Journal of Machine Learning Research 11 (2010) 3371-3408 Submitted 5/10; Published 12/10

Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion

Pascal Vincent PASCAL.VINCENT@UMONTREAL.CA
Département d'informatique et de recherche opérationnelle
Université de Montréal
2920, chemin de la Tour
Montréal, Québec, H3T 1J8, Canada

Hugo Larochelle LAROCHEH@CS.TORONTO.EDU
Department of Computer Science
University of Toronto
10 King's College Road
Toronto, Ontario, M5S 3G4, Canada

Isabelle Lajoie ISABELLE.LAJOIE.1@UMONTREAL.CA
Yoshua Bengio YOSHUA.BENGIO@UMONTREAL.CA
Pierre-Antoine Manzagol PIERRE-ANTOINE.MANZAGOL@UMONTREAL.CA
Département d'informatique et de recherche opérationnelle
Université de Montréal
2920, chemin de la Tour
Montréal, Québec, H3T 1J8, Canada

Editor: Léon Bottou

Abstract

We explore an original strategy for building deep networks, based on stacking layers of *denoising autoencoders* which are trained locally to denoise corrupted versions of their inputs. The resulting algorithm is a straightforward variation on the stacking of ordinary autoencoders. It is however shown on a benchmark of classification problems to yield significantly lower classification error, thus bridging the performance gap with deep belief networks (DBN), and in several cases surpassing it. Higher level representations learnt in this purely unsupervised fashion also help boost the performance of subsequent SVM classifiers. Qualitative experiments show that, contrary to ordinary autoencoders, denoising autoencoders are able to learn Gabor-like edge detectors from natural image patches and larger stroke detectors from digit images. This work clearly establishes the value of using a denoising criterion as a tractable unsupervised objective to guide the learning of useful higher level representations.

Keywords: deep learning, unsupervised feature learning, deep belief networks, autoencoders, denoising

1. Introduction

It has been a long held belief in the field of neural network research that the composition of *several levels of nonlinearity* would be key to efficiently model complex relationships between variables and to achieve better generalization performance on difficult recognition tasks (McClelland et al., 1986; Hinton, 1989; Utgoff and Stracuzzi, 2002). This viewpoint is motivated in part by knowledge

of the layered architecture of regions of the human brain such as the visual cortex, and in part by a body of theoretical arguments in its favor (Håstad, 1986; Håstad and Goldmann, 1991; Bengio and LeCun, 2007; Bengio, 2009). Yet, looking back at the history of multi-layer neural networks, their problematic non-convex optimization has for a long time prevented reaping the expected benefits (Bengio et al., 2007; Bengio, 2009) of going beyond one or two hidden layers.[1] Consequently much of machine learning research has seen progress in shallow architectures allowing for convex optimization, while the difficult problem of learning in deep networks was left dormant. 暂时搁置的

The recent revival of interest in such *deep architectures* is due to the discovery of novel approaches (Hinton et al., 2006; Hinton and Salakhutdinov, 2006; Bengio et al., 2007; Ranzato et al., 2007; Lee et al., 2008) that proved successful at learning their parameters. Several alternative techniques and refinements have been suggested since the seminal work on deep belief networks (DBN) by Hinton et al. (2006) and Hinton and Salakhutdinov (2006). All appear however to build on the same principle that we may summarize as follows:

- Training a deep network to directly optimize only the supervised objective of interest (for example the log probability of correct classification) by gradient descent, starting from random initialized parameters, does not work very well.

- What works *much* better is to initially use a *local unsupervised criterion* to (pre)train each layer in turn, with the goal of learning to produce a useful *higher-level representation* from the lower-level representation output by the previous layer. From this starting point on, gradient descent on the supervised objective leads to much better solutions in terms of generalization performance.

Deep layered networks trained in this fashion have been shown empirically to avoid getting stuck in the kind of poor solutions one typically reaches with only random initializations. See Erhan et al. (2010) for an in depth empirical study and discussion regarding possible explanations for the phenomenon.

In addition to the supervised criterion relevant to the task, what appears to be key is using an additional *unsupervised criterion* to guide the learning at each layer. In this sense, these techniques bear much in common with the semi-supervised learning approach, except that they are useful even in the scenario where all examples are labeled, exploiting the input part of the data to regularize, thus approaching better minima of generalization error (Erhan et al., 2010).

There is yet no clear understanding of what constitutes "good" representations for initializing deep architectures or what explicit unsupervised criteria may best guide their learning. We know but a few algorithms that work well for this purpose, beginning with restricted Boltzmann machines (RBMs) (Hinton et al., 2006; Hinton and Salakhutdinov, 2006; Lee et al., 2008), and autoencoders (Bengio et al., 2007; Ranzato et al., 2007), but also semi-supervised embedding (Weston et al., 2008) and kernel PCA (Cho and Saul, 2010).

It is worth mentioning here that RBMs (Hinton, 2002; Smolensky, 1986) and basic classical autoencoders are very similar in their functional form, although their interpretation and the procedures used for training them are quite different. More specifically, the deterministic function that maps from input to *mean hidden representation*, detailed below in Section 2.2, is the same for both models. One important difference is that deterministic autoencoders consider that *real valued*

1. There is a notable exception to this in the more specialized convolutional network architecture of LeCun et al. (1989).

mean as their hidden representation whereas stochastic RBMs sample a *binary* hidden representation from that mean. However, after their initial pretraining, the way layers of RBMs are typically used in practice when stacked in a deep neural network is by propagating these real-valued means (Hinton et al., 2006; Hinton and Salakhutdinov, 2006). This is more in line with the deterministic autoencoder interpretation. Note also that reconstruction error of an autoencoder can be seen as an approximation of the log-likelihood gradient in an RBM, in a way that is similar to the approximation made by using the Contrastive Divergence updates for RBMs (Bengio and Delalleau, 2009). It is thus not surprising that initializing a deep network by stacking autoencoders yields almost as good a classification performance as when stacking RBMs (Bengio et al., 2007; Larochelle et al., 2009a). But why is it only *almost* as good? An initial motivation of the research presented here was to find a way to bridge that performance gap.

With the autoencoder paradigm in mind, we began an inquiry into the question of what can shape a good, useful representation. We were looking for unsupervised learning principles likely to lead to the learning of feature detectors that detect important structure in the input patterns.

Section 2 walks the reader along the lines of our reasoning. Starting from the simple intuitive notion of preserving information, we present a generalized formulation of the classical autoencoder, before highlighting its limitations. This leads us in Section 3 to motivate an alternative *denoising* criterion, and derive the *denoising autoencoder* model, for which we also give a possible intuitive geometric interpretation. A closer look at the considered noise types will then allow us to derive a further extension of the base model. Section 4 discusses related preexisting works and approaches. Section 5 presents experiments that qualitatively study the feature detectors learnt by a single-layer denoising autoencoder under various conditions. Section 6 describes experiments with multi-layer architectures obtained by stacking denoising autoencoders and compares their classification performance with other state-of-the-art models. Section 7 is an attempt at turning stacked (denoising) autoencoders into practical generative models, to allow for a qualitative comparison of generated samples with DBNs. Section 8 summarizes our findings and concludes our work.

1.1 Notation

We will be using the following notation throughout the article:

- Random variables are written in upper case, for example, X.
- If X is a random vector, then its j^{th} *component* will be noted X_j.
- Ordinary vectors are written in lowercase bold. For example, a realization of a random vector X may be written $\mathbf{x}$. Vectors are considered column vectors.
- Matrices are written in uppercase bold (e.g., $\mathbf{W}$). $\mathbf{I}$ denotes the identity matrix.
- The transpose of a vector $\mathbf{x}$ or a matrix $\mathbf{W}$ is written $\mathbf{x}^T$ or $\mathbf{W}^T$ (*not* $\mathbf{x}'$ or $\mathbf{W}'$ which may be used to refer to an entirely different vector or matrix).
- We use lower case p and q to denote both probability density functions or probability mass functions according to context.
- Let X and Y two random variables with marginal probability $p(X)$ and $p(Y)$. Their joint probability is written $p(X,Y)$ and the conditional $p(X|Y)$.
- We may use the following common shorthands when unambiguous: $p(\mathbf{x})$ for $p(X=\mathbf{x})$; $p(X|\mathbf{y})$ for $p(X|Y=\mathbf{y})$ (denoting a conditional distribution) and $p(\mathbf{x}|\mathbf{y})$ for $p(X=\mathbf{x}|Y=\mathbf{y})$.

不含糊的

- f, g, h, will be used for ordinary functions.
- Expectation (discrete case, p is probability mass): $\mathbb{E}_{p(X)}[f(X)] = \sum_{\mathbf{x}} p(X = \mathbf{x}) f(\mathbf{x})$.
- Expectation (continuous case, p is probability density): $\mathbb{E}_{p(X)}[f(X)] = \int p(\mathbf{x}) f(\mathbf{x}) \mathrm{d}\mathbf{x}$.
- Entropy or differential entropy: $\mathbb{H}(X) = \mathbb{H}(p) = \mathbb{E}_{p(X)}[-\log p(X)]$.
- Conditional entropy: $\mathbb{H}(X|Y) = \mathbb{E}_{p(X,Y)}[-\log p(X|Y)]$.
- Kullback-Leibler divergence: $\mathbb{D}_{\mathsf{KL}}(p\|q) = \mathbb{E}_{p(X)}[\log \frac{p(X)}{q(X)}]$.
- Cross-entropy: $\mathbb{H}(p\|q) = \mathbb{E}_{p(X)}[-\log q(X)] = \mathbb{H}(p) + \mathbb{D}_{\mathsf{KL}}(p\|q)$.
- Mutual information: $\mathbb{I}(X;Y) = \mathbb{H}(X) - \mathbb{H}(X|Y)$.
- Sigmoid: $s(x) = \frac{1}{1+e^{-x}}$ and $s(\mathbf{x}) = (s(\mathbf{x}_1), \ldots, s(\mathbf{x}_d))^T$.
- Bernoulli distribution with mean μ: $\mathcal{B}(\mu)$. By extension for vector variables: $X \sim \mathcal{B}(\mu)$ means $\forall i, X_i \sim \mathcal{B}(\mu_i)$.

1.2 General setup

We consider the typical supervised learning setup with a training set of n (input, target) pairs $D_n = \{(\mathbf{x}^{(1)}, t^{(1)}) \ldots, (\mathbf{x}^{(n)}, t^{(n)})\}$, that we suppose to be an i.i.d. sample from an **unknown distribution** $q(X,T)$ with corresponding marginals $q(X)$ and $q(T)$. We denote $q^0(X,T)$ and $q^0(X)$ the empirical distributions defined by the samples in D_n. X is a d-dimensional random vector (typically in $\mathbb{R}^d$ or in $[0,1]^d$).

In this work we are primarily concerned with finding a new, higher-level representation Y of X. Y is a d'-dimensional random vector (typically in $\mathbb{R}^{d'}$ or in $[0,1]^{d'}$). If $d' > d$ we will talk of an *over-complete* representation, whereas it will be termed an *under-complete* representation if $d' < d$. Y may be linked to X by a deterministic or stochastic mapping $q(Y|X;\theta)$ parameterized by a vector of parameters θ.

2. What Makes a Good Representation? From Mutual Information to Autoencoders

From the outset we can give an *operational definition* of a "good" representation as one that will eventually be *useful* for addressing tasks of interest, in the sense that it will help the system quickly achieve higher performance on those tasks than if it hadn't first learned to form the representation. Based on the objective measure typically used to assess algorithm performance, this might be phrased as "A good representation is one that will yield a better performing classifier". Final classification performance will indeed typically be used to objectively compare algorithms. However, if a lesson is to be learnt from the recent breakthroughs in deep network training techniques, it is that the error signal from a single narrowly defined classification task should not be the only nor primary criterion used to *guide* the learning of representations. First because it has been shown experimentally that beginning by optimizing an unsupervised criterion, oblivious of the specific classification problem, can actually greatly help in eventually achieving superior performance for that classification problem. Second it can be argued that the capacity of humans to quickly become proficient in new tasks builds on much of what they have learnt *prior* to being faced with that task.

In this section, we begin with the simple notion of retaining information and progress to formally introduce the traditional *autoencoder* paradigm from this more general vantage point.

2.1 Retaining Information about the Input

We are interested in learning a (possibly stochastic) mapping from input X to a novel representation Y. To make this more precise, let us restrict ourselves to parameterized mappings $q(Y|X) = q(Y|X;\theta)$ with parameters θ that we want to learn.

One natural criterion that we may expect any good *representation* to meet, at least to some degree, is to retain a significant amount of information about the *input*. It can be expressed in information-theoretic terms as maximizing the mutual information $\mathbb{I}(X;Y)$ between an input random variable X and its higher level representation Y. This is the *infomax principle* put forward by Linsker (1989).

Mutual information can be decomposed into an entropy and a conditional entropy term in two different ways. A first possible decomposition is $\mathbb{I}(X;Y) = \mathbb{H}(Y) - \mathbb{H}(Y|X)$ which lead Bell and Sejnowski (1995) to their infomax approach to Independent Component Analysis. Here we will start from another decomposition: $\mathbb{I}(X;Y) = \mathbb{H}(X) - \mathbb{H}(X|Y)$. Since observed input X comes from an unknown distribution $q(X)$ on which θ has no influence, this makes $\mathbb{H}(X)$ an unknown constant. Thus the infomax principle reduces to:

$$\begin{aligned}\arg\max_{\theta} \mathbb{I}(X;Y) &= \arg\max_{\theta} -\mathbb{H}(X|Y) \\ &= \arg\max_{\theta} \mathbb{E}_{q(X,Y)}[\log q(X|Y)].\end{aligned}$$

Now for any distribution $p(X|Y)$ we will have

$$\mathbb{E}_{q(X,Y)}[\log p(X|Y)] \leq \underbrace{\mathbb{E}_{q(X,Y)}[\log q(X|Y)]}_{-\mathbb{H}(X|Y)}, \tag{1}$$

as can easily be shown starting from the property that for any two distributions p and q we have $\mathbb{D}_{\mathsf{KL}}(q\|p) \geq 0$, and in particular $\mathbb{D}_{\mathsf{KL}}(q(X|Y=\mathbf{y})\|p(X|Y=\mathbf{y})) \geq 0$.

Let us consider a parametric distribution $p(X|Y;\theta')$, parameterized by θ', and the following optimization:

$$\max_{\theta,\theta'} \mathbb{E}_{q(X,Y;\theta)}[\log p(X|Y;\theta')].$$

From Equation 1, we see that this corresponds to maximizing a lower bound on $-\mathbb{H}(X|Y)$ and thus on the mutual information. We would end up maximizing the *exact* mutual information provided $\exists\theta'$ s.t. $q(X|Y) = p(X|Y;\theta')$.

