \mathcal{Z}} E(Y, Z) \tag{5}$$

In other word, the optimal code Z_Y^* for an input vector Y is obtained by minimizing the energy function with respect to Z:

$$Z_Y^* = \text{argmin}_{Z \in \mathcal{Z}} E(Y, Z) \tag{6}$$

The minimum is searched within a set $\mathcal{Z}$. In probabilistic terms, this corresponds to finding the maximum likelihood estimate for the latent variable Z.

The vast majority of unsupervised learning methods that extract a code are a special case of the architecture depicted in figure 1. The system is composed of an *encoder* and a *decoder*. The encoder takes the input Y and computes a prediction of the best value of the latent code Z. The decoder computes a reconstruction of Y from the latent code vector Z. The energy of the system is the sum of two terms: the decoding energy $E_d(Y, Z)$ which measures the discrepancy between the input vector and its reconstruction, and the encoding energy $E_e(Y, Z)$ which measures the discrepancy between the code and its prediction by the encoder:

$$E(Y, Z) = \gamma E_e(Y, Z) + E_d(Y, Z) \tag{7}$$

The positive coefficient γ sets the tradeoff between minimizing the reconstruction energy and minimizing the code prediction error when searching for the optimal code $Z_Y^* = \text{argmin}_Z E(Y, Z)$.

Specializations of this architecture include cases where either the encoder or the decoder is missing, as well as cases in which the encoding energy is constrained to be zero: the optimal code is constrained to be equal to the value predicted by the encoder (this can be interpreted as equivalent to setting γ to a very large value). Section 2 re-interprets several classical unsupervised methods in this framework.

1.4 Avoiding Flat Energy Surfaces by Limiting the Information Content of the Code

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. 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.

One solution to this problem is to make the energy surface a little "stiff", so that pulling on a small number of well-chosen points will automatically pull up the energies of many points [1]. This can be achieved by limiting the number of parameters in the energy parameterization, or by favoring smooth energy surfaces through a regularization term in the loss.

Another solution is to design the energy-surface 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 $E(Y) = \min_{i\in[1,N]} ||Y - W_i||^2$, can only produce low energy values (good reconstructions) around one of the N prototypes W_i, and high 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 reconstructed from the code with a small reconstruction error (low energy). The architecture should also be designed so that unobserved points are assigned codes similar to those associated with training samples, so that their reconstruction is close to a training sample. In other words: only training samples (and similar points) can possess codes from which they can be correctly reconstructed. Satisfying this property can be done in a number of ways, but the simplest way is to limit the information content of the code. This can be done by making the code a discrete variable with a small number of different values (as with the example of the previous paragraph), or by making the code have a lower dimension than the input, or by forcing the code to be a "sparse" vector in which most components are zero. Classical unsupervised learning methods use these methods implicitly as described in the next section. From the probabilistic point of view, bounding the log partition function also bounds the entropy of the code.

2 Classical Methods in the Light of the Energy-Based Framework

In this section, we review a number of classical unsupervised learning models in the light of the energy-based framework. Most of them use special cases of the encoder/decoder architecture described above. They differ in the specifics of the architecture, the constraints on the code, and the loss function used for training. To illustrate how each method works, we will use a toy problem, designed to facilitate the visualization of the energy surface.

2.1 A Toy Problem: the spiral

We consider a training dataset consisting of 10,000 points in the 2D plane (y_1, y_2). The training 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 with low energies along the spiral and higher energies everywhere else. The spiral 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: the real problems in unsupervised learning occur in high dimensional tasks. Nevertheless, this toy example is a useful didactic tool.

2.2 Principal Component Analysis

PCA is an encoder-decoder architecture that minimizes the energy loss (mean square reconstruction error), without requiring an explicit contrastive term to pull up the energies of unobserved patterns. In PCA the optimal code is constrained to be equal to the value predicted by the encoder. PCA avoids flat energy surface by using a code with a lower dimension than the input. With reference to the general architecture in figure 1, we have $Enc(Y) = WY$, $Dec(Z) = W^T Z$, with $W \in R^{N\times M}$ where $N \leq M$ and the rows of the matrix W are orthonormal. The cost modules are $C_e(Y, Z) = \|WY - Z\|^2$, $C_d(Y, Z) = \|W^T Z - Y\|^2$, with a very large value for γ so that $Z^* = WY$. Hence the energy can be reduced to $E(Y, W) = \|W^T WY - Y|^2$. Only those Y vectors

