Journal of Machine Learning Research 1 (2009) 1-40 Submitted 12/07; Revised 9/08; Published 1/09

Exploring Strategies for Training Deep Neural Networks

Hugo Larochelle LAROCHEH@IRO.UMONTREAL.CA
Yoshua Bengio BENGIOY@IRO.UMONTREAL.CA
Jérôme Louradour LOURADOJ@IRO.UMONTREAL.CA
Pascal Lamblin LAMBLINP@IRO.UMONTREAL.CA
Département d'informatique et de recherche opérationnelle
Université de Montréal
2920, chemin de la Tour
Montréal, Québec, Canada, H3T 1J8

Editor: Léon Bottou

Abstract

Deep multi-layer neural networks have many levels of non-linearities allowing them to compactly represent highly non-linear and highly-varying functions. However, until recently it was not clear how to train such deep networks, since gradient-based optimization starting from random initialization often appears to get stuck in poor solutions. Hinton et al. recently proposed a greedy layer-wise unsupervised learning procedure relying on the training algorithm of restricted Boltzmann machines (RBM) to initialize the parameters of a deep belief network (DBN), a generative model with many layers of hidden causal variables. This was followed by the proposal of another greedy layer-wise procedure, relying on the usage of autoassociator networks. In the context of the above optimization problem, we study these algorithms empirically to better understand their success. Our experiments confirm the hypothesis that the greedy layer-wise unsupervised training strategy helps the optimization by initializing weights in a region near a good local minimum, but also implicitly acts as a sort of regularization that brings better generalization and encourages internal distributed representations that are high-level abstractions of the input. We also present a series of experiments aimed at evaluating the link between the performance of deep neural networks and practical aspects of their topology, for example, demonstrating cases where the addition of more depth helps. Finally, we empirically explore simple variants of these training algorithms, such as the use of different RBM input unit distributions, a simple way of combining gradient estimators to improve performance, as well as on-line versions of those algorithms.

Keywords: artificial neural networks, deep belief networks, restricted Boltzmann machines, autoassociators, unsupervised learning

1. Introduction

Training deep multi-layered neural networks is known to be hard. The standard learning strategy—consisting of randomly initializing the weights of the network and applying gradient descent using backpropagation—is known empirically to find poor solutions for networks with 3 or more hidden layers. As this is a negative result, it has not been much reported in the machine learning literature. For that reason, artificial neural networks have been limited to one or two hidden layers.

However, complexity theory of circuits strongly suggests that deep architectures can be much more efficient (sometimes exponentially) than shallow architectures, in terms of computational el-

©2009 Hugo Larochelle, Yoshua Bengio, Jérôme Louradour and Pascal Lamblin.

ements and parameters required to represent some functions (Bengio and Le Cun, 2007; Bengio, 2007). Whereas it cannot be claimed that deep architectures are better than shallow ones on every problem (Salakhutdinov and Murray, 2008; Larochelle and Bengio, 2008), there has been evidence of a benefit when the task is complex enough, and there is enough data to capture that complexity (Larochelle et al., 2007). Hence finding better learning algorithms for such deep networks could be beneficial.

An approach that has been explored with some success in the past is based on constructively adding layers. Each layer in a multi-layer neural network can be seen as a representation of the input obtained through a learned transformation. What makes a good internal representation of the data? We believe that it should disentangle the factors of variation that inherently explain the structure of the distribution. When such a representation is going to be used for unsupervised learning, we would like it to preserve information about the input while being easier to model than the input itself. When a representation is going to be used in a supervised prediction or classification task, we would like it to be such that there exists a "simple" (i.e., somehow easy to learn) mapping from the representation to a good prediction. To constructively build such a representation, it has been proposed to use a *supervised* criterion at each stage (Fahlman and Lebiere, 1990; Lengellé and Denoeux, 1996). However, as we discuss here, the use of a supervised criterion at each stage may be too greedy and does not yield as good generalization as using an unsupervised criterion. Aspects of the input may be ignored in a representation tuned to be immediately useful (with a linear classifier) but these aspects might turn out to be important when more layers are available. Combining unsupervised (e.g., learning about $p(x)$) and supervised components (e.g., learning about $p(y|x)$) can be be helpful when both functions $p(x)$ and $p(y|x)$ share some structure.

The idea of using unsupervised learning at each stage of a deep network was recently put forward by Hinton et al. (2006), as part of a training procedure for the deep belief network (DBN), a generative model with many layers of hidden stochastic variables. Upper layers of a DBN are supposed to represent more "abstract" concepts that explain the input observation $\mathbf{x}$, whereas lower layers extract "low-level features" from $\mathbf{x}$. In other words, this model first learns simple concepts, on which it builds more abstract concepts.

This training strategy has inspired a more general approach to help address the problem of training deep networks. Hinton (2006) showed that stacking restricted Boltzmann machines (RBMs)—that is, training upper RBMs on the distribution of activities computed by lower RBMs—provides a good initialization strategy for the weights of a deep artificial neural network. This approach can be extended to non-linear autoencoders or autoassociators (Saund, 1989), as shown by Bengio et al. (2007), and is found in stacked autoassociators network (Larochelle et al., 2007), and in the deep convolutional neural network (Ranzato et al., 2007b) derived from the convolutional neural network (LeCun et al., 1998). Since then, deep networks have been applied with success not only in classification tasks (Bengio et al., 2007; Ranzato et al., 2007b; Larochelle et al., 2007; Ranzato et al., 2008), but also in regression (Salakhutdinov and Hinton, 2008), dimensionality reduction (Hinton and Salakhutdinov, 2006; Salakhutdinov and Hinton, 2007b), modeling textures (Osindero and Hinton, 2008), information retrieval (Salakhutdinov and Hinton, 2007a), robotics (Hadsell et al., 2008), natural language processing (Collobert and Weston, 2008; Weston et al., 2008), and collaborative filtering (Salakhutdinov et al., 2007).

In this paper, we discuss in detail three principles for training deep neural networks and present experimental evidence that highlight the role of each in successfully training deep networks:

1. Pre-training one layer at a time in a greedy way;

2. using unsupervised learning at each layer in a way that preserves information from the input and disentangles factors of variation;
3. fine-tuning the whole network with respect to the ultimate criterion of interest.

The experiments reported here suggest that this strategy improves on the traditional random initialization of supervised multi-layer networks by providing "hints" to each intermediate layer about the kinds of representations that it should learn, and thus initializing the supervised fine-tuning optimization in a region of parameter space from which a better local minimum (or plateau) can be reached. We also present a series of experiments aimed at evaluating the link between the performance of deep neural networks and aspects of their topology such as depth and the size of the layers. In particular, we demonstrate cases where the addition of depth helps classification error, but too much depth hurts. Finally, we explore simple variants of the aforementioned training algorithms, such as a simple way of combining them to improve their performance, RBM variants for continuous-valued inputs, as well as on-line versions of those algorithms.

2. Notations and Conventions

Before describing the learning algorithms that we will study and experiment with in this paper, we first present the mathematical notation we will use for deep networks.

A deep neural network contains an input layer and an output layer, separated by l layers of hidden units. Given an input sample clamped to the input layer, the other units of the network compute their values according to the activity of the units that they are connected to in the layers below. We will consider a particular sort of topology here, where the input layer is fully connected to the first hidden layer, which is fully connected to the second layer and so on up to the output layer.