If, as is done in infomax ICA, we further restrict ourselves to a deterministic mapping from X to Y, that is, representation Y is to be computed by a parameterized function $Y = f_\theta(X)$ or equivalently $q(Y|X;\theta) = \delta(Y - f_\theta(X))$ (where δ denotes Dirac-delta), then this optimization can be written:

$$\max_{\theta,\theta'} \mathbb{E}_{q(X)}[\log p(X|Y = f_\theta(X);\theta')].$$

This again corresponds to maximizing a lower bound on the mutual information.

Since $q(X)$ is unknown, but we have samples from it, the empirical average over the training samples can be used instead as an unbiased estimate (i.e., replacing $\mathbb{E}_{q(X)}$ by $\mathbb{E}_{q^0(X)}$):

$$\max_{\theta,\theta'} \mathbb{E}_{q^0(X)}[\log p(X|Y = f_\theta(X);\theta')]. \tag{2}$$

We will see in the next section that this equation corresponds to the *reconstruction error* criterion used to train *autoencoders*.

2.2 Traditional Autoencoders (AE)

Here we briefly specify the traditional *autoencoder* (AE)[2] framework and its terminology, based on f_θ and $p(X|Y;\theta')$ introduced above.

Encoder: The deterministic mapping f_θ that transforms an input vector $\mathbf{x}$ into hidden representation $\mathbf{y}$ is called the **encoder**. Its typical form is an affine mapping followed by a nonlinearity:

$$f_\theta(\mathbf{x}) = s(\mathbf{W}\mathbf{x}+\mathbf{b}).$$

Its parameter set is $\theta = \{\mathbf{W},\mathbf{b}\}$, where $\mathbf{W}$ is a $d' \times d$ weight matrix and $\mathbf{b}$ is an offset vector of dimensionality d'.

Decoder: The resulting hidden representation $\mathbf{y}$ is then mapped back to a reconstructed d-dimensional vector $\mathbf{z}$ in input space, $\mathbf{z} = g_{\theta'}(\mathbf{y})$. This mapping $g_{\theta'}$ is called the **decoder**. Its typical form is again an affine mapping optionally followed by a squashing non-linearity, that is, either $g_{\theta'}(\mathbf{y}) = \mathbf{W}'\mathbf{y}+\mathbf{b}'$ or

$$g_{\theta'}(\mathbf{y}) = s(\mathbf{W}'\mathbf{y}+\mathbf{b}'), \tag{3}$$

with appropriately sized parameters $\theta' = \{\mathbf{W}',\mathbf{b}'\}$.

In general $\mathbf{z}$ is not to be interpreted as an exact reconstruction of $\mathbf{x}$, but rather in probabilistic terms as the parameters (typically the mean) of a distribution $p(X|Z=\mathbf{z})$ that may generate $\mathbf{x}$ with high probability. We have thus completed the specification of $p(X|Y;\theta')$ from the previous section as $p(X|Y=\mathbf{y}) = p(X|Z=g_{\theta'}(\mathbf{y}))$. This yields an associated reconstruction error to be optimized:

$$L(\mathbf{x},\mathbf{z}) \propto -\log p(\mathbf{x}|\mathbf{z}). \tag{4}$$

Common choices for $p(\mathbf{x}|\mathbf{z})$ and associated loss function $L(\mathbf{x},\mathbf{z})$ include:

- For real-valued $\mathbf{x}$, that is, $\mathbf{x} \in \mathbb{R}^d$: $X|\mathbf{z} \sim \mathcal{N}(\mathbf{z},\sigma^2\mathbf{I})$, that is, $X_j|\mathbf{z} \sim \mathcal{N}(\mathbf{z}_j,\sigma^2)$.
  This yields $L(\mathbf{x},\mathbf{z}) = L_2(\mathbf{x},\mathbf{z}) = C(\sigma^2)\|\mathbf{x}-\mathbf{z}\|^2$ where $C(\sigma^2)$ denotes a constant that depends only on σ^2 and that can be ignored for the optimization. This is the *squared error* objective found in most traditional autoencoders. In this setting, due to the Gaussian interpretation, it is more natural *not* to use a squashing nonlinearity in the decoder.

- For binary $\mathbf{x}$, that is, $\mathbf{x} \in \{0,1\}^d$: $X|\mathbf{z} \sim \mathcal{B}(\mathbf{z})$, that is, $X_j|\mathbf{z} \sim \mathcal{B}(\mathbf{z}_j)$.
  In this case, the decoder needs to produce a $\mathbf{z} \in [0,1]^d$. So a squashing nonlinearity such as a sigmoid s will typically be used in the decoder. This yields $L(\mathbf{x},\mathbf{z}) = L_{\mathbb{H}}(\mathbf{x},\mathbf{z}) = -\sum_j[\mathbf{x}_j\log\mathbf{z}_j + (1-\mathbf{x}_j)\log(1-\mathbf{z}_j)] = \mathbb{H}(\mathcal{B}(\mathbf{x})\|\mathcal{B}(\mathbf{z}))$ which is termed the *cross-entropy loss* because it is seen as the cross-entropy between two independent multivariate Bernoullis, the first with mean $\mathbf{x}$ and the other with mean $\mathbf{z}$. This loss can also be used when $\mathbf{x}$ is not strictly binary but rather $\mathbf{x} \in [0,1]^d$.

2. Note: *AutoEncoders* (AE) are also often called *AutoAssociators* (AA) in the literature. The shorter autoencoder term was preferred in this work, as we believe *encoding* better conveys the idea of producing a novel useful representation. Similarly, what we call Stacked Auto Encoders (SAE) has also been called Stacked AutoAssociators (SAA).

Note that in the general autoencoder framework, we may use other forms of parameterized functions for the encoder or decoder, and other suitable choices of the loss function (corresponding to a different $p(X|z)$). In particular, we investigated the usefulness of a more complex encoding function in Larochelle, Erhan, and Vincent (2009b). For the experiments in the present work however, we will restrict ourselves to the two usual forms detailed above, that is, an **affine+sigmoid encoder** and either **affine decoder with squared error loss** or **affine+sigmoid decoder with cross-entropy loss**. A further constraint that can optionally be imposed, and that further parallels the workings of RBMs, is having *tied weights* between W and W', in effect defining W' as $W' = W^T$.

Autoencoder training consists in minimizing the reconstruction error, that is, carrying the following optimization:

$$\arg\min_{\theta,\theta'} \mathbb{E}_{q^0(X)}[L(X, Z(X))],$$

where we wrote $Z(X)$ to emphasize the fact that Z is a deterministic function of X, since Z is obtained by composition of deterministic encoding and decoding.

Making this explicit and using our definition of loss L from Equation 4 this can be rewritten as:

$$\arg\max_{\theta,\theta'} \mathbb{E}_{q^0(X)}[\log p(X|Z = g_{\theta'}(f_\theta(X)))],$$

or equivalently

$$\arg\max_{\theta,\theta'} \mathbb{E}_{q^0(X)}[\log p(X|Y = f_\theta(X); \theta')].$$

We see that this last line corresponds to Equation 2, that is, the maximization of a lower bound on the mutual information between X and Y.

It can thus be said that **training an autoencoder to minimize reconstruction error amounts to maximizing a lower bound on the mutual information between input X and learnt representation Y**. Intuitively, if a representation allows a good reconstruction of its input, it means that it has retained much of the information that was present in that input.

2.3 Merely Retaining Information is Not Enough

The criterion that representation Y should retain information about input X is not by itself sufficient to yield a useful representation. Indeed mutual information can be trivially maximized by setting $Y = X$. Similarly, an ordinary autoencoder where Y is of the same dimensionality as X (or larger) can achieve perfect reconstruction simply by learning an identity mapping.[3] Without any other constraints, this criterion alone is unlikely to lead to the discovery of a more useful representation than the input.

Thus further constraints need to be applied to attempt to separate useful information (to be retained) from noise (to be discarded). This will naturally translate to non-zero reconstruction error. The traditional approach to autoencoders uses a *bottleneck* to produce an *under-complete* representation where $d' < d$. The resulting lower-dimensional Y can thus be seen as a *lossy compressed representation* of X. When using affine encoder and decoder without any nonlinearity and a squared error loss, the autoencoder essentially performs principal component analysis (PCA) as showed by

3. More precisely, it suffices that $g \circ f$ be the identity to obtain zero reconstruction error. For $d = d'$ if we had a linear encoder and decoder this would be achieved for any invertible matrix $\mathbf{W}$ by setting $\mathbf{W}' = \mathbf{W}^{-1}$. Now there is a sigmoid nonlinearity in the encoder, but it is possible to stay in the linear part of the sigmoid with small enough $\mathbf{W}$.

Baldi and Hornik (1989).[4] When a nonlinearity such as a sigmoid is used in the encoder, things become a little more complicated: obtaining the PCA subspace is a likely possibility (Bourlard and Kamp, 1988) since it is possible to stay in the linear regime of the sigmoid, but arguably not the only one (Japkowicz et al., 2000). Also when using a cross-entropy loss rather than a squared error the optimization objective is no longer the same as that of PCA and will likely learn different features. The use of "tied weights" can also change the solution: forcing encoder and decoder matrices to be symmetric and thus have the same scale can make it harder for the encoder to stay in the linear regime of its nonlinearity without paying a high price in reconstruction error.

Alternatively it is also conceivable to impose on Y different constraints than that of a lower dimensionality. In particular the possibility of using *over-complete* (i.e., higher dimensional than the input) but *sparse* representations has received much attention lately. Interest in sparse representations is inspired in part by evidence that neural activity in the brain seems to be sparse and has burgeoned following the seminal work of Olshausen and Field (1996) on *sparse coding*. Other motivations for sparse representations include the ability to handle effectively variable-size representations (counting only the non-zeros), and the fact that dense compressed representations tend to entangle information (i.e., changing a single aspect of the input yields significant changes in all components of the representation) whereas sparse ones can be expected to be easier to interpret and to use for a subsequent classifier. Various modifications of the traditional autoencoder framework have been proposed in order to learn sparse representations (Ranzato et al., 2007, 2008). These were shown to extract very useful representations, from which it is possible to build top performing deep neural network classifiers. A sparse over-complete representations can be viewed as an alternative "compressed" representation: it has *implicit* straightforward compressibility due to the large number of zeros rather than an explicit lower dimensionality.

3. Using a Denoising Criterion

We have seen that the reconstruction criterion alone is unable to guarantee the extraction of useful features as it can lead to the obvious solution "simply copy the input" or similarly uninteresting ones that trivially maximizes mutual information. One strategy to avoid this phenomenon is to constrain the representation: the traditional bottleneck and the more recent interest on sparse representations both follow this strategy.

Here we propose and explore a very different strategy. Rather than constrain the representation, we change the reconstruction criterion for a both more challenging and more interesting objective: cleaning partially corrupted input, or in short *denoising*. In doing so we modify the implicit definition of a good representation into the following: *"a good representation is one that can be obtained robustly from a corrupted input and that will be useful for recovering the corresponding clean input"*. Two underlying ideas are implicit in this approach:

- First it is expected that a higher level representation should be rather stable and robust under corruptions of the input.
- Second, it is expected that performing the denoising task well requires extracting features that capture useful structure in the input distribution.

4. More specifically it will find the same *subspace* as PCA, but the specific projection directions found will in general not correspond to the actual principal directions and need not be orthonormal.

We emphasize here that our goal is *not* the task of denoising per se. Rather **denoising is advocated and investigated as a *training criterion* for learning to extract useful features** that will constitute better higher level representation. The usefulness of a learnt representation can then be assessed objectively by measuring the accuracy of a classifier that uses it as input.

3.1 The Denoising Autoencoder Algorithm

This approach leads to a very simple variant of the basic autoencoder described above. A *denoising autoencoder (DAE)* is trained to reconstruct a clean "repaired" input from a *corrupted* version of it (the specific types of corruptions we consider will be discussed below). This is done by first corrupting the initial input $\mathbf{x}$ into $\tilde{\mathbf{x}}$ by means of a stochastic mapping $\tilde{\mathbf{x}} \sim q_{\mathcal{D}}(\tilde{\mathbf{x}}|\mathbf{x})$.

Corrupted input $\tilde{\mathbf{x}}$ is then mapped, as with the basic autoencoder, to a hidden representation $\mathbf{y} = f_\theta(\tilde{\mathbf{x}}) = s(\mathbf{W}\tilde{\mathbf{x}} + \mathbf{b})$ from which we reconstruct a $\mathbf{z} = g_{\theta'}(\mathbf{y})$. See Figure 1 for a schematic representation of the procedure. Parameters θ and θ' are trained to minimize the average reconstruction error over a training set, that is, to have $\mathbf{z}$ as close as possible to the *uncorrupted* input $\mathbf{x}$. The key difference is that $\mathbf{z}$ is now a deterministic function of $\tilde{\mathbf{x}}$ rather than $\mathbf{x}$. As previously, the considered reconstruction error is either the cross-entropy loss $L_{\mathbb{H}}(\mathbf{x},\mathbf{z}) = \mathbb{H}(\mathcal{B}(\mathbf{x})\|\mathcal{B}(\mathbf{z}))$, with an affine+sigmoid decoder, or the squared error loss $L_2(\mathbf{x},\mathbf{z}) = \|\mathbf{x}-\mathbf{z}\|^2$, with an affine decoder. Parameters are initialized at random and then optimized by stochastic gradient descent. Note that each time a training example $\mathbf{x}$ is presented, a different corrupted version $\tilde{\mathbf{x}}$ of it is generated according to $q_{\mathcal{D}}(\tilde{\mathbf{x}}|\mathbf{x})$.

Note that denoising autoencoders are still minimizing the same reconstruction loss between a clean X and its reconstruction from Y. So this still amounts to maximizing a lower bound on the mutual information between clean input X and representation Y. The difference is that Y is now obtained by applying deterministic mapping f_θ to a *corrupted* input. It thus forces the learning of a far more clever mapping than the identity: one that extracts features useful for *denoising*.