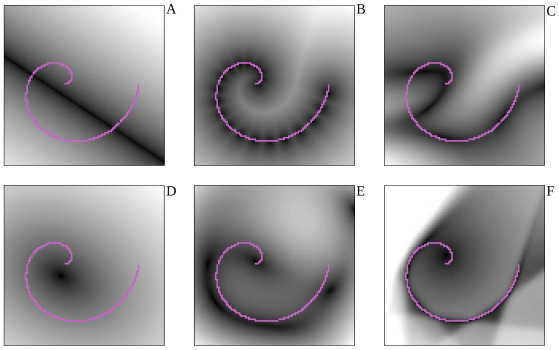

Figure 2: Energy surface for: A) PCA with one code unit, B) K-Means with 20 prototypes, C) autoencoder with 1 code unit and trained by energy loss minimization, D) autoencoder with 20 code units and trained by minimizing the negative log probability of the data. E) 2-20-2 autoencoder trained with the linear-linear contrastive loss. F) energy-based model for learning sparse overcomplete representations with 20 code units (average entropy 0.14 bits per code unit). The brigther the area, the higher is the energy (i.e. the reconstruction error).

that are in the space spanned by the lines of W will be exactly reconstructed (with zero energy). Hence only a linear subspace of the input can have low energy. Therefore, learning can be carried out by simply minimizing the average energy of the training samples, without having to worry about pulling up the energies of unobserved points: their energies will automatically become high (except if they happen to be on the linear subspace).

PCA performs very poorly on the toy spiral example. With one code unit, the region of low energy is a horizontal straight line covering the main dimension of the spiral. With two code units, the entire input space has low energy, see figure 2-A. Every point in the plane gets reconstructed perfectly (the system simply computes the identity function), but the model is essentially useless as it does not discriminate between training point areas and unobserved point areas.

2.3 K-Means Clustering

The architecture for K-means clustering has no encoder, only a decoder and a reconstruction cost module. The code variable Z is an integer variable between 1 and N, where N is the number of prototypes. The energy is simply $E(Y,Z) = ||Y - W_Z||^2$, where W_Z is the Z-th prototype. The inference process finds the value of Z that minimizes the energy:

$$E(Y) = \min_{Z \in [1,N]} E(Y,Z) = \min_{Z \in [1,N]} ||Y - W_Z||^2. \quad (8)$$

An alternative view of K-means is to consider Z as a binary vector with N components in which only one component is 1, and the others 0 (one-of-N sparse binary code), and the energy is simply: $E(Y,Z) = \sum_i Z_i ||Y - W_i||^2$. The entropy of this code as we sample from the training distribution (or from any high-entropy distribution over the input domain) is at most $log_2 N$ bits. The only points that are reconstructed with zero energy are the prototypes. Every other point has higher energy.

K-Means learning can be seen as a simple iterative minimization of the energy loss (the average energy of the training samples). Figure 2-B shows the energy surface obtained by training K-means with 20 prototypes on the toy spiral dataset. The energy is suitably low on the spiral and high everywhere else, but somewhat "bumpy". The limitation of K-means is that it has trouble giving low energies to vast areas of the space in high dimensions, a consequence of the local nature of its energy minima.

2.4 Narrow 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 in the entropy of the code allows us to simply pull down on the energy of the training samples, without having to pull up on unobserved points. The energy is simply $E(Y) = |\text{Dec}(\text{Enc}(Y)) - Y|^2$, in which both Enc and Dec are neural nets, possibly with multiple layers. Again, because of the restricted dimension of the code, minimizing the energy loss is sufficient. In the toy experiment we have used a multi-layer autoencoder. The encoder has a first hidden layer with 100 units, and a second hidden layer with only 1 code unit. The decoder is symmetric to the encoder. The energy surface is shown in figure 2-C. The result is slightly better than PCA's result with one code unit. As with PCA, using two code units simply results in a flat energy surface. This example only confirms what several authors have reported: gradient-descent learning in deep auto-encoders has a very hard time finding appropriate low-dimensional representations [5].