Given an input $\mathbf{x}$, the value of the j-th unit in the i-th layer is denoted $\widehat{h}^i_j(\mathbf{x})$, with $i = 0$ referring to the input layer, $i = l+1$ referring to the output layer (the use of "$\widehat{\ }$" will become clearer in Section 4). We refer to the size of a layer as $|\widehat{\mathbf{h}}^i(\mathbf{x})|$. The default activation level is determined by the internal bias b^i_j of that unit. The set of weights W^i_{jk} between $\widehat{h}^{i-1}_k(\mathbf{x})$ in layer $i-1$ and unit $\widehat{h}^i_j(\mathbf{x})$ in layer i determines the activation of unit $\widehat{h}^i_j(\mathbf{x})$ as follows:

$$\widehat{h}^i_j(\mathbf{x}) = \mathrm{sigm}\left(a^i_j\right) \text{ where } a^i_j(\mathbf{x}) = b^i_j + \sum_k W^i_{jk}\widehat{h}^{i-1}_k(\mathbf{x})\ \forall i \in \{1,\ldots,l\}, \text{ with } \widehat{h}^0(\mathbf{x}) = \mathbf{x} \tag{1}$$

where $\mathrm{sigm}(\cdot)$ is the sigmoid squashing function: $\mathrm{sigm}(a) = \frac{1}{1+e^{-a}}$ (alternatively, the sigmoid could be replaced by the hyperbolic tangent). Given the last hidden layer, the output layer is computed similarly by

$$\mathbf{o}(\mathbf{x}) = \widehat{\mathbf{h}}^{l+1}(\mathbf{x}) = f\left(\mathbf{a}^{l+1}(\mathbf{x})\right) \text{ where } \mathbf{a}^{l+1}(\mathbf{x}) = \mathbf{b}^{l+1} + \mathbf{W}^{l+1}\widehat{\mathbf{h}}^l(\mathbf{x})$$

where the activation function $f(\cdot)$ depends on the (supervised) task the network must achieve. Typically, it will be the identity function for a regression problem and the softmax function

$$f_j(\mathbf{a}) = \mathrm{softmax}_j(\mathbf{a}) = \frac{e^{a_j}}{\sum_{k=1}^{K} e^{a_k}} \tag{2}$$

for a classification problem, in order to obtain a distribution over the K classes.

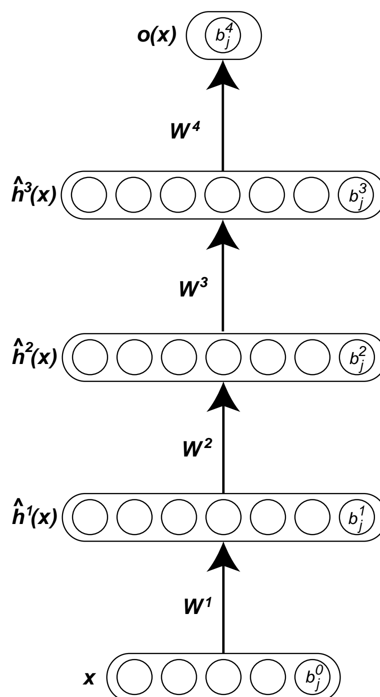


Figure 1: Illustration of a deep network and its parameters.

When an input sample **x** is presented to the network, the application of Equation 1 at each layer will generate a pattern of activity in the different layers of the neural network. Intuitively, we would like the activity of the first layer neurons to correspond to low-level features of the input (e.g., edge orientations for natural images) and to higher level abstractions (e.g., detection of geometrical shapes) in the last hidden layers.

3. Deep Neural Networks

It has been shown that a "shallow" neural network with only one arbitrarily large hidden layer could approximate a function to any level of precision (Hornik et al., 1989). Similarly, any Boolean function can be represented by a two-layer circuit of logic gates. However, most Boolean functions require an exponential number of logic gates (with respect to the input size) to be represented by a two-layer circuit (Wegener, 1987). For example, the parity function, which can be efficiently represented by a circuit of depth $O(\log n)$ (for n input bits) needs $O(2^n)$ gates to be represented by a depth two circuit (Yao, 1985). What about deeper circuits? Some families of functions which can be represented with a depth k circuit are such that they require an exponential number of logic gates to be represented by a depth $k-1$ circuit (Håstad, 1986). Interestingly, an equivalent result has been proved for architectures whose computational elements are not logic gates but linear threshold units (i.e., formal neurons) (Hastad and Goldmann, 1991). The machine learning literature also suggests that shallow architectures can be very inefficient in terms of the number of computational units (e.g., bases, hidden units), and thus in terms of required examples (Bengio and Le Cun, 2007; Bengio et al., 2006). On the other hand, a highly-varying function can sometimes be represented

compactly (with fewer parameters) through the composition of many non-linearities, that is, with a deep architecture. When the representation of a concept requires an exponential number of elements (more generally exponential capacity), for example, with a shallow circuit, the number of training examples required to learn the concept may also be impractical. Smoothing the learned function by regularization would not solve the problem here because in these cases the target function itself is complicated and requires exponential capacity just to be represented.

3.1 Difficulty of Training Deep Architectures

Given a particular task, a natural way to train a deep network is to frame it as an optimization problem by specifying a supervised cost function on the output layer with respect to the desired target and use a gradient-based optimization algorithm in order to adjust the weights and biases of the network so that its output has low cost on samples in the training set. Unfortunately, deep networks trained in that manner have generally been found to perform worse than neural networks with one or two hidden layers.

We discuss two hypotheses that may explain this difficulty. The first one is that gradient descent can easily get stuck in poor local minima (Auer et al., 1996) or plateaus of the non-convex training criterion. The number and quality of these local minima and plateaus (Fukumizu and Amari, 2000) clearly also influence the chances for random initialization to be in the basin of attraction (via gradient descent) of a poor solution. It may be that with more layers, the number or the width of such poor basins increases. To reduce the difficulty, it has been suggested to train a neural network in a constructive manner in order to divide the hard optimization problem into several greedy but simpler ones, either by adding one neuron (e.g., see Fahlman and Lebiere, 1990) or one layer (e.g., see Lengellé and Denoeux, 1996) at a time. These two approaches have demonstrated to be very effective for learning particularly complex functions, such as a very non-linear classification problem in 2 dimensions. However, these are exceptionally hard problems, and for learning tasks usually found in practice, this approach commonly overfits.

This observation leads to a second hypothesis. For high capacity and highly flexible deep networks, there actually exists many basins of attraction in its parameter space (i.e., yielding different solutions with gradient descent) that can give low training error but that can have very different generalization errors. So even when gradient descent is able to find a (possibly local) good minimum in terms of training error, there are no guarantees that the associated parameter configuration will provide good generalization. Of course, model selection (e.g., by cross-validation) will partly correct this issue, but if the number of good generalization configurations is very small in comparison to good training configurations, as seems to be the case in practice, then it is likely that the training procedure will not find any of them. But, as we will see in this paper, it appears that the type of unsupervised initialization discussed here can help to select basins of attraction (for the supervised fine-tuning optimization phase) from which learning good solutions is easier both from the point of view of the training set and of a test set.

3.2 Unsupervised Learning as a Promising Paradigm for Greedy Layer-Wise Training

A common approach to improve the generalization performance of a learning algorithm which is motivated by the Occam's razor principle is the use of regularization (such as weight decay) that will favor "simpler" models over more complicated ones. However, using generic priors such as the ℓ^2 norm of the parameters conveys limited information about what the solution to a particular learn-

ing task should be. This has motivated researchers to discover more meaningful, data-dependent regularization procedures, which are usually based on unsupervised learning and normally adapted to specific models.

For example, Ando and Zhang (2005) use "auxiliary tasks" designed from unlabelled data and that are appropriate for a particular learning problem, to learn a better regularization term for linear classifiers. Partial least squares (Frank and Friedman, 1993) can also be seen as combining unsupervised and supervised learning in order to learn a better linear regression model when few training data are available or when the input space is very high dimensional.

Many semi-supervised learning algorithms also involve a combination of unsupervised and supervised learning, where the unsupervised component can be applied to additional unlabelled data. This is the case for Fisher-kernels (Jaakkola and Haussler, 1999) which are based on a generative model trained on unlabelled input data and that can be used to solve a supervised problem defined for that input space. In all these cases, unsupervised learning can be seen as adding more constraints on acceptable configurations for the parameters of a model, by asking that it not only describes well the relationship between the input and the target but also contains relevant statistical information about the structure of the input or how it was generated.