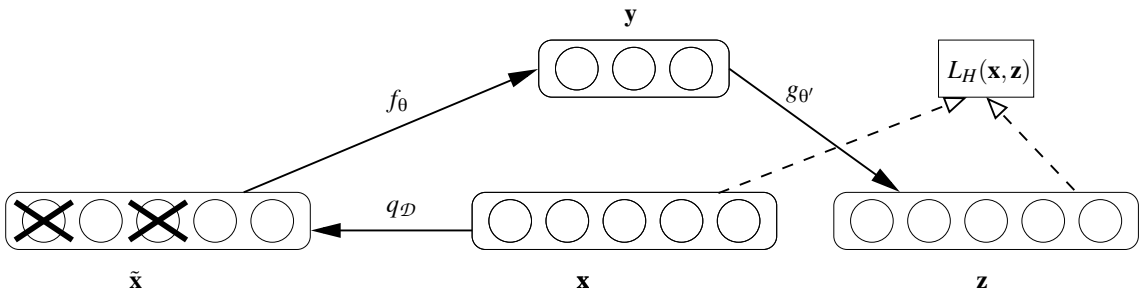


Figure 1: The denoising autoencoder architecture. An example $\mathbf{x}$ is stochastically corrupted (via $q_{\mathcal{D}}$) to $\tilde{\mathbf{x}}$. The autoencoder then maps it to $\mathbf{y}$ (via encoder f_θ) and attempts to reconstruct $\mathbf{x}$ via decoder $g_{\theta'}$, producing reconstruction $\mathbf{z}$. Reconstruction error is measured by loss $L_H(\mathbf{x},\mathbf{z})$.

3.2 Geometric Interpretation

The process of denoising, that is, mapping a corrupted example back to an uncorrupted one, can be given an intuitive geometric interpretation under the so-called *manifold assumption* (Chapelle et al., 2006), which states that natural high dimensional data concentrates close to a non-linear low-dimensional manifold. This is illustrated in Figure 2. During denoising training, we learn a stochastic operator $p(X|\widetilde{X})$ that maps a corrupted $\widetilde{X}$ back to its uncorrupted X, for example, in the case of binary data,

$$X|\widetilde{X} \sim \mathcal{B}(g_{\theta'}(f_\theta(\widetilde{X}))).$$

Corrupted examples are much more likely to be outside and farther from the manifold than the uncorrupted ones. Thus stochastic operator $p(X|\widetilde{X})$ learns a map that tends to go from lower probability points $\widetilde{X}$ to nearby high probability points X, on or near the manifold. Note that when $\widetilde{X}$ is farther from the manifold, $p(X|\widetilde{X})$ should learn to make bigger steps, to reach the manifold. Successful denoising implies that the operator maps even far away points to a small region close to the manifold.

The denoising autoencoder can thus be seen as a way to define and learn a manifold. In particular, if we constrain the dimension of Y to be smaller than the dimension of X, then the intermediate representation $Y = f(X)$ may be interpreted as a coordinate system for points on the manifold. More generally, one can think of $Y = f(X)$ as a representation of X which is well suited to capture the main variations in the data, that is, those along the manifold.

3.3 Types of Corruption Considered

The above principle and technique can potentially be used with any type of corruption process. Also the corruption process is an obvious place where prior knowledge, if available, could be easily incorporated. But in the present study we set to investigate a technique that is generally applicable. In

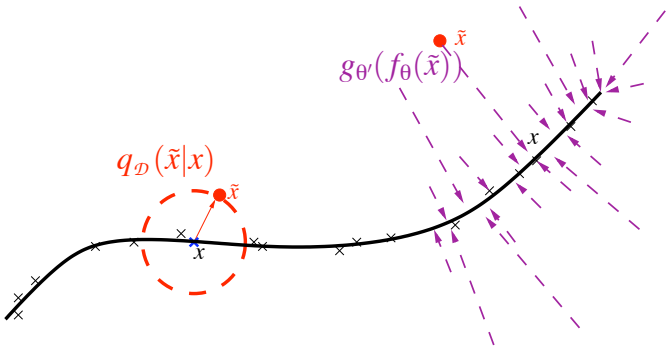


Figure 2: Manifold learning perspective. Suppose training data ($\times$) concentrate near a low-dimensional manifold. Corrupted examples ($\bullet$) obtained by applying corruption process $q_{\mathcal{D}}(\widetilde{X}|X)$ will generally lie farther from the manifold. The model learns with $p(X|\widetilde{X})$ to "project them back" (via autoencoder $g'_\theta(f_\theta(\cdot))$) onto the manifold. Intermediate representation $Y = f_\theta(X)$ may be interpreted as a coordinate system for points X on the manifold.

particular we want it to be usable for learning ever higher level representations by *stacking* denoising autoencoders. Now while prior knowledge on relevant corruption processes may be available in a particular input space (such as images), such prior knowledge will not be available for the space of intermediate-level representations.

We will thus restrict our discussion and experiments to the following simple corruption processes:

- Additive isotropic *Gaussian noise* (GS): $\tilde{\mathbf{x}}|\mathbf{x} \sim \mathcal{N}(\mathbf{x}, \sigma^2 I)$;
- *Masking noise* (MN): a fraction ν of the elements of $\mathbf{x}$ (chosen at random for each example) is forced to 0;
- *Salt-and-pepper noise* (SP): a fraction ν of the elements of $\mathbf{x}$ (chosen at random for each example) is set to their minimum or maximum possible value (typically 0 or 1) according to a fair coin flip.

Additive Gaussian noise is a very common noise model, and is a natural choice for real valued inputs. The *salt-and-pepper noise* will also be considered, as it is a natural choice for input domains which are interpretable as binary or near binary such as black and white images or the representations produced at the hidden layer after a sigmoid squashing function.

Much of our work however, both for its inspiration and in experiments, focuses on *masking noise* which can be viewed as turning off components considered missing or replacing their value by a default value—that is, a common technique for handling missing values. All information about these masked components is thus removed from that particular input pattern, and we can view the denoising autoencoder as trained to *fill-in* these artificially introduced "blanks". Also, numerically, forcing components to zero means that they are totally ignored in the computations of downstream neurons.

We draw the reader's attention to the fact that both salt-and-pepper and masking noise drastically corrupt but a fraction of the elements while leaving the others untouched. Denoising, that is, recovering the values of the corrupted elements, will only be possible thanks to dependencies between dimensions in high dimensional distributions. Denoising training is thus expected to capture these dependencies. The approach probably makes less sense for very low dimensional problems, at least with these types of corruption.

3.4 Extension: Putting an Emphasis on Corrupted Dimensions

Noise types such as *masking noise* and *salt-and-pepper* that erase only a changing subset of the input's components while leaving the others untouched suggest a straightforward extension of the denoising autoencoder criterion. Rather than giving equal weight to the reconstruction of all components of the input, we can put an *emphasis* on the corrupted dimensions. To achieve this we give a different weight α for the reconstruction error on components that were corrupted, and β for those that were left untouched. α and β are considered hyperparameters.

For the squared loss this yields

$$L_{2,\alpha}(\mathbf{x}, \mathbf{z}) = \alpha \left(\sum_{j \in \mathcal{J}(\tilde{\mathbf{x}})} (\mathbf{x}_j - \mathbf{z}_j)^2 \right) + \beta \left(\sum_{j \notin \mathcal{J}(\tilde{\mathbf{x}})} (\mathbf{x}_j - \mathbf{z}_j)^2 \right),$$

where $\mathcal{J}(\tilde{\mathbf{x}})$ denotes the indexes of the components of $\mathbf{x}$ that were corrupted.

And for the cross-entropy loss this yields

$$L_{\mathsf{IH},\alpha}(\mathbf{x},\mathbf{z}) = \alpha\left(-\sum_{j\in\mathcal{J}(\tilde{\mathbf{x}})}[\mathbf{x}_j\log\mathbf{z}_j+(1-\mathbf{x}_j)\log(1-\mathbf{z}_j)]\right) + \beta\left(-\sum_{j\notin\mathcal{J}(\tilde{\mathbf{x}})}[\mathbf{x}_j\log\mathbf{z}_j+(1-\mathbf{x}_j)\log(1-\mathbf{z}_j)]\right).$$

We call this extension *emphasized denoising autoencoder*. A special case that we call *full emphasis* is obtained for $\alpha = 1,\ \beta = 0$ where we only take into account the error on the prediction of corrupted elements.

3.5 Stacking Denoising Autoencoders to Build Deep Architectures

Stacking denoising autoencoders to initialize a deep network works in much the same way as stacking RBMs in deep belief networks (Hinton et al., 2006; Hinton and Salakhutdinov, 2006) or ordinary autoencoders (Bengio et al., 2007; Ranzato et al., 2007; Larochelle et al., 2009a). Let us specify that input corruption is only used for the initial denoising-training of each individual layer, so that it may learn useful feature extractors. Once the mapping f_θ has thus been learnt, it will henceforth be used on *uncorrupted* inputs. In particular no corruption is applied to produce the representation that will serve as clean input for training the next layer. The complete procedure for learning and stacking several layers of denoising autoencoders is shown in Figure 3.

Once a stack of encoders has thus been built, its highest level output representation can be used as input to a stand-alone supervised learning algorithm, for example a Support Vector Machine classifier or a (multi-class) logistic regression. Alternatively, as illustrated in Figure 4, a logistic regression layer can be added on top of the encoders, yielding a *deep neural network* amenable to supervised learning. The parameters of all layers can then be simultaneously *fine-tuned* using a gradient-based procedure such as stochastic gradient descent.

4. Related Approaches in the Literature

In this section, we briefly review and discuss related prior work along three different axes.

4.1 Previous Work on Training Neural Networks for Denoising

The idea of training a multi-layer perceptron using error backpropagation on a denoising task is not new. The approach was first introduced by LeCun (1987) and Gallinari et al. (1987) as an alternative method to learn an *(auto-)associative memory* similar to how Hopfield Networks (Hopfield, 1982) were understood. The networks were trained and tested on binary input patterns, corrupted by flipping a fraction of input bits chosen at random. Both the model and training procedure in this precursory work are very similar to the denoising autoencoder we describe.[5] Our motivation and goal are however quite different. The objective of LeCun (1987) was to study the *capacity* of such a network for memorization tasks, that is, counting how many training patterns it was able to

5. There are a few minor differences; for example, the use of a squared error after sigmoid for binary data, while we tend to use a cross-entropy loss. Also their denoising procedure considers doing several recurrent passes through the autoencoder network, as in a recurrent net.

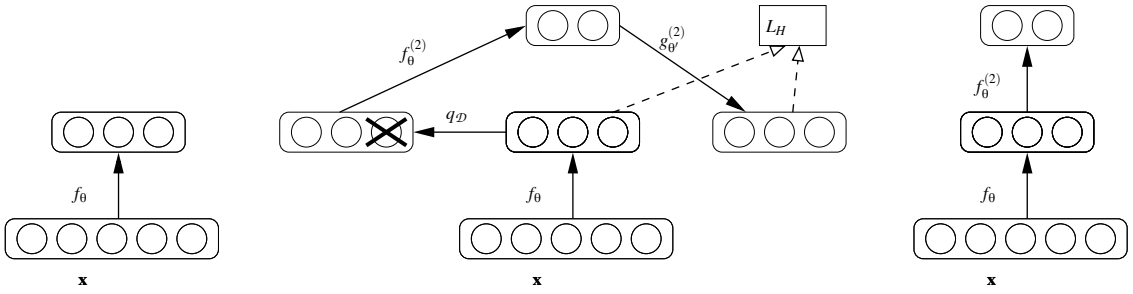


Figure 3: Stacking denoising autoencoders. After training a first level denoising autoencoder (see Figure 1) its learnt encoding function f_θ is used on clean input (left). The resulting representation is used to train a second level denoising autoencoder (middle) to learn a second level encoding function $f_\theta^{(2)}$. From there, the procedure can be repeated (right).

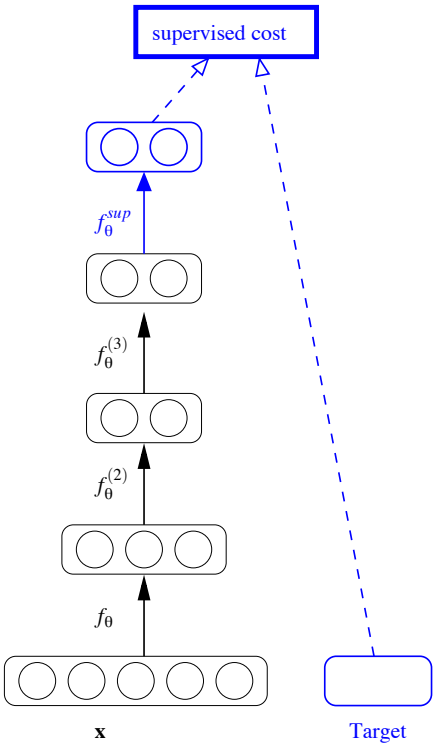


Figure 4: Fine-tuning of a deep network for classification. After training a stack of encoders as explained in the previous figure, an output layer is added on top of the stack. The parameters of the whole system are fine-tuned to minimize the error in predicting the supervised target (e.g., class), by performing gradient descent on a supervised cost.

correctly recall under these conditions. The work also clearly established the usefulness of a non-linear hidden layer for this. By contrast, our work is motivated by the search and understanding of unsupervised pretraining criteria to initialize deep networks. Our primary interest is thus in investigating the ability of the denoising criterion to learn good feature extractors, with which to initialize a deep network by stacking and composing these feature extractors. We focus on analyzing the learnt higher-level representations and their effect on the classification performance of resulting deep networks.

Another insightful work that is very much related to the approach advocated here is the research of Seung (1998), in which a recurrent neural network is trained to complete corrupted input patterns using backpropagation through time. Both the work of Seung (1998) and that of LeCun (1987) and Gallinari et al. (1987) appear to be inspired by Hopfield-type associative memories (Hopfield, 1982) in which learnt patterns are conceived as attractive fixed points of a recurrent network dynamic. Seung (1998) contributes a very interesting analysis in terms of continuous attractors, points out the limitations of regular autoencoding, and advocates the pattern completion task as an alternative to density estimation for unsupervised learning. Again, it differs form our study mainly by its *focus* a) on recurrent networks[6] and b) on the image denoising task per se. The latter justifies their use of prior knowledge of the 2D topology of images, both in the architectural choice of local 2D receptive field connectivity, and in the corruption process that consists in zeroing-out a square image patch at a random position. This occlusion by a 2D patch is a special form of a *structured* masking noise, where the a-priori known 2D topological structure of images is taken into account. In our research we deliberately chose not to use topological prior knowledge in our model nor in our corruption process, so that the *same generic procedure* may be applied to learn higher levels of representation from lower ones, or to other domains for which we have no such topological prior knowledge.