3 Loss Functions for Wide Autoencoders

PCA, K-Means, and narrow autoencoders do not require contrastive terms in the loss to pull up the energies of unobserved points. We now consider architectures that can, in principle, produce flat energy surfaces, and explore various strategies to prevent them from doing so. An example of such model is the encoder-decoder architecture in which the code may have a larger dimension than the input. The simplest such architecture is the so-called *wide autoencoder*. It is a multilayer neural net with input and output layers of identical sizes, but with a larger hidden layer. In the following experiments, we use a 2-20-2 autoencoder with sigmoid units and apply it to the spiral toy problem. The energy is defined as the square euclidean distance between the input and the output (the reconstruction). Training a wide autoencoder with the energy loss (minimizing the average energy of the training samples), leads to a catastrophic collapse and simply results in a flat or quasi-flat energy surface.

3.1 Negative Log Probability Loss

The most natural choice for the energy of the system is the square euclidean distance between input and output of the network (i.e. the square reconstruction error). To prevent the energy surface from collapsing, we can use a probabilistic approach and minimize the negative log likelihood loss. This will pull up on the energy of unobserved points. Naturally, there is no analytic expression for the log partition function, since it involves a non-linear neural net function in the exponential. Hence we must resort to approximate methods to evaluate the derivative of the log partition function as shown in equation 3. Since the toy problem is two-dimensional, we can get away with replacing the integral by a discrete sum at regularly-spaced gridpoints in a square with opposite corners at (-1,1) and (1,-1). We use a grid size of 0.2×0.2, resulting in 101×101 samples. The resulting energy surface is shown in figure 2-D. Since the number of parameters of the system is not very large (the energy surface is a bit "stiff"), the energy surface is not perfect. Hence, the reconstruction is poor compared to the other methods. More importantly, the learning is quite slow and inefficient because the procedure pulls on every single point on the grid, including the ones whose energy is already high as a result of pulling up other points. While this problem is merely an annoyance in this low-dimensional toy problem, it becomes an insurmountable obstacle in high dimensional problems.

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 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 square reconstruction error, an important consequence of this fact is that *minimizing the negative log probability loss does not lead to good reconstruction of training samples*. This is a crucially important deficiency of the negative log probability loss (and probabilistic methods that use it) for training unsupervised methods that reconstruct inputs from codes.

3.2 Contrastive Divergence: RBM

In a Restricted Boltzmann Machine [4], Z is binary. When Y is binary, we have: $E_e(Y,Z) = -\frac{1}{2}Z^TW^TY$, $E_d(Y,Z) = -\frac{1}{2}Y^TWZ = E_e(Y,Z)$, $\gamma = 1$. The number of hidden units is unconstrained. The code is not picked by minimizing energy, but sampled according to the distribution defined by the energy. Once Z has been picked, $Dec(Z)$ is chosen by sampling as well. When Y is continuous, $Enc(Y)$ is still chosen by sampling, but $Dec(Z) = \sigma(WZ)$, where $\sigma(.)$ is logistic, which corresponds to taking the average over the distribution of binary vectors Y corresponding to the energy $-Z^TW^TY$. Weights are updated according to the contrastive divergence rule [4], which approximates the gradient of the log-likelihood of the data, and leads to the same problems as the negative log probability loss on this dataset.

3.3 Linear-Linear Contrastive Loss

In this experiment also, the energy is defined as the square euclidean distance between input and output of the network. In order to reduce the cost associated with minimizing the negative log probability loss, we propose an idea that has a similar flavor to contrastive divergence. 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 start from the training sample and run a few steps of a Langevin dynamics on the energy surface. The Langevin dynamics updates the current estimate of $\bar{Y}$ in the following way:

$$\bar{Y} \leftarrow \bar{Y} - \eta \frac{\partial E(Y)}{\partial Y}(\bar{Y}) + \epsilon \qquad (9)$$

where ϵ is a random sample from a zero-mean multivariate Gaussian distribution with a predefined isotropic variance. 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 $\bar{Y}$ samples up to a given energy level m, called the *margin*:

$$L(Y,W) = \alpha E(Y,W) + \max(0, m - E(\bar{Y},W)) \qquad (10)$$

For the experiment, we set $m = 0.5$, and $\alpha = 2$. Setting the constant α to a value larger than 1 ensures that energies of training samples are pushed down a bit harder than the energies of unobserved points are pulled up. This is appropriate for energies that are bounded below by zero, as is the case here. The resulting energy surface is shown in figure 2-E. 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.