Moreover, there is a growing literature on the distinct advantages of generative and discriminative learning. Ng and Jordan (2001) argue that generative versions of discriminative models can be expected to reach their usually higher asymptotic out-of-sample classification error faster (i.e., with less training data), making them preferable in certain situations. Moreover, successful attempts at exploring the space between discriminative and generative learning have been studied (Lasserre et al., 2006; Jebara, 2003; Bouchard and Triggs, 2004; Holub and Perona, 2005).

The deep network learning algorithms that have been proposed recently and that we study in this paper can be seen as combining the ideas of *greedily learning the network* to break down the learning problem into easier steps, *using unsupervised learning* to provide an *effective hint* about what hidden units should learn, bringing along the way a form of regularization that prevents overfitting even in deep networks with many degrees of freedom (which could otherwise overfit). In addition, one should consider the supervised task the network has to solve. The greedy layer-wise unsupervised strategy provides an initialization procedure, after which the neural network is *fine-tuned to the global supervised objective*. The general paradigm followed by these algorithms (illustrated in Figure 2 and detailed in Appendix A) can be decomposed in two phases:

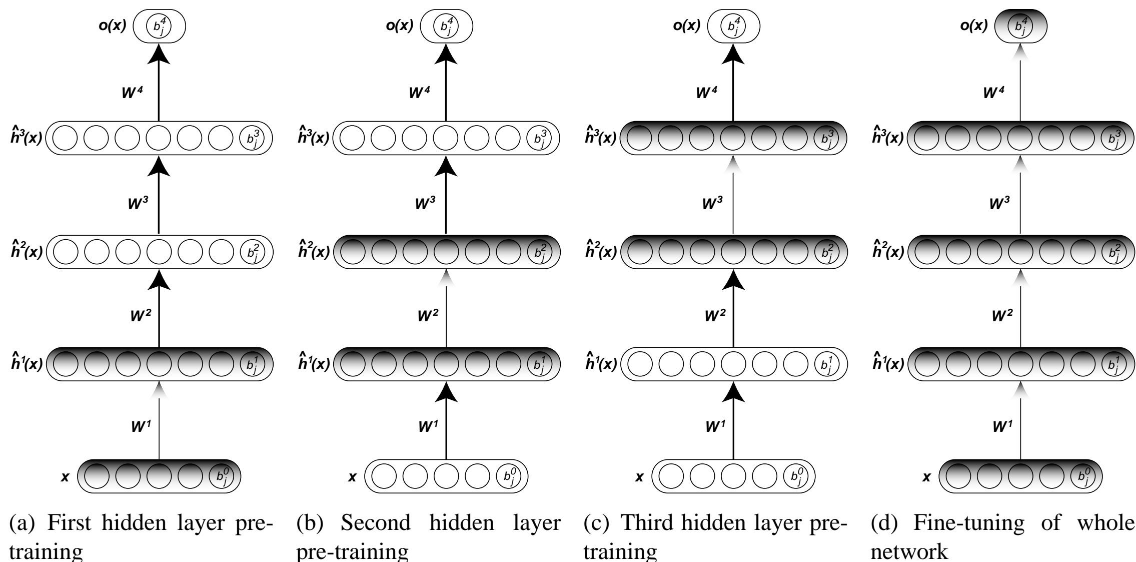
1. In the first phase, greedily train subsets of the parameters of the network using a layer-wise and unsupervised learning criterion, by repeating the following steps for each layer ($i \in \{1, \ldots, l\}$)

    Until a stopping criteria is met, iterate through training database by

    (a) mapping input training sample $\mathbf{x}_t$ to representation $\widehat{\mathbf{h}}^{i-1}(\mathbf{x}_t)$ (if $i > 1$) and hidden representation $\widehat{\mathbf{h}}^{i}(\mathbf{x}_t)$,

    (b) updating parameters $\mathbf{b}^{i-1}$, $\mathbf{b}^{i}$ and $\mathbf{W}^{i}$ of layer i using some unsupervised learning algorithm.

    Also, initialize (e.g., randomly) the output layer parameters $\mathbf{b}^{l+1}, \mathbf{W}^{l+1}$.



(a) First hidden layer pre-training

(b) Second hidden layer pre-training

(c) Third hidden layer pre-training

(d) Fine-tuning of whole network

Figure 2: Unsupervised greedy layer-wise training procedure.

2. In the second and final phase, fine-tune all the parameters θ of the network using backpropagation and gradient descent on a global supervised cost function $C(\mathbf{x}_t, y_t, \theta)$, with input $\mathbf{x}_t$ and label y_t, that is, trying to make steps in the direction $E\left[\frac{\partial C(\mathbf{x}_t, y_t, \theta)}{\partial \theta}\right]$.

Regularization is not explicit in this procedure, as it does not come from a weighted term that depends on the complexity of the network and that is added to the global supervised objective. Instead, it is implicit, as the first phase that initializes the parameters of the whole network will ultimately have an impact on the solution found in the second phase (the fine-tuning phase). Indeed, by using an iterative gradual optimization algorithm such as stochastic gradient descent with early-stopping (i.e., training until the error on a validation set reaches a clear minimum), the extent to which the configuration of the network's parameters can be different from the initial configuration given by the first phase is limited. Hence, similarly to using a regularization term on the parameters of the model that constrains them to be close to a particular value (e.g., 0 for weight decay), the first phase here will ensure that the parameter solution for each layer found by fine-tuning will not be far from the solution found by the unsupervised learning algorithm. In addition, the non-convexity of the supervised training criterion means that the choice of initial parameter values can greatly influence the quality of the solution obtained by gradient descent.

In the next two sections, we present a review of the two training algorithms that fall in paradigm presented above and which are empirically studied in this paper, in Section 6.

4. Stacked Restricted Boltzmann Machine Network

Intuitively, a successful learning algorithm for deep networks should be one that discovers a meaningful and possibly complex hidden representation of the data at its top hidden layer. However, learning such non-linear representations is a hard problem. A solution, proposed by Hinton (2006), is based on the learning algorithm of the restricted Boltzmann machine (RBM) (Smolensky, 1986), a generative model that uses a layer of binary variables to explain its input data. In an RBM (see

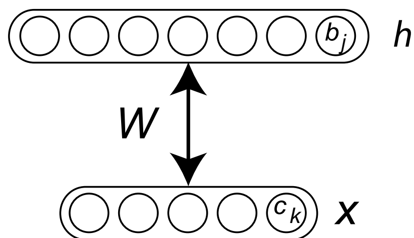


Figure 3: Illustration of a restricted Boltzmann machine and its parameters. $\mathbf{W}$ is a weight matrix, $\mathbf{b}$ is a vector of hidden unit biases, and $\mathbf{c}$ a vector of visible unit biases.

Figure 3 for an illustration), given an input $\mathbf{x}$, it is easy to obtain a hidden representation for that input by computing the posterior $\widehat{\mathbf{h}}(\mathbf{x})$ over the layer of binary hidden variables $\mathbf{h}$ (we use the "$\widehat{\ }$" symbol to emphasize that $\widehat{\mathbf{h}}(\mathbf{x})$ is not a random variable but a deterministic representation of $\mathbf{x}$).

Hinton (2006) argues that this representation can be improved by giving it as input to another RBM, whose posterior over its hidden layer will then provide a more complex representation of the input. This process can be repeated an arbitrary number of times in order to obtain ever more non-linear representations of the input. Finally, the parameters of the RBMs that compute these representations can be used to initialize the parameters of a deep network, which can then be fine-tuned to a particular supervised task. This learning algorithm clearly falls in the paradigm of Section 3.2, where the unsupervised part of the learning algorithm is that of an RBM. We will refer to deep networks trained using this algorithm as stacked restricted Boltzmann machine (SRBM) networks. For more technical details about the SRBM network, and how to train an RBM using the contrastive divergence algorithm (CD-k), see Appendix B.