More recently Jain and Seung (2008) presented a very interesting and successful approach for image denoising, that consists in layer-wise building of a deep convolutional neural network. Their algorithm yields comparable or better performance than state-of-the-art Markov Random Field and wavelet methods developed for image denoising. The approach clearly has roots in their earlier work (Seung, 1998) and appears also inspired by more recent research on deep network pretraining, including our own group's (Bengio et al., 2007). But apparently, neither of us was initially aware of the other group's relevant work on denoising (Vincent et al., 2008; Jain and Seung, 2008). Again the focus of Seung (1998) on image denoising per se differs from our own focus on studying deep network pretraining for classification tasks and results in marked differences in the actual algorithms. Specifically, in Jain and Seung (2008) each layer in the stack is trained to reconstruct the original clean image a little better, which makes sense for image denoising. This can be contrasted with our approach, in which upper layers are trained to denoise-and-reconstruct whatever representation they receive from the layer immediately below, rather than to restore the original input image in one operation. This logically follows from our search for a generic feature extraction algorithm for pretraining, where upper level representations will eventually be used for a totally different task such as classification.

6. Note however that a recurrent network can be seen as deep network, with the additional property that all layers share the same weights.

4.2 Training Classifiers with Noisy Inputs

The idea of training a neural network with noisy input (Scalettar and Zee, 1988; von Lehman et al., 1988)—or training with *jitter* as it is sometimes called—has been proposed to enhance generalization performance for supervised learning tasks (Sietsma and Dow, 1991; Holmstrm and Koistinen, 1992; An, 1996). This thread of research is less directly related to autoencoders and denoising than the studies discussed in the previous section. It is nevertheless relevant. After all, denoising amounts to using noisy patterns as input with the clean pattern as a supervised target, albeit a rather high dimensional one. It has been argued that training with noise is equivalent to applying generalized Tikhonov regularization (Bishop, 1995). On the surface, this may seem to suggest that training with noisy inputs has a similar effect to training with an L2 weight decay penalty (i.e., penalizing the sum of squared weights), but this view is incorrect. Tikhonov regularization applied to *linear regression* is indeed equivalent to a L2 weight decay penalty (i.e., ridge regression). But for a non-linear mapping such as a neural network, Tikhonov regularization is no longer so simple (Bishop, 1995). More importantly, in the non-linear case, the equivalence of noisy training with Tikhonov regularization is derived from a Taylor series expansion, and is thus only valid in the limit of very small additive noise. See Grandvalet et al. (1997) for a theoretical study and discussion regarding the limitations of validity for this equivalence. Last but not least, our experimental results in Section 5.1 clearly show qualitatively very different results when using denoising autoencoders (i.e., noisy inputs) than when using regular autoencoders with a L2 weight decay.

Here, we must also mention a well-known technique to improve the generalization performance of a classifier (neural network or other), which consists in augmenting the original training set with additional distorted inputs, either explicitly (Baird, 1990; Poggio and Vetter, 1992) or virtually through a modified algorithm (Simard et al., 1992; Schölkopf et al., 1996). For character images for instance such distortions may include small translations, rotations, scalings and shearings of the image, or even applying a scanner noise model. This technique can thus be seen as training with noisy corrupted inputs, but with a highly structured corruption process based on *much* prior knowledge.[7] As already explained and motivated above, our intent in this work is to develop and investigate a generally applicable technique, that should also be applicable to intermediate higher level representations. Thus we intentionally restrict our study to very simple generic corruption processes that do not incorporate explicit prior knowledge.

We also stress the difference between the approaches we just discussed, that consist in training a neural network by optimizing a *global supervised criterion using noisy input*, and the approach we investigate in the present work, that is, using a *local unsupervised denoising criterion to pretrain each layer of the network* with the goal to learn useful intermediate representations. We shall see in experimental Section 6.4 that the latter applied to a deep network yields better classification performance than the former.

4.3 Pseudo-Likelihood and Dependency Networks

The view of denoising training as "filling in the blanks" that motivated the masking noise and the extension that puts an emphasis on corrupted dimensions presented in Section 3.4, can also be related to the pseudo-likelihood (Besag, 1975) and Dependency Network (Heckerman et al., 2000)

7. Clearly, simple Tikhonov regularization cannot achieve the same as training with such prior knowledge based corruption process. This further illustrates the limitation of the equivalence between training with noise and Tikhonov regularization.

paradigms. Maximizing pseudo-likelihood instead of likelihood implies replacing the likelihood term $p(X)$ by the product of conditionals $\prod_{i=1}^{d} p(X_i|X_{\neg i})$. Here X_i denotes the i^{th} component of input vector variable X and $X_{\neg i}$ denotes all components but the i^{th}. Similarly, in the Dependency Network approach of Heckerman et al. (2000) one learns d conditional distributions, each trained to predict component i given (a subset of) all other components. This is in effect what an *emphasized denoising autoencoder* with a masking noise that masks but one input component ($\nu = \frac{1}{d}$), and a *full emphasis* ($\alpha = 1,\ \beta = 0$), is trained to do. More specifically, for binary variables it will learn to predict $p(X_i = 1|X_{\neg i})$; and when using squared error for real-valued variables it will learn to predict $\mathbb{E}[X_i|X_{\neg i}]$ assuming $X_i|X_{\neg i} \sim \mathcal{N}(\mathbb{E}[X_i|X_{\neg i}], \sigma^2)$. Note that with denoising autoencoders, all d conditional distributions are constrained to *share common parameters*, which define the mapping to and from hidden representation Y. Also when the emphasis is not fully put on the corrupted components ($\beta > 0$) some of the network's capacity will be devoted to encoding/decoding uncorrupted components.

A more important difference can be appreciated by considering the following scenario: What happens if components of the input come in identical pairs? In that case, conditional distribution $p(X_i|X_{\neg i})$ can simply learn to replicate the other component of the pair, thus not capturing any other potentially useful dependency. Now for dependency networks this exact scenario is forbidden by the formal requirement that no input configuration may have zero probability. But a similar problem may occur in practice if the components in a pair are not identical but very highly correlated. By contrast, denoising autoencoders can and will typically be trained with a larger fraction ν of corrupted components, so that reliable prediction of a component cannot rely exclusively on a single other component.

5. Experiments on Single Denoising Autoencoders: Qualitative Evaluation of Learned Feature Detectors

A first series of experiments was carried out using single denoising autoencoders, that is, without any stacking nor supervised fine tuning. The goal was to examine qualitatively the kind of feature detectors learnt by a denoising criterion, for various noise types, and compare these to what ordinary autoencoders yield.

Feature detectors that correspond to the first hidden layer of a network trained on image data are straightforward to visualize. Each hidden neuron $\mathbf{y}_j$ has an associated vector of weights $\mathbf{W}_j$ that it uses to compute a dot product with an input example (before applying its non-linearity). These $\mathbf{W}_j$ vectors, the *filters*, have the same dimensionality as the input, and can thus be displayed as little images showing what aspects of the input each hidden neuron is sensitive to.

5.1 Feature Detectors Learnt from Natural Image Patches

We trained both regular autoencoders and denoising autoencoders on 12×12 patches from whitened natural scene images, made available by Olshausen (Olshausen and Field, 1996).[8] A few of these patches are shown in Figure 5 (left). For these natural image patches, we used a linear decoder and a squared reconstruction cost. Network parameters were trained from a random start,[9] using

8. More specifically randomly positioned 12×12 patches were extracted from the 20×20 patches made available by Olshausen at the following URL: `https://redwood.berkeley.edu/bruno/sparsenet/`.
9. We applied the usual random initialization heuristic in which weights are sampled independently form a uniform in range $[-\frac{1}{\sqrt{\text{fanin}}}, \frac{1}{\sqrt{\text{fanin}}}]$ where fanin in this case is the input dimension.

stochastic gradient descent to perform 500000 weight updates with a fixed learning rate of 0.05. All filters shown were from experiments with tied weights, but untied weights yielded similar results.

Figure 5 displays filters learnt by a regular *under-complete* autoencoder that used a bottleneck of 50 hidden units, as well as those learnt by an *over-complete* autoencoder using 200 hidden units. Filters obtained in the under-complete case look like very local blob detectors. No clear structure is apparent in the filters learnt in the over-complete case.

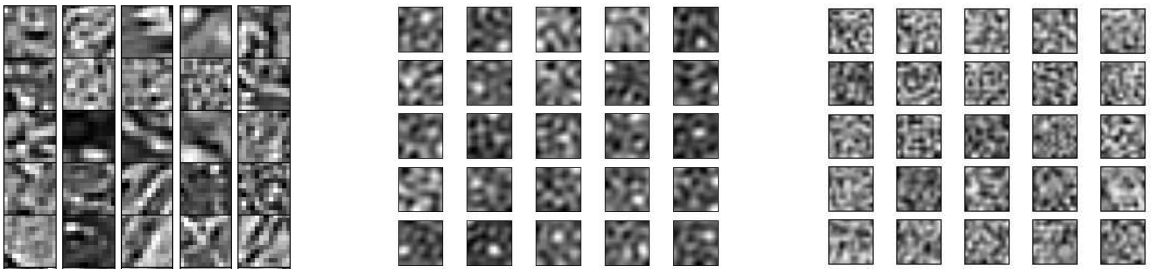


Figure 5: Regular autoencoder trained on natural image patches. *Left:* some of the 12×12 image patches used for training. *Middle:* filters learnt by a regular *under-complete* autoencoder (50 hidden units) using tied weights and L2 reconstruction error. *Right:* filters learnt by a regular *over-complete* autoencoder (200 hidden units). The under-complete autoencoder appears to learn rather uninteresting local blob detectors. Filters obtained in the over-complete case have no recognizable structure, looking entirely random.

We then trained 200 hidden units over-complete noiseless autoencoders regularized with L2 weight decay, as well as 200 hidden units denoising autoencoders with isotropic Gaussian noise (but no weight decay). Resulting filters are shown in Figure 6. Note that a denoising autoencoder with a noise level of 0 is identical to a regular autoencoder. So, naturally, filters learnt by a denoising autoencoder at small noise levels (not shown) look like those obtained with a regular autoencoder previously shown in Figure 5. With a sufficiently large noise level however ($\sigma = 0.5$), the denoising autoencoder learns Gabor-like local oriented edge detectors (see Figure 6). This is similar to what is learnt by sparse coding (Olshausen and Field, 1996, 1997), or ICA (Bell and Sejnowski, 1997) and resembles simple cell receptive fields from the primary visual cortex first studied by Hubel and Wiesel (1959). The L2 regularized autoencoder on the other hand learnt nothing interesting beyond restoring some of the local blob detectors found in the under-complete case. Note that we did try a wide range of values for the regularization hyperparameter,[10] but were never able to get Gabor-like filters. From this experiment, we see clearly that **training with sufficiently large noise yields a qualitatively very different outcome than training with a weight decay regularization**, which confirms experimentally that the two are *not* equivalent for a non-linear autoencoder, as discussed earlier in Section 4.2.

Figure 7 shows some of the results obtained with the other two noise types considered, that is, salt-and-pepper noise, and masking-noise. We experimented with 3 corruption levels ν: $10\%, 25\%, 55\%$. The filters shown were obtained using 100 hidden units, but similar filters were found with 50 or 200 hidden units. Salt-and-pepper noise yielded Gabor-like edge detectors, whereas masking noise

10. Attempted weight decays values were the following: $\lambda \in \{0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.25, 0.5, 1.0\}$.

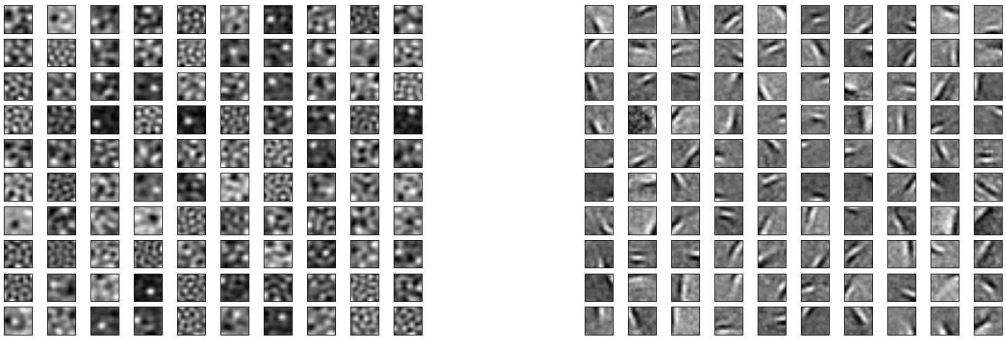


Figure 6: Weight decay vs. Gaussian noise. We show typical filters learnt from natural image patches in the over-complete case (200 hidden units). *Left:* regular autoencoder with weight decay. We tried a wide range of weight-decay values and learning rates: filters never appeared to capture a more interesting structure than what is shown here. Note that some local blob detectors are recovered compared to using no weight decay at all (Figure 5 right). *Right:* a denoising autoencoder with additive Gaussian noise ($\sigma = 0.5$) learns Gabor-like local oriented edge detectors. Clearly the filters learnt are qualitatively very different in the two cases.

yielded a mixture of edge detectors and grating filters. Clearly different corruption types and levels can yield qualitatively different filters. But it is interesting to note that all three noise types we experimented with were able to yield some potentially useful edge detectors.

5.2 Feature Detectors Learnt from Handwritten Digits

We also trained denoising autoencoders on the 28×28 gray-scale images of handwritten digits from the MNIST data set. For this experiment, we used denoising autoencoders with tied weights, cross-entropy reconstruction error, and zero-masking noise. The goal was to better understand the qualitative effect of the noise level. So we trained several denoising autoencoders, all starting from *the same initial random point in weight space*, but *with different noise levels*. Figure 8 shows some of the resulting filters learnt and how they are affected as we increase the level of corruption. With 0% corruption, the majority of the filters appear totally random, with only a few that specialize as little ink blob detectors. With increased noise levels, a much larger proportion of interesting (visibly non random and with a clear structure) feature detectors are learnt. These include local oriented stroke detectors and detectors of digit parts such as loops. It was to be expected that denoising a more corrupted input requires detecting bigger, less local structures: the denoising auto-encoder must rely on longer range statistical dependencies and pool evidence from a larger subset of pixels. Interestingly, filters that start from the same initial random weight vector often look like they "grow" from random, to local blob detector, to slightly bigger structure detectors such as a stroke detector, as we use increased noise levels. By "grow" we mean that the slightly larger structure learnt at a higher noise level often appears related to the smaller structure obtained at lower noise levels, in that they share about the same position and orientation.