3.4 Sparse codes

We already pointed out that a good way to prevent flat energy surfaces is to limit the information content of the code. The Restricted Boltzmann Machine can be seen as using this method by making the code "noisy", hence limiting its information content, a technique also used implicitly in [6]. This idea can be applied to the wide auto-encoder in another way. The idea is to constrain the code to be sparse by forcing each variable in the code to be zero most of the time. This idea was used by [7] and [2] in a linear architecture that did not include an encoder (just a decoder). A simple way to apply this to the wide autoencoder is to use logistic non-linearities in the hidden layer with a very high threshold [11]. The threshold is dynamically adjusted so that the unit turns on only a small portion of the time (or rather, for a small portion of the training samples). This creates particular difficulties for back-propagation learning because the gradient all but vanishes when back-propagated through those high-threshold logistics. The solution to this problem is to use a latent variable for the code with an encoding energy. The method is described in detail in the next section. In this section, we merely describe the results on the toy dataset. Using an architecture with 20 code units, the energy landscape shown in figure 2-F presents a nice groove along the training samples. Each code unit has a measured empirical entropy of 0.14 bits.

4 Learning Sparse Features with the Encoder/Decoder Architecture

In this section we describe the energy-based model for learning sparse and overcomplete representations recently introduced in [11], and give new results on image denoising. The architecture of the system is identical to the one in figure 1. In the experiments described here, the encoder and the decoder are a set of linear filters, contained in the rows of matrix W_e and the columns of matrix W_d respectively. Between them, there is a non-linearity dubbed *Sparsifying Logistic*. Conceptually, this non-linearity is part of the decoder and transforms the code vector Z into a sparse code vector $\bar{Z}$ with components in the interval $[0, 1]$. Let us consider the i-th component of the code for the k-th training sample $z_i(k)$. After the sparsifying logistic, the sparse code component is:

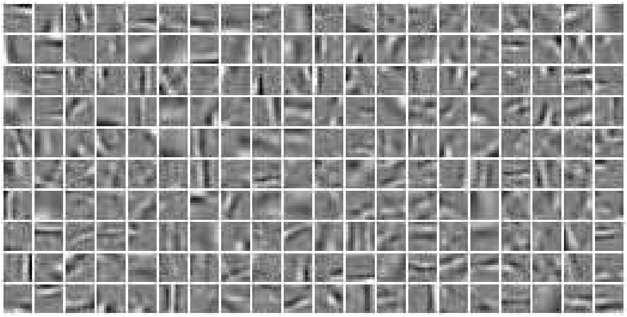

Figure 3: Results of feature extraction from 12x12 patches taken from the Berkeley dataset, showing the 200 filters learned by the decoder.

$$\bar{z}_i(k) = \frac{\eta e^{\beta z_i(k)}}{\zeta_i(k)},\ i \in [1..m] \quad \text{with}$$
$$\zeta_i(k) = \eta e^{\beta z_i(k)} + (1-\eta)\zeta_i(k-1) \qquad (11)$$

where $\eta \in [0, 1]$ and $\beta > 0$ are predefined parameters, and $\zeta_i(k)$ is a moving average of $e^{\beta z_i(n)}$ over past training samples. This non-linearity can be easily understood as a weighted softmax function applied to consecutive samples of the same code unit. This non-linearity outputs a sequence of positive values in the range $[0, 1]$ which, for large values of β and small values of η, is characterized by brief and punctuate activities across samples. Parameters η and β allow us to control the representational capacity of the code. In particular, large values of β and small values of η yield a quasi-binary and sparse representations. The *energy* of the system is the sum of the prediction energy (or encoding energy) and reconstruction energy (or decoding energy):

$$E(Y, Z, W_e, W_d) = E_e(Y, Z, W_e) + E_d(Y, Z, W_d) \quad (12)$$

with $E_e(Y, Z, W_e) = \frac{1}{2}||Z - W_e Y||^2$ and $E_d(Y, Z, W_d) = \frac{1}{2}||Y - W_d\bar{Z}||^2$, where $\bar{Z}$ is computed by applying the sparsifying logistic non-linearity to Z.