5. Stacked Autoassociators Network

There are theoretical results about the advantage of stacking many RBMs into a DBN: Hinton et al. (2006) show that this procedure optimizes a bound on the likelihood of the input data when all layers have the same size. An additional hypothesis to explain why this process provides a good initialization for the network is that it makes each hidden layer compute a different, possibly more abstract representation of the input. This is done implicitly, by asking that each layer captures features of the input that help characterize the distribution of values at the layer below. By transitivity, each layer contains some information about the input. However, stacking any unsupervised learning model does not guarantee that the representations learned get increasingly complex or appropriate as we stack more layers. For instance, many layers of linear PCA models could be summarized by only one layer. However, there may be other non-linear, unsupervised learning models that, when stacked, are able to improve the learned representation at the last layer added.

An example of such a non-linear unsupervised learning model is the autoassociator or autoencoder network (Cottrell et al., 1987; Saund, 1989; Hinton, 1989; Baldi and Hornik, 1989; DeMers and Cottrell, 1993). Autoassociators are neural networks that are trained to compute a representation of the input from which it can be reconstructed with as much accuracy as possible. In this paper, we will consider autoassociator networks of only one hidden layer, meaning that the hidden

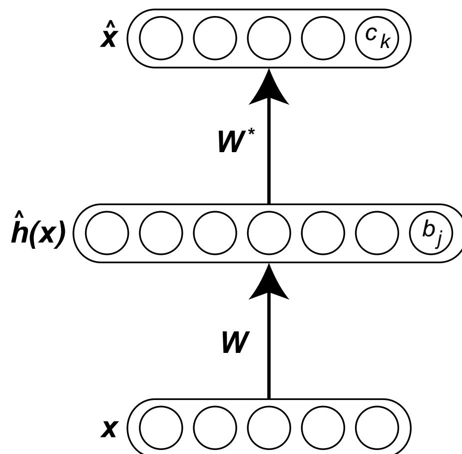


Figure 4: Illustration of an autoassociator and its parameters. $\mathbf{W}$ is the matrix of encoder weights and $\mathbf{W}^*$ the matrix of decoder weights. $\widehat{\mathbf{h}}(\mathbf{x})$ is the code or representation of $\mathbf{x}$.

representation of $\mathbf{x}$ is a code $\widehat{\mathbf{h}}(x)$ obtained from the encoding function

$$\widehat{h}_j(\mathbf{x}) = f(a_j) \text{ where } a_j(\mathbf{x}) = b_j + \sum_k W_{jk} x_k \ . \tag{3}$$

The input's reconstruction is obtained from a decoding function, here a linear transformation of the hidden representation with weight matrix $\mathbf{W}^*$, possibly followed by a non-linear activation function:

$$\widehat{x}_k = g(\widehat{a}_k) \text{ where } \widehat{a}_k = c_k + \sum_j W^*_{jk} \widehat{h}_j(\mathbf{x}) \ .$$

In this work, we used the sigmoid activation function for both $f(\cdot)$ and $g(\cdot)$. Figure 4 shows an illustration of this model.

By noticing the similarity between Equations 3 and 1, we are then able to use the training algorithm for autoassociators as the unsupervised learning algorithm for the greedy layer-wise initialization phase of deep networks. In this paper, stacked autoassociators (SAA) networks will refer to deep networks trained using the procedure of Section 3.2 and the learning algorithm of an autoassociator for each layer, as described in Section 5.1.

Though these neural networks were designed with the goal of dimensionality reduction in mind, the new representation's dimensionality does not necessarily need to be lower than the input's in practice. However, in that particular case, some care must be taken so that the network does not learn a trivial identity function, that is, finds weights that simply "copy" the whole input vector in the hidden layer and then copy it again at the output. For example, a network with small weights W_{jk} between the input and hidden layers (maintaining activations in the linear regime of the activation function f) and large weights W^*_{jk} between the hidden and output layers could encode such an uninteresting identity function. An easy way to avoid such a pathological behavior in the case of continuous inputs is to set the weight matrices $\mathbf{W}^\top$ and $\mathbf{W}^*$ to be the same. This adjustment is motivated by its similarity with the parametrization of the RBM model and by an empirical observation that $\mathbf{W}^\top$ and $\mathbf{W}^*$ tend to be similar up to a multiplicative factor after training. In

the case of binary inputs, if the weights are large, the input vector can still be copied (up to a permutation of the elements) to the hidden units, and in turn these used to perfectly reconstruct the input. Weight decay can be useful to prevent such a trivial and uninteresting mapping to be learned, when the inputs are binary. We set $\mathbf{W}^{\mathsf{T}} = \mathbf{W}^*$ in all of our experiments. Vincent et al. (2008) have an improved way of training autoassociators in order to produce interesting, non-trivial features in the hidden layer, by partially corrupting the network's inputs.

The reconstruction error of an autoassociator can be connected to the log-likelihood of an RBM in several ways. Ranzato et al. (2008) connect the log of the numerator of the input likelihood with a form of reconstruction error (where one replaces the sum over hidden unit configurations by a maximization). The denominator is the normalization constant summing over all input configurations the same expression as in the numerator. So whereas maximizing the numerator is similar to minimizing reconstruction error for the training examples, minimizing the denominator means that most input configurations should not be reconstructed well. This can be achieved if the autoassociator is constrained in such a way that it cannot compute the identity function, but only minimizes the reconstruction for training examples.

Another connection between reconstruction error and log-likelihood of the RBM was made in Bengio and Delalleau (2007). They consider a converging series expansion of the log-likelihood gradient and show that whereas CD-k truncates the series by keeping the first $2k$ terms and then approximates expectations by a single sample, reconstruction error is a mean-field approximation of the first term in that series.

5.1 Learning in an Autoassociator Network

Training an autoassociator network is almost identical to training a standard artificial neural network. Given a cost function, backpropagation is used to compute gradients and perform gradient descent. However, autoassociators are "self-supervised", meaning that the target to which the output of the autoassociator is compared is the input that it was fed.

Previous work on autoassociators minimized the squared reconstruction error:

$$C(\widehat{\mathbf{x}}, \mathbf{x}) = \sum_k (\widehat{x}_k - x_k)^2 \ .$$

However, with squared reconstruction error and linear decoder, the "optimal codes" (the implicit target for the encoder, irrespective of the encoder) are in the span of the principal eigenvectors of the input covariance matrix. When we introduce a saturating non-linearity such as the sigmoid, and we want to reconstruct values $[0,1]$, the binomial KL divergence (also known as cross-entropy) seems more appropriate:

$$C(\widehat{\mathbf{x}}, \mathbf{x}) = -\sum_k \left(x_k \log(\widehat{x}_k) + (1 - x_k) \log(1 - \widehat{x}_k) \right) \ . \tag{4}$$

It corresponds to the assumption that $\widehat{\mathbf{x}}$ and $\mathbf{x}$ can be interpreted as factorized distributions over binary units. It is well known that the cross-entropy $-p\log(q) - (1-p)\log(1-q)$ between two binary distributions parametrized by p and q is minimized when $q = p$ (for a fixed p), making it an appropriate cost function to evaluate the quality of a reconstruction. We used this cost function in all the experiments with SAA networks. Appendix C details the corresponding autoassociator training update.

6. Experiments

In this section, we present several experiments set up to evaluate the deep network learning algorithms that fall in the paradigm presented in the Section 3.2 and highlight some of their properties. Unless otherwise stated, stochastic gradient descent was used for layer-wise unsupervised learning (first phase of the algorithm) and global supervised fine-tuning (second phase of the algorithm). The data sets were separated in disjoint training, validation and testing subsets. Model selection consisted of finding the best values for the learning rates of layer-wise unsupervised and global supervised learning as well as the number of unsupervised updates preceding the fine-tuning phase. The number of epochs of fine-tuning was chosen using early-stopping based on the progression of classification error on the validation set. All experiments correspond to classification problems. Hence, to fine-tune the deep networks, we optimized the negative conditional log-likelihood of the training samples' target class (as given by the softmax output of the neural network).

The experiments are based on the MNIST data set[1] (see Figure 5), a benchmark for handwritten digit recognition, as well as variants of this problem where the input distribution has been made more complex by inserting additional factors of variations, such as rotations and background images. The input images are made of 28×28 pixels giving an input dimensionality of 784, the number of classes is 10 (corresponding to the digits from 0 to 9) and the inputs were scaled between 0 and 1.