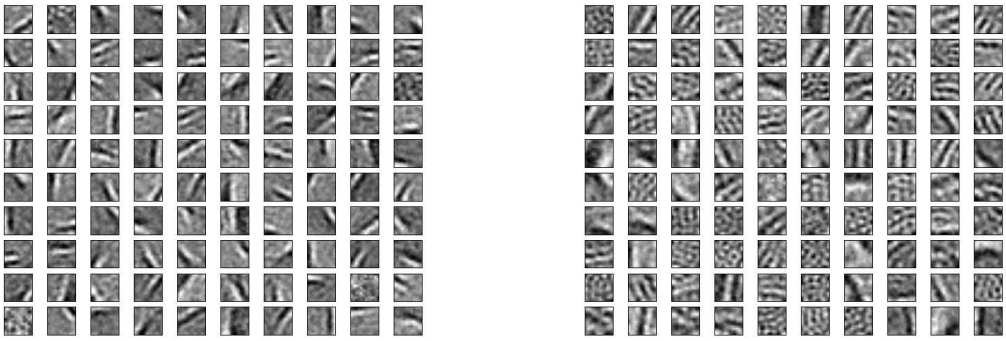


Figure 7: Filters obtained on natural image patches by denoising autoencoders using other noise types. *Left:* with 10% salt-and-pepper noise, we obtain oriented Gabor-like filters. They appear slightly less localized than when using Gaussian noise (contrast with Figure 6 right). *Right:* with 55% zero-masking noise we obtain filters that look like oriented gratings. For the three considered noise types, denoising training appears to learn filters that capture meaningful natural image statistics structure.

6. Experiments on Stacked Denoising Autoencoders

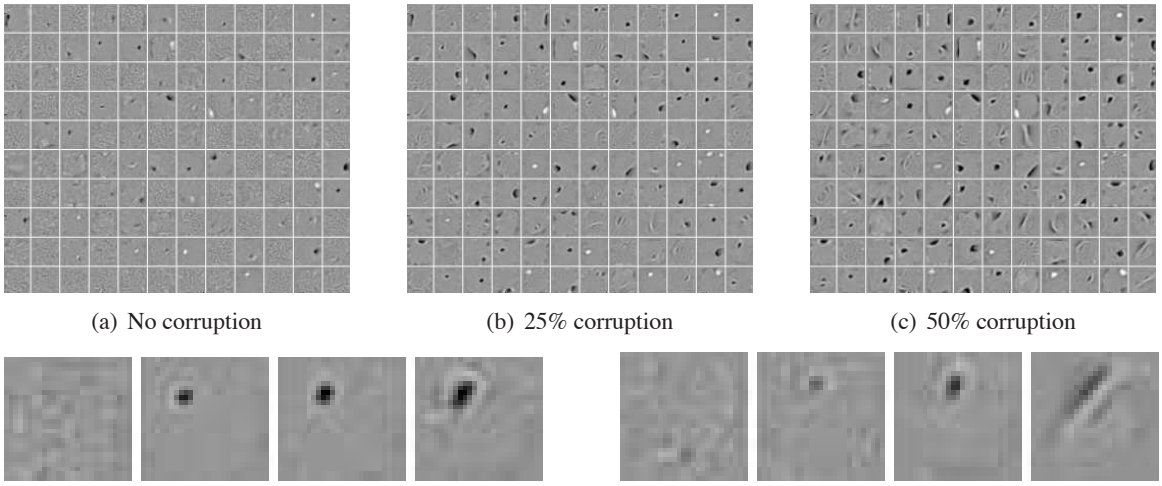
In this section, we evaluate denoising autoencoders as a pretraining strategy for building deep networks, using the stacking procedure that we described in Section 3.5. We shall mainly compare the classification performance of networks pretrained by stacking denoising autoencoders (SDAE), versus stacking regular autoencoders (SAE), versus stacking restricted Boltzmann machines (DBN), on a benchmark of classification problems.

6.1 Considered Classification Problems and Experimental Methodology

We considered 10 classification problems, the details of which are listed in Table 1. They consist of:

- The standard MNIST digit classification problem with 60000 training examples.
- The eight benchmark image classification problems used in Larochelle et al. (2007) which include more challenging variations of the MNIST digit classification problem (all with 10000 training examples), as well as three artificial 28×28 binary image classification tasks.[11] These problems were designed to be particularly challenging to current generic learning algorithms (Larochelle et al., 2007). They are illustrated in Figure 9.
- A variation of the *tzanetakis* audio genre classification data set (Bergstra, 2006) which contains 10000 three-second audio clips, equally distributed among 10 musical genres: blues, classical, country, disco, hiphop, pop, jazz, metal, reggae and rock. Each example in the set

11. The data sets for this benchmark are available at `http://www.iro.umontreal.ca/~lisa/icml2007`.



(a) No corruption (b) 25% corruption (c) 50% corruption

(d) Neuron A (0%, 10%, 20%, 50% corruption) (e) Neuron B (0%, 10%, 20%, 50% corruption)

Figure 8: Filters learnt by denoising autoencoder on MNIST digits, using zero-masking noise. *(a-c)* show some of the filters learnt by denoising autoencoders trained with various corruption levels ν. Filters at the same position in the three images are related only by the fact that the autoencoders were started from the same random initialization point in parameter space. *(d)* and *(e)* zoom in on the filters obtained for two of the neurons. As can be seen, with no noise, many filters remain similarly uninteresting (undistinctive almost uniform random grey patches). As we increase the noise level, denoising training forces the filters to differentiate more, and capture more distinctive features. Higher noise levels tend to induce less local filters, as expected. One can distinguish different kinds of filters, from local blob detectors, to stroke detectors, and character parts detectors at the higher noise levels.

was represented by 592 *Mel Phon Coefficient* (MPC) features. These are a simplified formulation of the *Mel-frequency cepstral coefficients* (MFCCs) that were shown to yield better classification performance (Bergstra, 2006).

All problems except *tzanetakis* had their data split into training set, validation set and test set. We kept the same standard splits that were used in Larochelle et al. (2007). The training set is used for both pretraining and fine tuning of the models. Classification performance on the validation set is used for choosing the best configuration of hyperparameters (model selection). The corresponding classification performance on the test set is then reported together with a 95% confidence interval.

For *tzanetakis* we used a slightly different procedure, since there was no predefined standard split and fewer examples. We used 10-fold cross validation, where each fold consisted of 8000 training examples, 1000 test and 1000 validation examples. For each fold, hyperparameters were chosen based on the performance on the validation set, and the retained model was used for computing the corresponding test error. We report the average test error and standard deviation across the 10 folds.

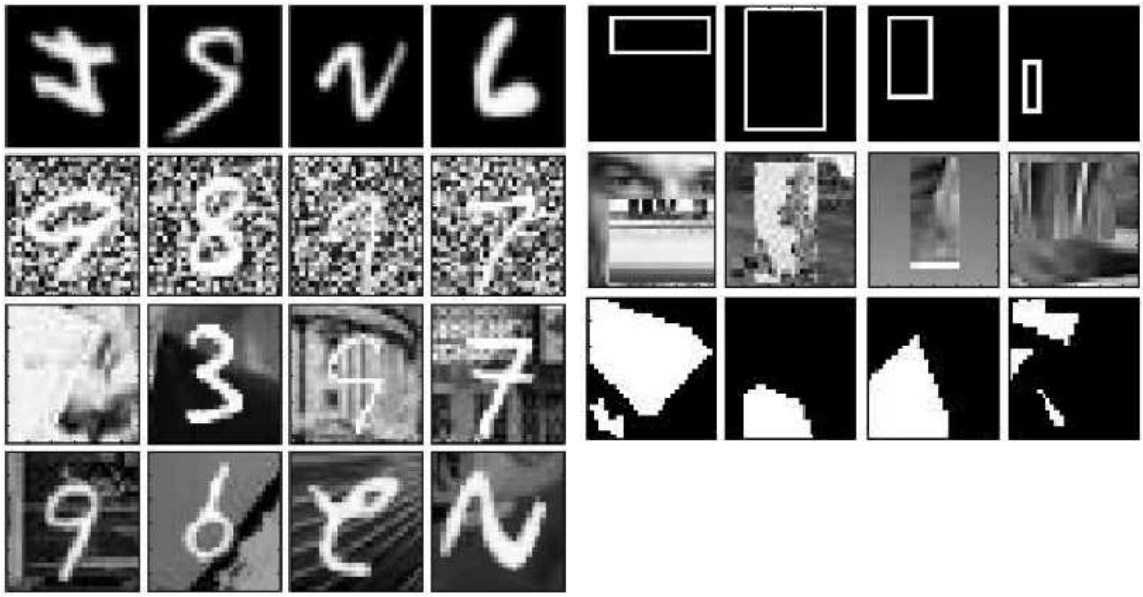
We were thus able to compare the classification performance of deep neural networks using different unsupervised initialization strategies for their parameters:

- MLP random: multilayer perceptron with usual random initialization;
- DBN (deep belief networks) corresponds to stacked RBM pretraining;
- SAE stacked autoencoder pretraining;
- SDAE stacked denoising autoencoder pretraining.

In all cases, the same supervised fine-tuning procedure was then used, that is, simple stochastic gradient descent with early stopping based on validation set performance.

Data Set	**Description**	**input**	m	**Train-Valid-Test**
MNIST	Standard MNIST digit classification problem.	784 gray-scale values scaled to [0,1]	10	50000-10000-10000
basic	Smaller subset of MNIST.		10	10000-2000-50000
rot	MNIST digits with added random rotation.		10	10000-2000-50000
bg-rand	MNIST digits with random noise background.		10	10000-2000-50000
bg-img	MNIST digits with random image background.		10	10000-2000-50000
bg-img-rot	MNIST digits with rotation and image background.		10	10000-2000-50000
rect	Discriminate between tall and wide rectangles (white on black).	784 values $\in\{0,1\}$	2	10000-2000-50000
rect-img	Discriminate between tall and wide rectangular image overlayed on a different background image.	784 values $\in [0,1]$	2	10000-2000-50000
convex	Discriminate between convex and concave shape.	784 values $\in\{0,1\}$	2	6000-2000-50000
tzanetakis	Classify genre of thirty second audio-clip.	592 MPC coefficients, standardized.	10	10-fold cross validation on 10000 training samples.

Table 1: Data sets. Characteristics of the 10 different problems considered. Except for *tzanetakis* whose input is made of 592 MPC features extracted from short audio sequences, all other problems are 28×28 gray-scale image classification tasks (i.e., input dimensionality is $28 \times 28 = 784$). See Larochelle et al. (2007) and Bergstra (2006) for further details on these data sets. The table gives, for each task, its shorthand name, a description of the problem, a description of input preprocessing, the number of classes (m), and the number of examples used for the training, validation and test sets respectively.



(a) *rot*, *bg-rand*, *bg-img*, *bg-img-rot*

(b) *rect*, *rect-img*, *convex*

Figure 9: Samples form the various image classification problems. (a): harder variations on the MNIST digit classification problems. (b): artificial binary classification problems.

On the 28×28 gray-scale image problems, SAE and SDAE used linear+sigmoid decoder layers and were trained using a cross-entropy loss, due to this being a natural choice for this kind of (near) binary images, as well as being functionally closer to RBM pretraining we wanted to compare against.

However for training the *first* layer on the *tzanetakis* problem, that is, for reconstructing *MPC coefficients*, a linear decoder and a squared reconstruction cost were deemed more appropriate (subsequent layers used sigmoid cross entropy as before). Similarly the first layer RBM used in pretraining a DBN on *tzanetakis* was defined with a Gaussian visible layer.

Table 2 lists the hyperparameters that had to be tuned by proper model selection (based on validation set performance). Note that to reduce the choice space, we restricted ourselves to the same number of hidden units, the same noise level, and the same learning rate for *all* hidden layers.

6.2 Empirical Comparison of Deep Network Training Strategies

Table 3 reports the classification performance obtained on all data sets using a 3 hidden layer neural network pretrained with the three different strategies: by stacking denoising autoencoders (SDAE-3), stacking restricted Boltzmann machines (DBN-3), and stacking regular autoencoders (SAE-3). For reference the table also contains the performance obtained by a single hidden-layer DBN-1 and by a Support Vector Machine with a RBF kernel (SVM_{rbf}).[12]

12. SVMs were trained using the libsvm implementation. Their hyperparameters (C and kernel width) were tuned semi-automatically (i.e., by human guided grid-search), searching for the best performer on the validation set.

hyperparameter	description	considered values
nHLay	number of hidden layers	{1,2,3}
nHUnit	number of units per hidden layer (same for all layers)	{1000,2000,3000}
lRate	fixed learning rate for unsupervised pretraining	$\{ 5 \times 10^{-6}, 5 \times 10^{-5}, 5 \times 10^{-4}, 5 \times 10^{-3}, 5 \times 10^{-2}, 10^{-1}\}$
lRateSup	fixed learning rate for supervised fine-tuning	{0.0005,0.005,0.05,0.1,0.15,0.2}
nEpoq	number of pretraining epochs (passages through the whole training set)	{5,10,50,100,125,150,200,300}
ν	corrupting noise level	fraction of corrupted inputs $(0, 0.10, 0.25, 0.40)$ or standard deviation for Gaussian noise $(0, 0.05, 0.10, 0.15, 0.30, 0.50)$

Table 2: List of hyperparameters for deep networks. These hyperparameters are common to all considered deep network training strategies, except for noise level ν which is specific to SDAE (for which we must also choose the kind of corruption). We list the typical values we considered in the majority of our experiments. Best performing configuration on the validation set was always searched for in a semi-automatic fashion, that is, running experiments in parallel on a large computation cluster, but with manual guidance to avoid wasting resources on unnecessary parts of the configuration space. Some experiments meant to study more closely the influence of specific hyperparameters occasionally used a finer search grid for them, as will be specified in the description of these experiments.

In these experiments, SDAE used a zero-masking corruption noise for all problems except for *tzanetakis*, for which a Gaussian noise was deemed more appropriate due to the nature of the input.

We see that SDAE-3 systematically outperforms the baseline SVM, as well as SAE-3 (except for *convex* for which the difference is not statistically significant). This shows clearly that denoising pretraining with a non-zero noise level is a better strategy than pretraining with regular autoencoders. For all but one problem, SDAE-3 is either the best performing algorithm or has its confidence interval overlap with that of the winning algorithm (i.e., difference cannot be considered statistically significant). In most cases, stacking 3 layers of denoising autoencoder seems to be on par or better than stacking 3 layers of RBMs in DBN-3.

In the following subsections, we will be conducting further detailed experiments to shed light on particular aspects of the denoising autoencoder pretraining strategy.