Learning is accomplished by minimizing the average energy in eq. (12) over the training set. The on-line learning algorithm for one training sample can be summarized as follows: (1) propagate the input Y through the encoder to get a codeword prediction Z_{init}; (2) minimize the loss in eq. (12), sum of reconstruction and code prediction energy, with respect to Z by gradient descent using Z_{init} as the initial value; (3) compute the gradient of the loss with respect to W_e and W_d, and perform a gradient step. Since the energy is the sum of encoder and decoder energies, the system converges to a state where minimum-energy code

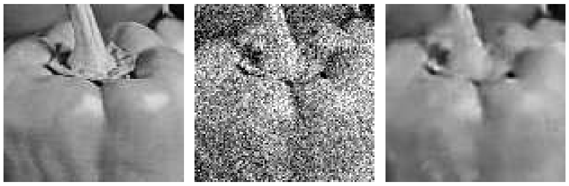

Figure 4: Example of denoising (details of the "peppers" image). Left panel: original image. Central panel: noisy image ($\sigma = 50$); PSNR equal to 14.15dB. Right panel: reconstructed image using a machine trained to reconstruct clean patches from noisy ones (with $\sigma = 50$); PSNR equal to 26.35dB.

vectors not only reconstruct the image patch but can also be easily predicted by the encoder filters. This algorithm is very efficient because the sparsity constraints enforced on the code free us from the pull-up phase which is necessary to avoid trivial solutions, in general. Since we limit the code representational capacity and, at the same time, force it to reconstruct the training samples, low energy cannot possibly be assigned to other points in input space. Hence, there is no need to spend resources pulling up energies of unobserved points. Moreover, during testing, computing the code of the input is achieved by a forward propagation through the encoder which was trained to produce prediction of optimal codes. Unlike other methods [7], no minimization in code space is required at this stage. The decoder provides a direct way to reconstruct code vectors and we never need to resort to computationally expensive sampling techniques [3]. Finally, the method is robust to noise in the input. Pre-processing consists in simple mean removal and scaling. Other methods [7, 3] need to whiten the input images in order to get a better convergence of their learning algorithms.

4.1 Experiments

In the first experiment, the system was trained on 100,000 gray-level patches of size 12x12 extracted from the Berkeley segmentation data set [8]. Pre-processing of images consists of subtracting the local mean, and dividing the result by 100. The sparsifying logistic parameters η and β were equal to 0.02 and 1, respectively. Learning proceeds as described in the previous section with the addition of a *lasso* regularizer to the energy loss function minimized during training in order to encourage spatial localization and suppress noise: $L(W_e, W_d, T) = E_e(Y, Z, W_e) + E_d(Y, Z, W_d) + \lambda \|W_e\|_1 + \lambda \|W_d\|_1$, with λ equal to 0.001. Finally, during training the running averages ζ_i of eq. (11) are saturated to allow the units corresponding to the low pass filters to be more often active. Training by one pass over the training samples takes less than half an hour on a 2GHz processor. After few thousand samples the parameters are very close to their final value; the decoding filters that were learned are shown in figure 3. They are spatially localized, and have different orientations, frequencies and scales. They are somewhat similar to, but more localized than, Gabor wavelets and are reminiscent of the receptive fields of V1 neurons. Interestingly, the encoder and decoder filter values are nearly identical up to a scale factor.

This method can be easily modified by using hierarchical architectures in the encoder and decoder, and it can find many different applications [11]. For instance, it can be applied to *denoise* images. Suppose we know the statistics of the noise, e.g. additive Gaussian, Poisson, etc. Then, we can build a dataset of patches corrupted by the same kind of noise and train the machine by feeding the encoder with these noisy patches and assigning as target value for the decoder the corresponding original clean image patches. The system will learn the mapping from noisy to clean image patches, and it can be used to denoise images. In order to reconstruct an image, we consider all possible overlapping patches that can be taken from it and we average their reconstructions. Experiments were performed using additive Gaussian noise with a standard deviation equal to 50, 75 and 100, training on the Berkeley dataset, and applying the system to standard test images. The system has been trained on 8x8 patches by using 256 filters in both encoder and decoder. An example of denoising is shown in figure 4. Table 1 summarizes the results in terms of PSNR by comparing with three state-of-the-art denoising methods. The performance achieved by this simple method is comparable to these other methods that were explicitly designed for denoising. Note that in order to compute each image patch reconstruction we need to compute two matrix vector multiplication and a rectification since no minimization in code space is necessary after training. In order to denoise a 256x256 image about 2 billion operations are required with this choice of number and size of filters.