Successful applications of deep networks have already been presented on a large variety of data, such as images of faces (Salakhutdinov and Hinton, 2008), real-world objects (Ranzato et al., 2007a) as well as text data (Hinton and Salakhutdinov, 2006; Salakhutdinov and Hinton, 2007a; Collobert and Weston, 2008; Weston et al., 2008), and on different types of problems such as regression (Salakhutdinov and Hinton, 2008), information retrieval (Salakhutdinov and Hinton, 2007a), robotics Hadsell et al. (2008), and collaborative filtering (Salakhutdinov et al., 2007).

In Bengio et al. (2007), we performed experiments on two regression data sets, with non-image continuous inputs (UCI Abalone, and a financial data set), demonstrating the use of unsupervised (or partially supervised) pre-training of deep networks on these tasks. In Larochelle et al. (2007), we studied the performance of several architectures on various data sets, including variations of MNIST (with rotations, random background, and image background), and discrimination tasks between wide and tall rectangles, and between convex and non-convex images. On these tasks, deep networks compared favorably to shallow architectures.

Our focus is hence not on demonstrating their usefulness on a wide range of tasks, but on exploring their properties empirically. Such experimental work required several weeks of cumulative CPU time, which restricted the number of data sets we could explore. However, by concentrating on the original MNIST data set and harder versions of it, we were able not only to confirm the good performance of deep networks, but also to study practical variations, to help understand the algorithms, and to discuss the impact on a deep network's performance of stepping to a more complicated problem.

6.1 Validating the Unsupervised Layer-Wise Strategy for Deep Networks

In this section, we evaluate the advantages brought by the unsupervised layer-wise strategy of Section 3.2. We want to separate the different algorithmic concepts behind it, in order to understand their contribution to the whole strategy. In particular, we pursue the following two questions:

1. This data set can be downloaded from `http://yann.lecun.com/expdb/mnist/`.

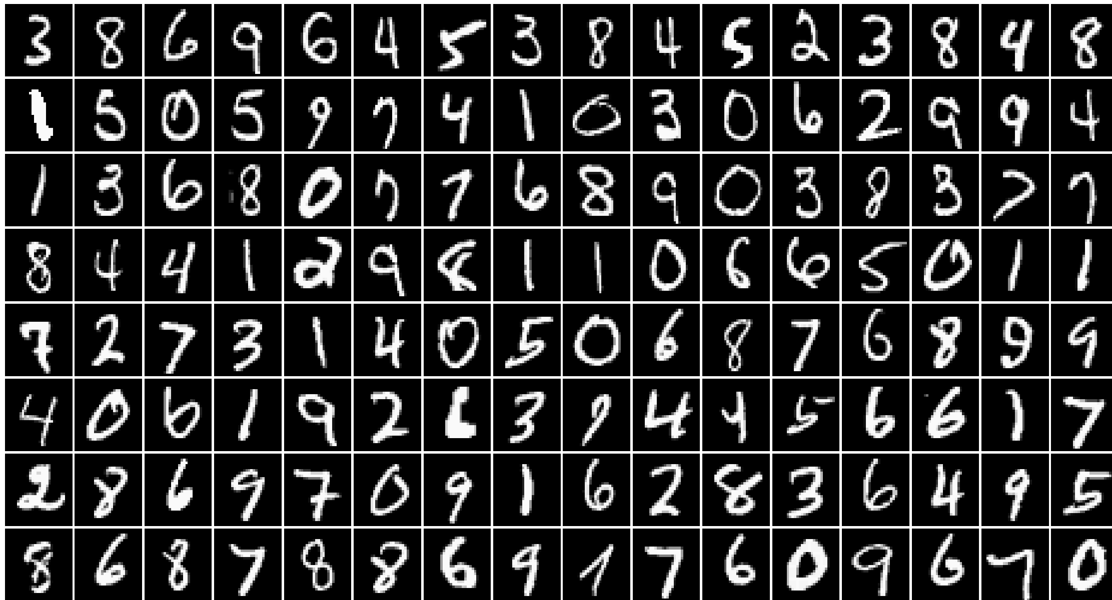


Figure 5: Samples from the MNIST digit recognition data set. Here, a black pixel corresponds to an input value of 0 and a white pixel corresponds to 1 (the inputs are scaled between 0 and 1).

1. To what extent does initializing greedily the parameters of the different layers help?
2. How important is unsupervised learning for this procedure?

To address these two questions, we will compare the learning algorithms for deep networks of Sections 4 and 5 with the following algorithms.

6.1.1 DEEP NETWORK WITHOUT PRE-TRAINING

To address the first question above, we compare the greedy layer-wise algorithm with a more standard way to train neural networks: using standard backpropagation and stochastic gradient descent, and starting at a randomly initialized configuration of the parameters. In other words, this variant simply puts away the pre-training phase of the other deep network learning algorithms.

6.1.2 DEEP NETWORK WITH SUPERVISED PRE-TRAINING

To address the second question, we run an experiment with the following algorithm. We greedily pre-train the layers using a *supervised criterion* (instead of the unsupervised one), before performing as before a final supervised fine-tuning phase. Specifically, when greedily pre-training the parameters $\mathbf{W}^i$ and $\mathbf{b}^i$, we also train another set of weights $\mathbf{V}^i$ and biases $\mathbf{c}^i$ which connect hidden layer $\widehat{\mathbf{h}}^i(\mathbf{x})$ to a temporary output layer as follows:

$$o^i(\mathbf{x}) = f\left(\mathbf{c}^i + \mathbf{V}^i\widehat{\mathbf{h}}^i(\mathbf{x})\right)$$

where $f(\cdot)$ is the softmax function of Equation 2. This output layer can be trained using the same cost as the global supervised cost. However, as this is a greedy procedure, only the parameters $\mathbf{W}^i$, $\mathbf{b}^i$, $\mathbf{V}^i$ and $\mathbf{c}^i$ are updated, that is, the gradient is not propagated to the layers below. When the training of a layer is finished, we can simply discard the parameters $\mathbf{V}^i$ and $\mathbf{c}^i$ and move to pre-training the next hidden layer, having initialized $\mathbf{W}^i$ and $\mathbf{b}^i$.

6.1.3 STACKED LOGISTIC AUTOREGRESSION NETWORK

The second question aims at evaluating to what extent any unsupervised learning can help. We already know that stacking linear PCA models is not expected to help improve generalization. A slightly more complex yet very simple unsupervised model for data in $[0,1]$ is the logistic autoregression model (see also Frey, 1998)

$$\widehat{x}_k = \mathrm{sigm}\left(b_k + \sum_{j \neq k} W_{kj} x_j\right) \tag{5}$$

where the reconstruction $\widehat{\mathbf{x}}$ is log-linear in the input $\mathbf{x}$. The parameters $\mathbf{W}$ and $\mathbf{b}$ can be trained using the same cost used for the autoassociators in Equation 4. This model can be used to initialize the weights $\mathbf{W}^i$ and biases $\mathbf{b}^i$ of the i-th hidden layer of a deep network. However, because $\mathbf{W}$ in Equation 5 is square, the deep network will need to have hidden layers with the same size as the input layer. Also, the weights on the diagonal of $\mathbf{W}$ are not trained in this model, so we initialize them to zero. The stacked logistic autoregression network will refer to deep networks using this unsupervised layer-wise learning algorithm.

6.1.4 RESULTS

The results for all these deep networks are given in Table 1. We also give results for a "shallow", one hidden layer neural network, to validate the utility of deep architectures. Instead of the sigmoid, this network uses hyperbolic tangent squashing functions, which are usually found to work better for one hidden layer neural networks. The MNIST training set was separated into training (50,000) and validation (10,000) sets. The test set has size 10,000. In addition to the hyperparameters mentioned at the beginning of this section, the validation set was used also to select appropriate decrease constants[2] for the learning rates of the greedy and fine-tuning phases. The SRBM and SAA networks had 500, 500 and 2000 hidden units in the first, second and third layers respectively, as in Hinton et al. (2006) and Hinton (2006). In the pre-training phase of the SRBM and SAA networks, when training the parameters of the i-th layer, the down-biases c_k where set to be equal to b_k^{i-1} (although similar results were obtained by using a separate set of biases c_k^{i-1} when the $i-1$-th layer is the down-layer). For the deep networks with supervised or no pre-training, different sizes of hidden layers were compared, including sizes similar to the stacked logistic autoregression network, and to the SRBM and SAA networks. All deep networks had 3 hidden layers.