6.3 Influence of Number of Layers, Hidden Units per Layer, and Noise Level

Next we wanted to study more closely the influence of important architectural hyperparameters, namely the number of layers, the number of hidden units per layer, and the noise level. For this finer

Data Set	SVM_{rbf}	DBN-1	SAE-3	DBN-3	SDAE-3 (ν)
MNIST	**1.40**±0.23	**1.21**±0.21	**1.40**±0.23	**1.24**±0.22	**1.28**±0.22 (25%)
basic	**3.03**±0.15	3.94±0.17	3.46±0.16	3.11±0.15	**2.84**±0.15 (10%)
rot	11.11±0.28	14.69±0.31	10.30±0.27	10.30±0.27	**9.53**±0.26 (25%)
bg-rand	14.58±0.31	9.80±0.26	11.28±0.28	**6.73**±0.22	10.30±0.27 (40%)
bg-img	22.61±0.37	**16.15**±0.32	23.00±0.37	**16.31**±0.32	**16.68**±0.33 (25%)
bg-img-rot	55.18±0.44	52.21±0.44	51.93±0.44	47.39±0.44	**43.76**±0.43 (25%)
rect	**2.15**±0.13	4.71±0.19	2.41±0.13	2.60±0.14	**1.99**±0.12 (10%)
rect-img	24.04±0.37	23.69±0.37	24.05±0.37	22.50±0.37	**21.59**±0.36 (25%)
convex	19.13±0.34	19.92±0.35	**18.41**±0.34	**18.63**±0.34	**19.06**±0.34 (10%)
tzanetakis	**14.41**±2.18	18.07±1.31	**16.15**±1.95	18.38±1.64	**16.02**±1.04(0.05)

Table 3: Comparison of stacked denoising autoencoders (SDAE-3) with other models. Test error rate on all considered classification problems is reported together with a 95% confidence interval. Best performer is in bold, as well as those for which confidence intervals overlap. SDAE-3 appears to achieve performance superior or equivalent to the best other model on all problems except *bg-rand*. For SDAE-3, we also indicate the fraction ν of corrupted input components, or in case of *tzanetakis*, the standard deviation of the Gaussian noise, as chosen by proper model selection. Note that SAE-3 is equivalent to SDAE-3 with $\nu = 0\%$.

grained series of experiments, we chose to concentrate on the hardest of the considered problems, that is, the one with the most factors of variation: *bg-img-rot*.

We first examine how the proposed network training strategy behaves as we increase the capacity of the model both in breadth (number of neurons per layer) and in depth (number of hidden layers). Figure 10 shows the evolution of the performance as we increase the number of hidden layers from 1 to 3, for three different network training strategies: without any pretraining (standard MLP), with ordinary autoencoder pretraining (SAE) and with denoising autoencoder pretraining (SDAE). We clearly see a strict ordering: denoising pretraining being better than autoencoder pretraining being better than no pretraining. The advantage appears to increase with the number of layers (note that without pretraining it seems impossible to successfully train a 3 hidden layer network) and with the number of hidden units. This general behavior is a typical illustration of what is gained by pretraining deep networks with a good unsupervised criterion, and appears to be common to several pretraining strategies. We refer the reader to Erhan et al. (2010) for an empirical study and discussion regarding possible explanations for the phenomenon, centered on the observation of *regularization* effects (we exploit the hypothesis that features of X that help to capture $P(X)$ also help to capture $P(Y|X)$) and *optimization* effects (unsupervised pre-training initializes parameters near a better *local minimum* of *generalization* error).

Notice that in tuning the hyperparameters for all classification performances so far reported, we considered only a coarse choice of noise levels ν (namely 0%, 10%, 25%, or 40% of zero-masking corruption for the image classification problems). Clearly it was not necessary to pick the noise level very precisely to obtain good performances. In Figure 11 we examine in more details the influence of the level of corruption ν using a more fine-grained grid for problem *bg-img-rot*. We

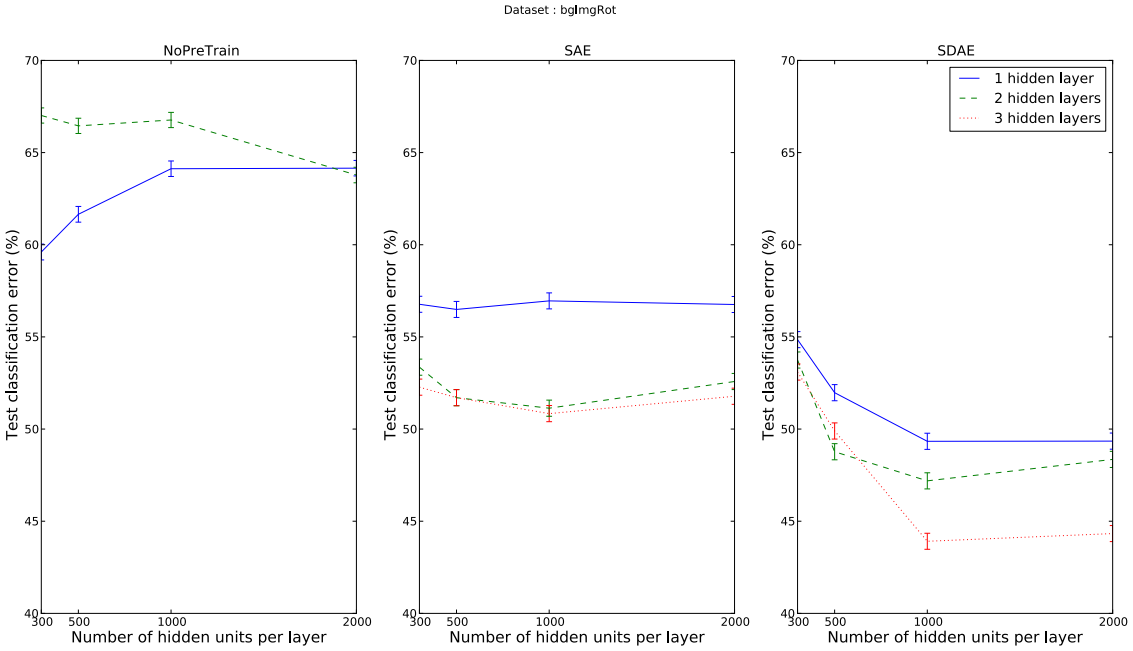


Figure 10: Classification performance on *bg-img-rot* for standard MLP with random initialization (NoPreTrain, left), SAE (middle), and SDAE (right), as we increase the number of hidden layers and the number of neurons per layer. Error bars show 95% confidence intervals. Note that without pretraining the curve for 3 layers is outside the graphic, the classification error being above 89%.

notice that SDAE appears to perform better than SAE (0 noise) for a rather wide range of noise levels, regardless of the number of hidden layers.

The following section reports an experiment that was conducted on three other data sets. The experiment had a different goal and thus used a coarser ν grid, but the resulting curves (see Figure 12) appear to follow a similar pattern to the one seen here (Figure 11).

6.4 Denoising Pretraining vs. Training with Noisy Input

We discussed in Section 4.2 the important distinction between denoising pretraining as it is done in SDAE and simply training with noisy inputs. SDAE uses a denoising criterion to learn good initial feature extractors at each layer that will be used as initialization for a *noiseless* supervised training. This is very different from training with noisy inputs, which amounts to training with a (virtually) expanded data set. This latter approach can in principle be applied to any classifier, such as an SVM[13] or a SAE. Note that in the case of the SAE, since there are two phases (pretraining and fine-tuning), it is possible to use noisy inputs for only the pretraining or for both the pretraining

13. For SAE, input examples can cheaply be corrupted on the fly, but this is not an option with standard SVM algorithms. So for SVM training we first augmented the training set by generating 9 extra variations of each original training example thus yielding a training set 10 times bigger than the original. Alternatively, we could instead have used the

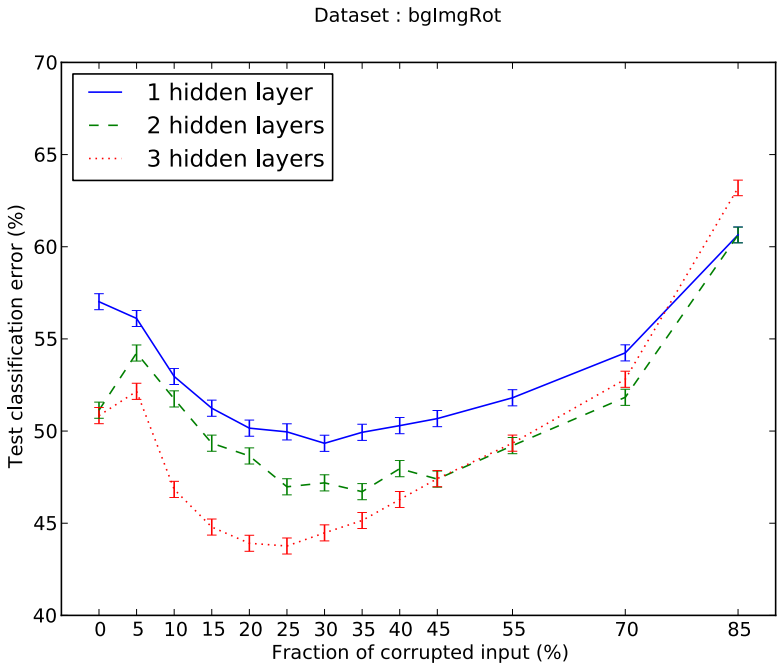


Figure 11: Sensitivity to the level of corruption. The curves report the test error rate for SDAE trained on problem *bg-img-rot* as a function of the fraction ν of corrupted input components (using zero masking noise). Error bars show 95% confidence interval. Note that 0% corruption corresponds to a SAE (regular autoencoder).

and fine-tuning phase. We experimentally compare these different approaches on three data sets in Figure 12. We see that denoising pretraining with SDAE, for a large range of noise levels, yields significantly improved performance, whereas training with noisy inputs sometimes degrades the performance, and sometimes improves it slightly but is clearly less beneficial than SDAE.

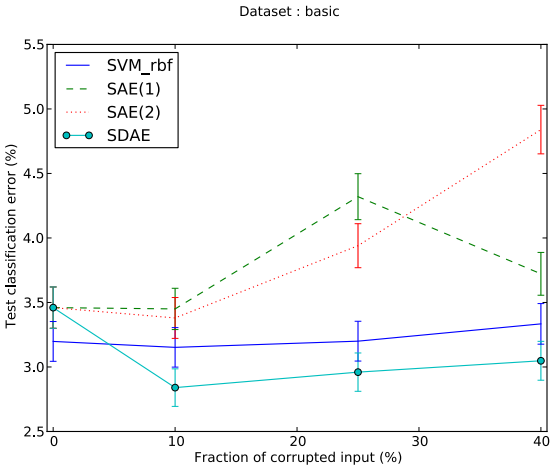
6.5 Variations on the Denoising Autoencoder: Alternate Corruption Types and Emphasizing

In the next series of experiments, we wanted to evaluate quantitatively the effect of using the various corrupting noises described in Section 3.3 as well as the *emphasized denoising autoencoder* variant described in Section 3.4.

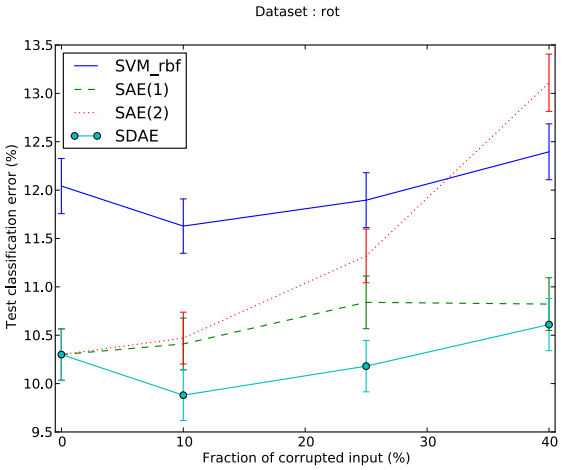
Extensive experiments were carried out on the same 3 problems we used in the previous section. Besides zero-masking noise (MN) we trained 3 hidden layer SDAE using salt-and-pepper noise (SP) and additive Gaussian noise (GS). For MN and SP, we also tried the emphasized variant.[14] For each considered variant, hyperparameters were selected as usual to yield the best performance on the

virtual SV technique (Schölkopf et al., 1996), which may or may not have yielded better performance, but since our main focus here is comparing noisy SAE with SDAE, SVM only serves as a simple baseline.

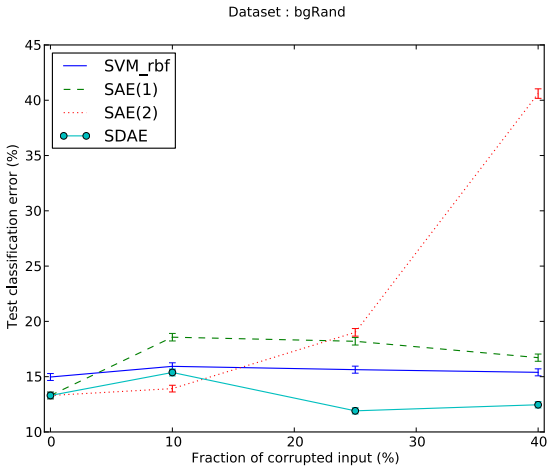
14. As already mentioned previously, since Gaussian noise corrupts every dimension, emphasized denoising does not make sense for this type of corruption.



(a) *basic*



(b) *rot*



(c) *bg-rand*

Figure 12: SDAE vs. training with noisy input. The test error of a **SDAE** with 3 hidden layers is compared to other algorithms trained with noisy inputs: a SVM with RBF kernel (**SVM$_{\text{rbf}}$**), a 3-hidden-layers SAE where noisy inputs were used for pretraining only (**SAE(1)**) and one where noisy inputs were used both for pretraining and supervised fine-tuning (**SAE(2)**). Hidden layers have 1000 neurons each. Zero-masking noise was used. Note that at 0% noise, the three stacked models correspond to an ordinary SAE. Error bars show 95% confidence interval. Denoising pretraining with SDAE appears to always yield significantly better performance, unlike training with noisy inputs.