5 Conclusion

This paper focuses on six main ideas. First, many unsupervised learning methods can be described in the framework of energy-based models as carving an energy surface so as to minimize the energies of training points, and maximizing the energies of unobserved points. Second, the inference process in several unsupervised learning methods can be seen as minimizing an energy function of the form $E(Y) = \min_{Z \in \mathcal{Z}} E(Y, Z)$, where Z is a latent code. Third, many unsupervised methods use an architecture for

Image	std. 50/PSNR 14.15				std. 75/PSNR 10.63				std. 100/PSNR 8.13			
Lena	27.86	**28.61**	27.79	26.49	25.97	**26.84**	25.80	24.13	24.49	**25.64**	24.46	21.87
Barb.	23.46	**25.48**	25.47	23.15	22.46	**23.65**	23.01	21.36	21.77	**22.61**	21.89	19.77
Boat	26.02	**26.38**	25.95	24.53	24.31	**24.79**	23.98	22.48	23.09	**23.75**	22.81	20.80
House	27.85	**28.26**	27.95	26.74	25.77	**26.41**	25.22	24.13	24.20	**25.11**	23.71	21.66
Peppers	**26.35**	25.90	26.13	24.52	**24.56**	24.00	23.69	21.68	**23.04**	22.66	21.75	19.60

Table 1: Summary of denoising results in PSNR [dB] for different noise levels. Starting from the left column of each section: the energy-based model, Portilla (2003) [10], Elad (2006) [9] and Roth (2005) [12] (in bold font the best performance).

$E(Y, Z)$ that contain a encoder that predicts the code Z, and a decoder that reconstructs Y from the code. Fourth, probabilistic methods ensure that the energy surface is not flat by pulling up on the energies of unobserved points through the log partition function. Fifth, we proposed more efficient methods to pick the unobserved points whose energy must be pulled up, as well as other loss function that do not excessively pull up on energies. Sixth, we proposed methods that guarantee that the energies of unobserved points is high by restricting the information content in the code. Application of this idea to image denoising yields state-of-the-art performance at high noise levels.

References

[1] LeCun, Y., Chopra, S., Hadsell, R., Ranato, M., and Huang, F.-J. (2006) A Tutorial on Energy-Based Learning. in Bakir et al. (eds) "Predicting Structured Outputs", MIT Press.

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

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

[4] Hinton, G.E. (2002) Training products of experts by minimizing contrastive divergence. Neural Computation, 14:1771-1800.

[5] Hinton, G. E. and Salakhutdinov, R. R (2006) Reducing the dimensionality of data with neural networks. Science, Vol. 313. no. 5786, pp. 504 - 507, 28 July 2006.

[6] Doi E., Balcan, D.C. and Lewicki, M.S. (2006) A Theoretical Analysis of Robust Coding over Noisy Overcomplete Channels. Advances in Neural Information Processing Systems 18, MIT Press.

[7] 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.

[8] The Berkeley Segmentation Dataset http://www.cs.berkeley.edu/projects/vision/grouping/segbench/

[9] Elad, M. and Aharon, M. (2006) Image Denoising Via Learned Dictionaries and Sparse Representation. Computer Vision and Pattern Recognition, 895-900.

[10] Portilla, J., Strela, V., Wainwright, M. and Simoncelli, E.P. (2003) Image Denoising Using Scale Mixtures of Gaussians in the Wavelet Domain. IEEE Trans. Image Processing, 12(11): 1338-1351.

[11] Ranzato, M., Poultney, C.S., Chopra, S. and LeCun, Y. (2006) Efficient Learning of Sparse Overcomplete Representations with an Energy-Based Model. Advances in Neural Information Processing Systems 19, MIT Press.

[12] Roth, S. and Black, M.J. (2005) Fields of experts: a framework for learning image priors. IEEE Conference on Computer Vision and Pattern Recognition