Overall, the models that use the unsupervised layer-wise procedure of Section 3.2 outperform those that do not. We also observe a slight advantage in the performance of the SRBM network over that of the SAA network (on the MNIST test set, differences of more than 0.1% are statistically significant). The performance difference between the stacked logistic autoregressions network and

2. When using a decrease constant β, the learning rate for the tth update becomes $\frac{\varepsilon_0}{1+t\beta}$, where ε_0 is the initial learning rate.

Models	Train.	Valid.	Test
SRBM (stacked restricted Boltzmann machines) network	0%	1.20%	1.20%
SAA (stacked autoassociators) network	0%	1.31%	1.41%
Stacked logistic autoregressions network	0%	1.65%	1.85%
Deep network with supervised pre-training	0%	1.74%	2.04%
Deep network, no pre-training	0.004%	2.07%	2.40%
Shallow network, no pre-training	0%	1.91%	1.93%

Table 1: Classification error on MNIST training, validation, and test sets, with the best hyperparameters according to validation error.

the deep network with supervised layer-wise pre-training particularly highlights the importance of unsupervised learning. Indeed, even though supervised layer-wise pre-training explicitly trains the hidden layers to capture non-linear information about the input, the overall procedure seems to be too greedy with respect to the supervised task to be learned. On the other hand, even though logistic autoregressions are simple log-linear models and their optimization is blind with respect to the future usage of the weights $\mathbf{W}$ as connections into non-linear hidden layers, the unsupervised nature of training makes them still useful for improving generalization. As a point of comparison, besides the deep networks, the best result on this data set reported for a learning algorithm that does not use any prior knowledge about the task (e.g., image pre-processing like deskewing or subsampling) is that of a support vector machine with a Gaussian kernel,[3] with 1.4% classification error on the test set.

At this point, it is clear that unsupervised layer-wise pre-training improves generalization. However, we could wonder whether it also facilitates the optimization problem of the global fine-tuning. The results of Table 1 do not shed any light on this aspect. Indeed, all the networks, even those without greedy layer-wise pre-training, perform almost perfectly on the training set. The explanatory hypothesis we evaluate here is that, without pre-training, the lower layers are initialized poorly, but still allow the top two layers to learn the training set almost perfectly because the output layer and the last hidden layer form a standard shallow but fat neural network. Consider the top two layers of the deep network *with pre-training*: it presumably takes as input a *better representation*, one that allows for better generalization. Instead, the network *without pre-training* sees a "random" transformation of the input, one that preserves enough information about the input to fit the training set, but that does not help to generalize. To test this hypothesis, we performed a second series of experiments in which we constrain the top hidden layer to be small (20 hidden units).

The results (Table 2) clearly suggest that optimization of the global supervised objective is made easier by greedy layer-wise pre-training. This result for supervised greedy pre-training is also coherent with past experiments on similar greedy strategies (Fahlman and Lebiere, 1990; Lengellé and Denoeux, 1996). Here, we have thus confirmed that it also applies to unsupervised greedy pre-training. With no pre-training, training error degrades significantly when there are only 20 hidden units in the top hidden layer. In addition, the results obtained without pre-training were found to have much larger variance than those with pre-training, indicating high sensitivity to initial

3. See `http://yann.lecun.com/exdb/mnist/` for more details.

Models	Train.	Valid.	Test
SRBM network	0%	1.5%	1.5%
SAA network	0%	1.38%	1.65%
Deep network with supervised pre-training	0%	1.77%	1.89%
Deep network, no pre-training	0.59%	2.10%	2.20%
Shallow network, no pre-training	3.6%	4.77%	5.00%

Table 2: Classification error on MNIST training, validation, and test sets, with the best hyperparameters according to validation error, when the last hidden layer only contains 20 hidden units

conditions: the unsupervised pre-training more consistently puts the parameters in a "good" basin of attraction for the supervised gradient descent procedure.

Figures 6 and 7 show the sorts of first hidden layer features (weights going into different hidden neurons) that are learned by the first (bottom) RBM and autoassociator respectively, before fine-tuning. Both models were trained on the MNIST training set of Section 6.1 for 40 epochs, with 250 hidden units and a learning rate of 0.005. We see that they both learn visual features characterized by local receptive fields, which ought to be useful to recognize more global shapes (though the autoassociator also learns high frequency receptive fields that are spread over the whole image). This is another account of how unsupervised greedy pre-training is able to help the optimization of the network. Even if the supervised fine-tuning gradient at the first hidden layer is weak, we can see that the first hidden layer appears to learn a relevant representation.

6.2 Exploring the Space of Network Architectures

An important practical aspect in using deep network is the choice the architecture or topology of the network. Once we allow ourselves to consider an arbitrary number of hidden layers of arbitrary sizes, some questions naturally arise. First, we would like to know how deep a neural network can be made while still obtaining generalization gains, given a strategy for initializing its parameters (randomly or with unsupervised greedy pre-training). We would also like to know, for a determined depth, what type of architecture is more appropriate. Should the hidden layer's size increase, decrease or stay the same from the first to the last? In this section, we explore those two questions with experiments on the MNIST data set as well as a variant, taken from Larochelle et al. (2007), where the digit images have been randomly rotated. This last data set, noted MNIST-rotation[4] (see Figure 8), contains much more intraclass variability, is much less well described by relatively well separated class-specific clusters and corresponds to a much harder classification problem. The training, validation and test sets contain 10 000, 2 000 and 50 000 examples each. We also generated sets of the same size for the MNIST data set. We refer to this version with a smaller training set by MNIST-small.

4. This data set has been regenerated since Larochelle et al. (2007) and is available here: `http://www.iro.umontreal.ca/~lisa/icml2007`.

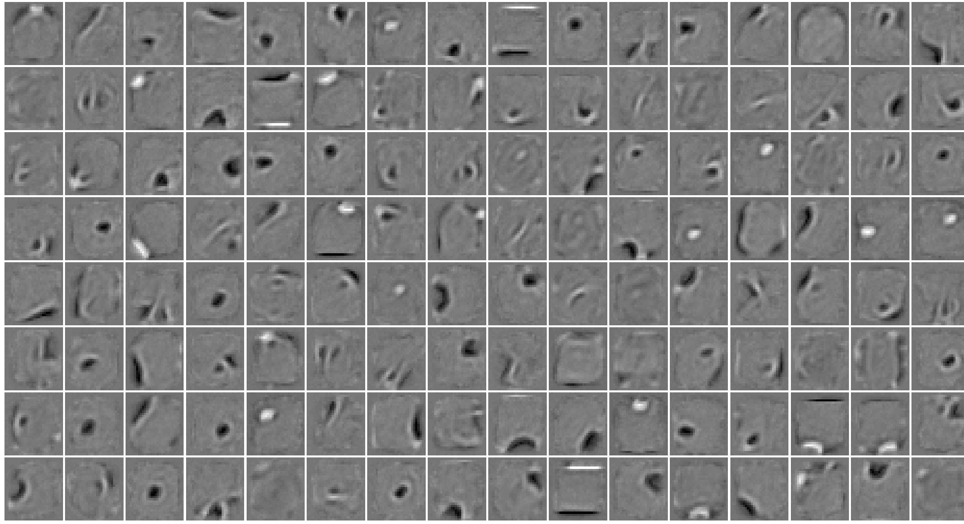


Figure 6: Display of the input weights of a random subset of the hidden units, learned by an RBM when trained on samples from the MNIST data set. The activation of units of the first hidden layer is obtained by a dot product of such a weight "image" with the input image. In these images, a black pixel corresponds to a weight smaller than -3 and a white pixel to a weight larger than 3, with the different shades of gray corresponding to different weight values uniformly between -3 and 3.