Model	*basic*	*rot*	*bg-rand*
SVM$_{rbf}$	3.03±0.15	11.11±0.28	14.58±0.31
SAE-3	3.46±0.16	10.30±0.27	11.28±0.28
DBN-3	3.11±0.15	10.30±0.27	**6.73**±0.22
SDAE-3$_{MN}(\nu)$	2.84±0.15(10%)	9.53±0.26(25%)	10.30±0.27(40%)
SDAE-3$_{MN}(\nu)$ **+ emph**	**2.76**±0.14(25%)	10.36±0.27(25%)	9.69±0.26(40%)
SDAE-3$_{SP}(\nu)$	**2.66**±0.14(25%)	9.33±0.25(25%)	10.03±0.26(25%)
SDAE-3$_{SP}(\nu)$ **+ emph**	**2.48**±0.14(25%)	**8.76**±0.29(25%)	8.52±0.24(10%)
SDAE-3$_{GS}(\nu)$	**2.61**±0.14(0.1)	**8.86**±0.28(0.3)	11.73±0.28(0.1)

Table 4: Variations on 3-hidden-layer stacked denoising autoencoders (SDAE-3): alternative noise types and effect of emphasis. Considered noise types are masking noise (MN), salt-and-pepper (SP) and Gaussian noise (GS). Emphasized version considered double emphasis and full emphasis (see main text for detailed explanation). For easy comparison, the table also reproduces previously shown results for SVM_{rbf}, SAE-3, and DBN-3. Test error rate is reported together with a 95% confidence interval. Best performer is in bold, as well as those for which confidence intervals overlap. Corruption level ν (fraction of corrupted input components or Gaussian standard deviation) that was retained by model selection on the validation set is specified in parenthesis. SDAE-3$_{SP}$ with emphasis on reconstruction of corrupted dimension appears to be the best SDAE variant for these data sets, significantly improving performance on *rot* and *bg-rand*.

validation set. These included the number of units per layer (same for all layers), the corruption level ν (fraction of corrupted dimensions for MN and SP, or standard deviation for GS), with the usual considered values (listed previously in Table 2). For the emphasized version, a further hyperparameter was the degree of emphasis. We considered both *double emphasis*, where the weight on the reconstruction of the corrupted components is twice that on the uncorrupted components ($\alpha = 1,\ \beta = 0.5$), and *full emphasis* where all the weight is on reconstructing the corrupted components and none on the uncorrupted dimensions ($\alpha = 1,\ \beta = 0$). Table 4 reports the corresponding classification performance on the held-out test set. For the three considered data sets, an emphasized SDAE with salt-and-pepper noise appears to be the winning SDAE variant. It thus appears that a judicious choice of noise type and added emphasis may often buy us a better performance. However we had hoped, with these variants, to catch up with the performance of DBN-3 on the *bg-rand* problem,[15] but DBN-3 still performs significantly better than the best SDAE variant on this particular problem.

6.6 Are Features Learnt in an Unsupervised Fashion by SDAE Useful for SVMs?

In the following series of experiments, we wanted to verify whether the higher level representations extracted using SDAE could improve the performance of learning algorithms other than a neural network, such as SVMs.

15. As discussed in Larochelle et al. (2007), *bg-rand* is particularly favorable to RBMs because the pixel-wise independent noise perfectly matches what an RBM expects and will naturally not be represented in the hidden units.

To this end, we fed the representations learnt by the purely unsupervised phase of SDAE, at increasing higher levels (first, second and third hidden layer) to both a linear SVM and a Kernel SVM (using a RBF kernel). The hyperparameters of the SVM and its kernel were tuned on the validation set as usual. For computational reasons, we did not re-tune SDAE hyperparameters. Instead, we identified the best performing SDAE-pretrained neural networks with 1000 units per layer, based on their validation performance after fine-tuning from previous experiments, but used their saved weights prior to fine-tuning (i.e., after unsupervised denoising training only).

Results for all considered data sets are reported in Table 5, and Figure 13 highlights performance curves for two of them. Clearly, SVM performance can benefit significantly from using the higher level representation learnt by SDAE.[16] On all problems we see improved performance compared to using the original input (SVM_0). More interestingly, on most problems, SVM performance improves steadily as we use ever higher level representations. While it is not too surprising that linear SVMs can benefit from having the original input processed non-linearly, it is noteworthy that RBF kernel SVMs, which are high-capacity non-linear classifiers, also seem to benefit greatly from the non-linear mapping learned by SDAE.

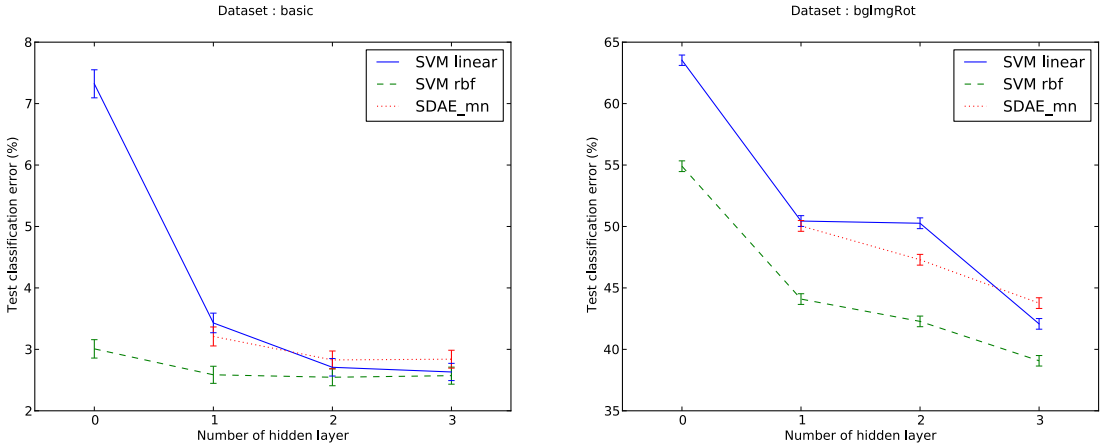


Figure 13: SVM on representations learnt by SDAE. The curves show evolution, on two data sets, of the test performance of linear and RBF kernel SVMs as we train them on higher level representations learnt in the unsupervised phase of SDAE. Performance of SDAE after supervised fine-tuning is also shown as $SDAE_{mn}$ (mn stands for masking noise). Hidden layer 0 corresponds to original input representation.

7. Generating Samples from Stacked Denoising Autoencoder Networks

Besides the classification performance comparisons and qualitative visual inspection of learnt filters, it is also customary in studies of deep generative models such as DBNs, to show samples generated

16. To verify that the learnt representation was responsible for the improved performance, rather than a random non-linear transformation, we also trained SVMs on the representation of the same neural network architecture but using randomly initialized weights: the performance degraded as we used the higher level representations.

Data Set	SVM kernel	SVM_0	SVM_1	SVM_2	SVM_3
MNIST	linear	5.33±0.44	1.49 ±0.24	**1.24**±0.22	**1.2**±0.21
	rbf	1.40±0.23	**1.04**±0.20	**0.94**±0.19	**0.95**±0.19
basic	linear	7.32±0.23	3.43±0.16	**2.71**±0.14	**2.63**±0.14
	rbf	3.03±0.15	**2.59**±0.14	**2.55**±0.14	**2.57**±0.14
rot	linear	43.47±0.43	21.74±0.36	15.15±0.31	10.00±0.26
	rbf	11.11±0.28	**8.45**±0.24	**8.27**±0.24	**8.64**±0.25
bg-rand	linear	24.14±0.38	13.58±0.30	13.00±0.29	11.32±0.28
	rbf	14.58±0.31	11.00±0.27	**10.08**±0.26	**10.16**±0.26
bg-img	linear	25.08±0.38	16.72±0.33	20.73±0.36	**14.55**±0.31
	rbf	22.61±0.37	15.91±0.32	16.36±0.32	**14.06**±0.30
bg-img-rot	linear	63.53±0.42	50.44±0.44	50.26±0.44	42.07±0.43
	rbf	55.18±0.44	44.09±0.44	42.28±0.43	**39.07**±0.43
rect	linear	29.04±0.40	6.43±0.22	2.31±0.13	1.80±0.12
	rbf	2.15±0.13	2.19±0.13	1.46±0.11	**1.22**±0.10
rect-img	linear	49.64±0.44	23.12±0.37	23.01±0.37	**21.43**±0.36
	rbf	24.04±0.37	22.27±0.36	21.56±0.36	**20.98**±0.36
convex	linear	45.75±0.44	24.10±0.37	18.40±0.34	**18.06**±0.34
	rbf	19.13±0.34	18.09±0.34	**17.39**±0.33	**17.53**±0.33
tzanetakis	linear	20.72±2.51	12.51±2.05	7.95±1.68	**5.04**±1.36
	rbf	14.41±2.18	7.54±1.64	**5.20**±1.38	**4.13**±1.23

Table 5: SVM performance on higher level representations learnt by SDAE. Performance of both linear SVM, and SVM with RBF kernel is reported, as they are trained on either original input (SVM_0), or on the representation learnt by a SDAE at the level of its first (SVM_1), second (SVM_2), or third (SVM_3) hidden layer. The representations used for the SVMs were those obtained prior to fine-tuning. Test error rate on all considered classification problems is reported together with a 95% confidence interval. Best performer is in bold, as well as those for which confidence intervals overlap. Clearly both linear and kernel SVM performance benefit from using the higher level representations learnt by SDAE. For most problems the performance increases steadily as we use representations from ever higher levels of the architecture.

from the trained models. This can yield another qualitative visual assessment of whether they were able to capture the input distribution well.

7.1 Top-Down Generation of a Visible Sample Given a Top-Layer Representation

Given a top-layer representation, a deep belief network (Hinton et al., 2006) is a directed graphical model, and it is easy to do a top down sampling pass, that is, sampling each layer conditioned on the layer above, to eventually produce a sample in the bottom layer that can be displayed. More precisely, in sigmoid deep belief networks (DBN), the representation at a lower layer X given the

layer above Y is distributed according to a product of independent Bernoullis whose mean is a deterministic function of Y, that is, $X|Y \sim \mathcal{B}(g_{\theta'}(Y))$, where $g_{\theta'}$ has the exact same form as that given in Equation 3 for the *decoder* part of an autoencoder. From a trained SAE or SDAE[17] it is thus possible to generate samples at one layer from the representation of the layer above in the exact same way as in a DBN.

7.2 Bottom-Up Inferring of the Top-Layer Representation Corresponding to a Given Input Pattern

In SAE/SDAE, *given an input representation at the bottom layer*, the corresponding representation in the top layer is computed in a deterministic *bottom-up* pass using encoding functions f_θ. The same procedure is used in DBNs and, in the graphical model perspective, can be viewed as an *approximate inference* of a factorial Bernoulli top-layer distribution given the low level input. This top-layer representation is to be understood as the parameters (the mean) of a factorial Bernoulli distribution for the actual binary units.

7.3 Generating Samples with SAE, SDAE, and DBN Using the Same Procedure

The deep belief network of Hinton et al. (2006) is a fully specified generative model. In particular the joint distribution of its top two layers is defined by an RBM model,[18], that is, an *undirected* graphical model from which one can efficiently sample using alternating Gibbs sampling (Hinton et al., 2006). So to sample from a DBN model, one would first sample from the top-layer RBM using alternating Gibbs sampling. Then, given the thus obtained top-layer representation, perform the single top down sampling pass previously described to produce a visible pattern at the bottom layer.

By contrast, SAE/SDAE training does not attempt to model the distribution of the top-layer representation. So even though—given a top-layer representation—we can use the exact same top down sampling procedure to generate input patterns from a SAE/SDAE as for a DBN, SAE/SDAE cannot by themselves alone be treated as fully specified generative models. They lack a model of the marginal distribution of their top layer.

We can easily fix this by modeling that top-layer distribution non-parametrically by the simple memory-based *empirical distribution* of the encoded top-layer representations of the n training set patterns. A visible sample can then be generated by simply taking the top-layer encoded representation of a randomly picked training set input, and carrying out the top-down sampling procedure explained previously, as illustrated in Figure 14. This same technique can also be used as an alternative sample-generation procedure for DBNs built by stacking RBMs.

If we keep the same fixed input pattern, and hence the same corresponding higher level representation, and perform several top-down samplings, we can thus observe what kind of pattern variations the deep multilayer part of a deep network has learnt to model (or abstract away in extracting the top-layer representation). Figure 15 shows the resulting variability in the regenerated patterns, for models pretrained respectively as SAE, SDAE[19] and DBN on MNIST without any su-

17. SAE and SDAE train such $g_{\theta'}$ to perform reconstruction, that is, predicting the mean value of a layer given the representation in the layer immediately above it.
18. This RBM was trained, using the training set, to model the representations obtained at the layer just below the top one, produced by the bottom-up pass we just described.
19. Both were pretrained with salt-and-pepper noise.

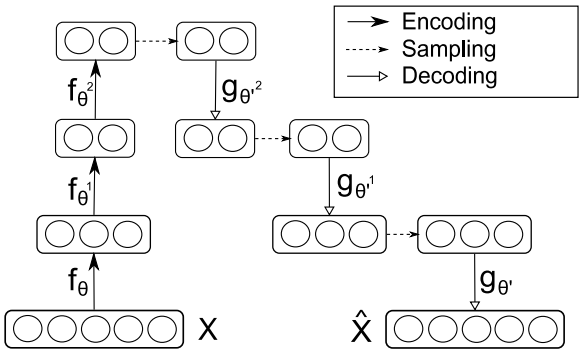


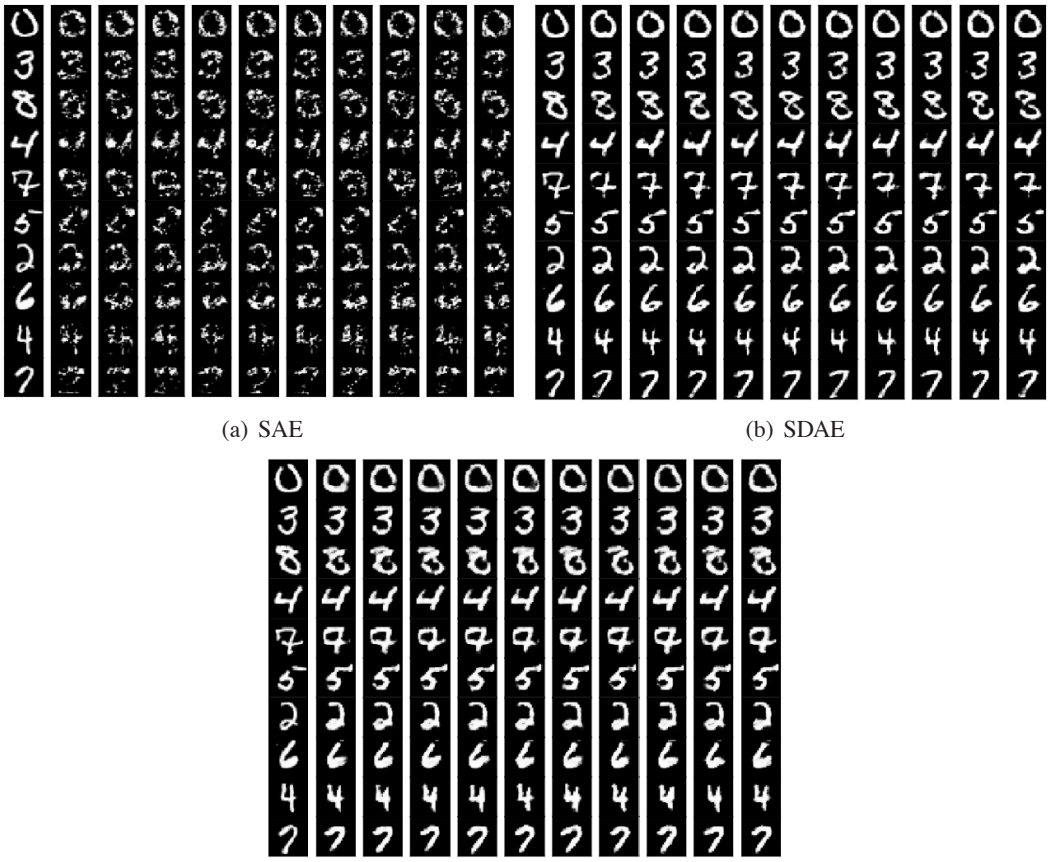
Figure 14: Non-parametric sampling procedure for pretrained networks. A randomly picked input form the original data set is provided as input. Its top level representation is obtained by deterministic bottom-up encoding using functions $f_{\theta^{(k)}}$. A visible pattern is generated given this high level representation, by alternating Bernoulli sampling and deterministic decoding, that is, by successively sampling from $\mathcal{B}(g_{\theta'^{(k)}}(\text{previouslayer}))$. This same procedure can be applied with SAE, SDAE and DBN. It allows to see the quality and variability of patterns one obtains given a high-level representation.

pervised fine-tuning. It appears that SDAE and DBN are able to resynthesize a variety of similarly good quality digits, whereas the SAE trained model regenerates patterns with much visible degradation in quality. This is further evidence of the qualitative difference resulting from optimizing a denoising criterion instead of mere reconstruction criterion. Note how SDAE puts back the missing hole in the loop of the regenerated 6, and sometimes straightens up the upper stroke of the 7, suggesting that it did indeed capture interesting specific characteristics. It appears that, when using this same sample generation procedure, SDAE and DBN yield a similar degree of variability in the regenerated patterns (with DBN patterns looking slightly fatter and SDAE patterns slightly thinner). Neither DBN nor SDAE guarantee that class boundaries will not be crossed,[20] for example DBN closes a loop in a 7 making it look closer to a 9, whereas SDAE sometimes breaks open the loop of an 8 making it look like a 3. But in all cases, and contrary to SAE, the regenerated patterns look like they could be samples from the same unknown input distribution that yielded the training set.

8. Conclusion and Future Work

The present work was inspired by recent successful approaches to training deep networks, specifically by their use of a local unsupervised criterion, and led by the question of what that criterion should be. At the same time we were motivated by a desire to bridge a remaining performance gap between deep belief networks and the stacking of ordinary autoencoders (Bengio et al., 2007; Larochelle et al., 2009a). This led us to address a theoretical shortcoming of traditional autoencoders—namely their inability in principle to learn useful over-complete representations—in a simple yet original way: by changing the objective from one involving mere reconstruction to the more challenging task of *denoising*. The resulting Stacked Denoising Autoencoder algorithm

20. The reader should however keep in mind that this results from unsupervised training only.



(a) SAE (b) SDAE

(c) DBN

Figure 15: Variability of the samples generated with 3-hidden-layer SAE, SDAE and DBN pre-trained models. Each sub-figure is to be read row-wise: the leftmost pattern in each row is a training set pattern. Following the sample generation depicted in Figure 14, it was provided as input to the network and its top-layer representation was computed by deterministic bottom up encoding. Patterns to its right were then generated independently given that top level representation. Clearly, SDAE trained networks, like DBNs, are able to regenerate high quality samples from their high level representation, contrary to SAE. SDAE and DBNs also appear to give rise to a similar level of variability in the bottom-up generated patterns (DBN patterns tending to be somewhat fatter). Note how SDAE puts back the missing hole in the loop of the regenerated 6, and sometimes straightens up the upper stroke of the last 7, suggesting that it did indeed capture meaningful specific characteristics. DBN and SDAE generated patterns can easily pass for samples from the unknown input distribution being modeled, unlike patterns generated by SAE.

for training deep networks, proved indeed able to bridge the performance gap with DBNs, yielding equivalent or better classification performance on all but one of the considered benchmark problems. As a deep network pretraining strategy, stacking of denoising autoencoders yielded in most cases a significant improvement in performance over the stacking of ordinary autoencoders. The representations thus extracted layer by layer, using a purely unsupervised local denoising criterion, appear to make subsequent classification tasks much easier. This is further evidenced by the fact that state-of-the-art shallow classifiers such as kernel SVMs also appear able to greatly benefit from it. Close examination of the feature extractors learnt by denoising autoencoders showed that they were able to zero in on useful structure in the data (such as Gabor-like edge detectors on natural image patches) that regular autoencoders seemed unable to learn.

The algorithm we developed is a straightforward, easy to implement, variation on the well-understood ordinary autoencoders. All that remains to be chosen is the kind and level of corrupting noise. It is likely that a careful choice, possibly guided by prior domain knowledge, may further boost application-specific performance. Nevertheless our experiments showed that high performance can already be achieved using very simple and generic noise types and with little tuning of the noise level. In addition, we were able to show that, contrary to what it may seem on the surface based on popular myths, the denoising training we advocate is *not* equivalent to using a mere weight decay regularization, nor is it the same as direct supervised training with corrupted (jittered) examples.

Beyond the specificities and practical usefulness of the simple algorithm we developed, **our results clearly establish the value of using a *denoising criterion* as an unsupervised objective to guide the learning of useful higher level representations.** This is in our view the most important contribution of our work, as it offers an interesting alternative to more usual (and often intractable) likelihood derived criteria. Indeed, denoising performance can easily be measured and directly optimized. The use of a denoising criterion is very different from the contrastive divergence training of RBMs or the direct enforcing of sparsity in autoencoders. We hope that our very encouraging results will inspire further research in this direction, both theoretical (to better understand the relationship between denoising and representation learning), and practical (to develop better learning algorithms based on this understanding).

There are certainly better ways to use denoising-based training signals in the learning of a deep network than the simple local approach we explored here. In particular, while stacking denoising autoencoders allows us to build a deep network, the denoising autoencoders we used here were shallow. It would thus be interesting to investigate deep denoising autoencoders with several hidden layers, and their ability to form useful representations. The choice and role of the corruption process also deserves further inquiry. If more involved corruption processes than those explored here prove beneficial, it would be most useful if they could be parameterized and learnt directly from the data, rather than having to be hand-engineered based on prior-knowledge.

Acknowledgments

This research was supported by funding from NSERC, MITACS, FQRNT, CIFAR, and the Canada Research Chairs, and partly carried out on computation resources made available by RQCHP.

References

G. An. The effects of adding noise during backpropagation training on a generalization performance. *Neural Computation*, 8(3):643–674, 1996.

H. Baird. Document image defect models. In *IAPR Workshop on Syntactic and Structural Pattern Recognition*, pages 38–46, Murray Hill, NJ., 1990.

P. Baldi and K. Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2:53–58, 1989.

A. Bell and T.J. Sejnowski. The independent components of natural scenes are edge filters. *Vision Research*, 37:3327–3338, 1997.

A.J. Bell and T.J. Sejnowski. An information maximisation approach to blind separation and blind deconvolution. *Neural Computation*, 7(6):1129–1159, 1995.

Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1): 1–127, 2009. Also published as a book. Now Publishers, 2009.

Y. Bengio and O. Delalleau. Justifying and generalizing contrastive divergence. *Neural Computation*, 21(6):1601–1621, June 2009.

Y. Bengio and Y. LeCun. Scaling learning algorithms towards AI. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large Scale Kernel Machines*. MIT Press, 2007.

Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In Bernhard Schölkopf, John Platt, and Thomas Hoffman, editors, *Advances in Neural Information Processing Systems 19 (NIPS'06)*, pages 153–160. MIT Press, 2007.

J. Bergstra. Algorithms for classifying recorded music by genre. Master's thesis, Université de Montreal, 2006.

J. Besag. Statistical analysis of non-lattice data. *The Statistician*, 24(3):179–195, 1975.

C.M. Bishop. Training with noise is equivalent to Tikhonov regularization. *Neural Computation*, 7 (1):108–116, 1995.

H. Bourlard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59:291–294, 1988.

O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, 2006.

Y. Cho and L. Saul. Kernel methods for deep learning. In Y. Bengio, D. Schuurmans, C. Williams, J. Lafferty, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22 (NIPS'09)*, pages 342–350. NIPS Foundation, 2010.

D. Erhan, Y. Bengio, A. Courville, P.A. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11:625–660, February 2010.

P. Gallinari, Y. LeCun, S. Thiria, and F. Fogelman-Soulie. Memoires associatives distribuees. In *Proceedings of COGNITIVA 87*, Paris, La Villette, 1987.

Y. Grandvalet, S. Canu, and S. Boucheron. Noise injection: Theoretical prospects. *Neural Computation*, 9(5):1093–1108, 1997.

J. Håstad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the 18th annual ACM Symposium on Theory of Computing*, pages 6–20, Berkeley, California, 1986. ACM Press.

J. Håstad and M. Goldmann. On the power of small-depth threshold circuits. *Computational Complexity*, 1:113–129, 1991.

D. Heckerman, D.M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie. Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research*, 1:49–75, 2000.

G.E. Hinton. Connectionist learning procedures. *Artificial Intelligence*, 40:185–234, 1989.

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

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

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

L. Holmstrm and P. Koistinen. Using additive noise in back-propagation training. *IEEE Transactions on Neural Networks*, 3(1):24–38, 1992.

J.J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences, USA*, 79, 1982.

D.H. Hubel and T.N. Wiesel. Receptive fields of single neurons in the cat's striate cortex. *Journal of Physiology*, 148:574–591, 1959.

V. Jain and S.H. Seung. Natural image denoising with convolutional networks. In Daphne Koller, Dale Schuurmans, Yoshua Bengio, and Leon Bottou, editors, *Advances in Neural Information Processing Systems 21 (NIPS'08)*, 2008.

N. Japkowicz, S.J. Hanson, and M.A. Gluck. Nonlinear autoassociation is not equivalent to PCA. *Neural Computation*, 12(3):531–545, 2000.

H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In Z. Ghahramani, editor, *Proceedings of the Twenty-fourth International Conference on Machine Learning (ICML'07)*, pages 473–480. ACM, 2007.

H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin. Exploring strategies for training deep neural networks. *Journal of Machine Learning Research*, 10:1–40, January 2009a.

H. Larochelle, D. Erhan, and P. Vincent. Deep learning using robust interdependent codes. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS 2009)*, pages 312–319, April 2009b.

Y. LeCun. *Modèles connexionistes de l'apprentissage*. PhD thesis, Université de Paris VI, 1987.

Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

H. Lee, C. Ekanadham, and A. Ng. Sparse deep belief net model for visual area V2. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20 (NIPS'07)*, pages 873–880, Cambridge, MA, 2008. MIT Press.

R. Linsker. An application of the principle of maximum information preservation to linear systems. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 1 (NIPS'88)*. Morgan Kaufmann, 1989.

J.L. McClelland, D.E. Rumelhart, and the PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 2. MIT Press, Cambridge, 1986.

B.A. Olshausen and D.J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607–609, 1996.

B.A. Olshausen and D.J. Field. Sparse coding with an overcomplete basis set: a strategy employed by V1? *Vision Research*, 37:3311–3325, December 1997.

T. Poggio and T. Vetter. Recognition and structure from one 2d model view: Observations on prototypes, object classes and symmetries. Technical Report A.I. Memo No. 1347, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1992.

M. Ranzato, C.S. Poultney, S. Chopra, and Y. LeCun. Efficient learning of sparse representations with an energy-based model. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19 (NIPS'06)*, pages 1137–1144. MIT Press, 2007.

M. Ranzato, Y. Boureau, and Y. LeCun. Sparse feature learning for deep belief networks. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20 (NIPS'07)*, pages 1185–1192, Cambridge, MA, 2008. MIT Press.

R. Scalettar and A. Zee. Emergence of grandmother memory in feed forward networks: Learning with noise and forgetfulness. In D. Waltz and J. A. Feldman, editors, *Connectionist Models and Their Implications: Readings from Cognitive Science*, pages 309–332. Ablex, Norwood, 1988.

B. Schölkopf, C.J.C. Burges, and V. Vapnik. Incorporating invariances in support vector learning machines. In C. von der Malsburg, W. von Seelen, J. C. Vorbrggen, and B. Sendhoff, editors, *Lecture Notes in Computer Science (Vol 112), Artificial Neural Netowrks ICANN'96*, pages 47–52. Springer, 1996.

S.H. Seung. Learning continuous attractors in recurrent networks. In M.I. Jordan, M.J. Kearns, and S.A. Solla, editors, *Advances in Neural Information Processing Systems 10 (NIPS'97)*, pages 654–660. MIT Press, 1998.

J. Sietsma and R. Dow. Creating artificial neural networks that generalize. *Neural Networks*, 4(1): 67–79, 1991.

P. Simard, B. Victorri, Y. LeCun, and J. Denker. Tangent prop - A formalism for specifying selected invariances in an adaptive network. In J.E. Moody S.J. Hanson and R.P. Lippmann, editors, *Advances in Neural Information Processing Systems 4 (NIPS'91)*, pages 895–903, San Mateo, CA, 1992. Morgan Kaufmann.

P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. In D.E. Rumelhart and J.L. McClelland, editors, *Parallel Distributed Processing*, volume 1, chapter 6, pages 194–281. MIT Press, Cambridge, 1986.

P.E. Utgoff and D.J. Stracuzzi. Many-layered learning. *Neural Computation*, 14:2497–2539, 2002.

P. Vincent, H. Larochelle, Y. Bengio, and P.A. Manzagol. Extracting and composing robust features with denoising autoencoders. In W.W. Cohen, A. McCallum, and S.T. Roweis, editors, *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, pages 1096–1103. ACM, 2008.

A. von Lehman, E.G. Paek, P.F. Liao, A. Marrakchi, and J.S. Patel. Factors influencing learning by back-propagation. In *IEEE International Conference on Neural Networks*, volume 1, pages 335–341, San Diego 1988, 1988. IEEE, New York.

J. Weston, F. Ratle, and R. Collobert. Deep learning via semi-supervised embedding. In William W. Cohen, Andrew McCallum, and Sam T. Roweis, editors, *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, pages 1168–1175, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390303.