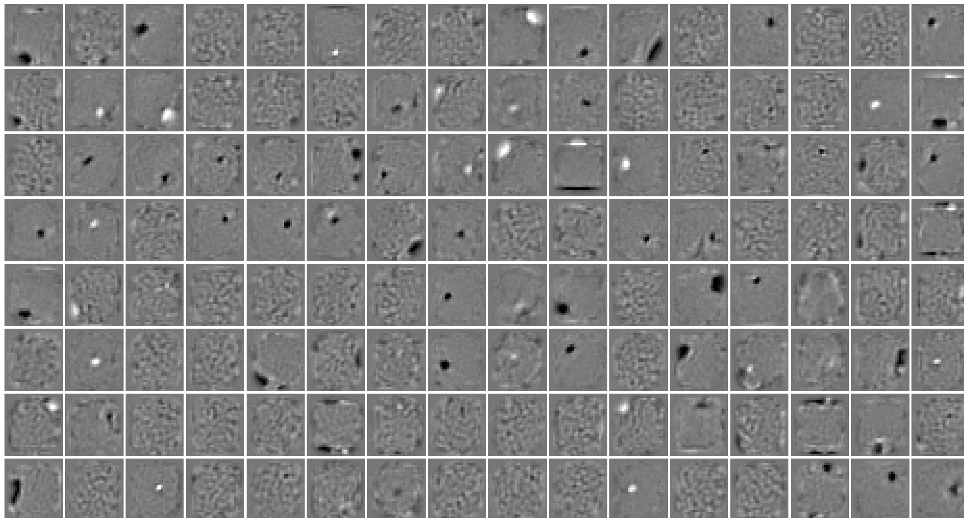


Figure 7: Input weights of a random subset of the hidden units, learned by an autoassociator when trained on samples from the MNIST data set. The display setting is the same as for Figure 6.

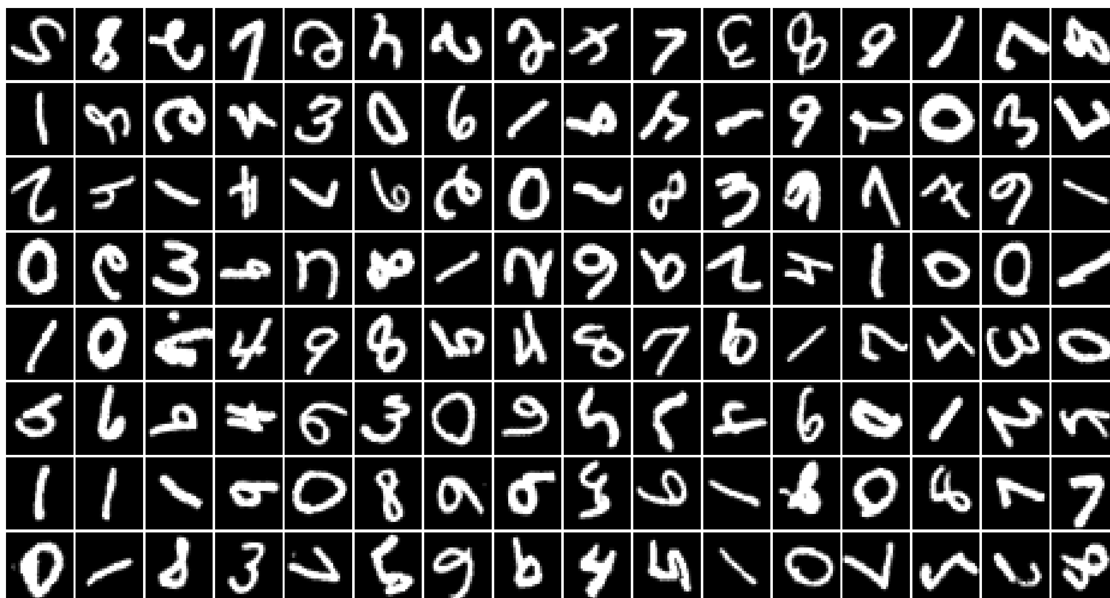


Figure 8: Samples from the MNIST-rotation data set. Here, a black pixel corresponds to an input value of 0 and a white pixel corresponds to 1 (the inputs are scaled between 0 and 1).

6.2.1 NETWORK DEPTH

One can wonder whether a neural network can be made too deep, that is, whether having too many hidden layers can worsen the generalization performance. Of course there are many reasons why this may happen. When a neuron is added, more parameters are inserted in the mathematical formulation of the model, giving it more degrees of freedom to fit the model and hence possibly making it able to overfit. On the other hand, it is less clear to what extent the performance can worsen, since a neuron added at the top layer of a neural network does not increase the capacity the same way a neuron added "in parallel" in a given hidden layer. Also, in the case of an SRBM network, we can imagine that as we stack RBMs, the representation at a hidden layer contains units that correspond to more and more disentangled concepts of the input. Now, consider a hypothetical deep network where the top-level stacked RBM has learned a representation made of units that are mostly independent. An additional RBM stacked on this representation would have no statistical structure to learn. This would initialize the weights of that new RBM to zero, which is particularly troublesome as the representation at this level would then contain no information about the input. It is not clear if this scenario is plausible, unlike in the case of independent component analysis, but if it were approached the result would be detrimental to supervised classification performance. This particular situation is not expected with stacked autoassociators, as it will always learn a representation from which the previous layer can be reconstructed. Another reason why a deeper architecture could produce worse results is simply that our algorithms for training a deep architecture can probably be improved. In particular, note that the only joint training of all layers that we have done in our experiments, if any, is at the supervised fine-tuning stage.

Network		MNIST-small classif. test error	MNIST-rotation classif. test error
Type	Depth		
Neural network	1	**4.14** % ± 0.17	15.22 % ± 0.31
(random initialization,	2	**4.03** % ± 0.17	**10.63** % ± 0.27
+ fine-tuning)	3	**4.24** % ± 0.18	11.98 % ± 0.28
	4	4.47 % ± 0.18	11.73 % ± 0.29
SAA network	1	3.87 % ± 0.17	11.43% ± 0.28
(autoassociator learning	2	**3.38** % ± 0.16	9.88 % ± 0.26
+ fine-tuning)	3	**3.37** % ± 0.16	**9.22** % ± 0.25
	4	**3.39** % ± 0.16	**9.20** % ± 0.25
SRBM network	1	3.17 % ± 0.15	10.47 % ± 0.27
(CD-1 learning	2	**2.74** % ± 0.14	9.54 % ± 0.26
+ fine-tuning)	3	**2.71** % ± 0.14	**8.80** % ± 0.25
	4	**2.72** % ± 0.14	**8.83** % ± 0.24

Table 3: Classification performance on MNIST-small and MNIST-rotation of different networks for different strategies to initialize parameters, and different depths (number of layers).

Table 3 presents the classification performance obtained by the different deep networks with up to 4 hidden layers on MNIST-small and MNIST-rotation. The hyperparameters of each layer were *separately* selected with the validation set for all hidden layers, using the following greedy strategy: for a network with l hidden layers, only the hyperparameters for the top layer were optimized, the hyperparameters for the layers below being set to those of the best $l-1$ layers deep network according to the validation performance. We settled for this strategy because of the exponential number of possible configurations of hyperparameters. For standard neural networks, we also tested several random initializations of the weights. For SRBM as well as SAA networks, we tuned the unsupervised learning rates and the number of updates. For MNIST-small, we used hidden layers of 500 neurons, since the experiments by Hinton (2006) suggest that it is an appropriate choice. As for MNIST-rotation, the size of each hidden layer had to be validated separately for each layer, and we tested values among 500, 1000, 2000 and 4000.

Table 3 show that there is indeed an optimal number of hidden layers for the deep networks, and that this optimum tends to be larger when unsupervised greedy layer-wise learning is used. For the MNIST-small data set (Table 3), the gain in performance between 2 and 3 hidden layers for SRBM and SAA networks is not statistically significant. However, for the MNIST-rotation data set, the improvement from 2 to 3 hidden layers is clear. This observation is consistent with the increased complexity of the input distribution and classification problem of MNIST-rotation, which should require a more complex model. The improvement remains significant when fixing the network's hidden layers to the same size as in the experiments on MNIST-small, as showed in the results of Table 4 where the number of units per hidden layer was set to 1000. We also compared the performance of shallow and deep SRBM networks with roughly the same number of parameters. With a shallow SRBM network, the best classification error achieved was 10.47%, with 4000 hidden units (around 3.2×10^6 free parameters). With a 3-layers deep SRBM network, we reached 9.38%

Network			MNIST-rotation classif. test error
Type	Depth	Layers width	
SRBM network (CD-1 learning + fine-tuning)	1	1k	12.44 % ± 0.29
	2	1k, 1k	9.98 % ± 0.26
	3	1k, 1k, 1k	**9.38** % ± 0.25

Table 4: Classification performance on MNIST-rotation of different networks for different strategies to initialize parameters, and different depths (number of layers). All hidden layers have 1000 units.

classification error with 1000 units in each layer (around 2.8×10^6 parameters): better generalization was achieved with deeper nets having less parameters.

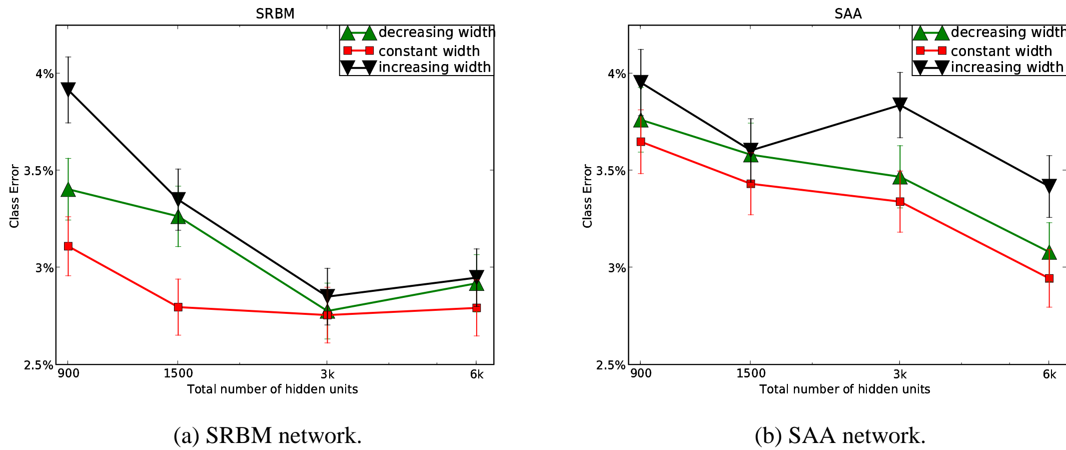
6.2.2 TYPE OF NETWORK ARCHITECTURE

The model selection procedure of Section 6.2.1 works well, but is rather expensive. Every time one wants to train a 4 hidden layer network, networks with 1, 2 and 3 hidden layers effectively have to be trained as well, in order to determine appropriate hyperparameters for the lower hidden layers. These networks can't even be trained in parallel, adding to the computational burden of this model selection procedure. Moreover, the optimal hidden layer size for a 1-hidden layer network could be much bigger than necessary for a 4 hidden layer network, since a shallow network cannot rely on other upper layers to increase its capacity.

Let us consider the situation where the number of hidden layers of a deep network has already been chosen and good sizes of the different layers must be found. Because the space of such possible choices is exponential in the number of layers, we consider here only three general cases where, as the layer index increases, their sizes either increases (doubles), decreases (halves) or does not change. We conducted experiments for all three cases and varied the total number of hidden neurons in the network. The same hyperparameters as in the experiment of Table 3 had to be selected for each network topologies, however a single unsupervised learning rate and number of updates were chosen for all layers.[5]

We observe in Figures 9 and 10 that the architecture that most often is among the best performing ones across the different sizes of network is the one with equal sizes of hidden layers. It should be noted that this might be a consequence of using the same unsupervised learning hyperparameters for each layer. It might be that the size of a hidden layer has a significant influence on the optimum value for these hyperparameters, and that tying them for all hidden layers induces a bias towards networks with equally-sized hidden layers. However, having untied hyperparameters would make model selection too computationally demanding. Actually, even with tied unsupervised learning hyperparameters, the model selection problem is already complex enough (and prone to overfitting with small data sets), as is indicated by the differences in the validation and test classification errors of Table 3.

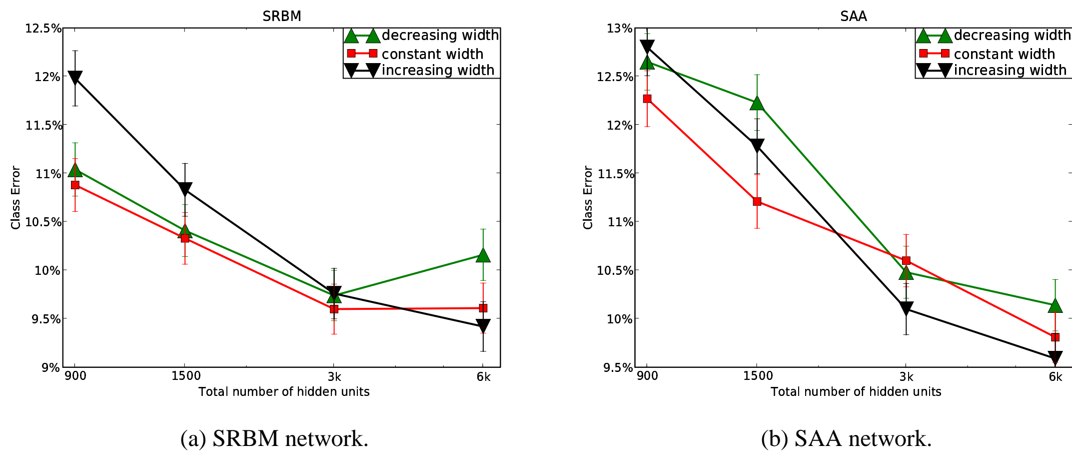
5. We imposed this restriction because of the large number of experiments that would otherwise had been required.



(a) SRBM network.

(b) SAA network.

Figure 9: Classification performance on MNIST-small of 3-layer deep networks for three kinds of architectures, as a function of the total number of hidden units. The three architectures have increasing / constant / decreasing layer sizes from the bottom to the top layers. Error-bars represent 95% confidence intervals.



(a) SRBM network.

(b) SAA network.

Figure 10: Classification performance on MNIST-rotation of 3-layer deep networks for three kinds of architectures. Same conventions as in Figure 9.

7. Continuous-Valued Inputs

In this section, we wish to emphasize the importance of adapting the unsupervised learning algorithms to the nature of the inputs. We will focus on the SRBM network because they rely on RBMs, which are less simple to work with and adapt to the sorts of visible data we want to model. With the binary units introduced for RBMs and DBNs in Hinton et al. (2006) one can "cheat" and handle continuous-valued inputs by scaling them to the $[0,1]$ interval and considering each input continuous value as the probability for a binary random variable to take the value 1. This has worked well for pixel gray levels, but it may be inappropriate for other kinds of input variables. Previous work on continuous-valued input in RBMs include Chen and Murray (2003), in which noise is added to sigmoidal units, and the RBM forms a special form of diffusion network (Movellan et al., 2002). Welling et al. (2005) also show how to derive RBMs from arbitrary choices of exponential distributions for the visible and hidden layers of an RBM. We show here simple extensions of the RBM framework in which only the energy function and the allowed range of values are changed. As can be seen in Figures 11 and 12 and in the experiment of Section 7.3, such extensions have a very significant impact on nature of the solution learned for the RBM's weights and hence on the initialization of a deep network and its performance.

7.1 Linear Energy: Exponential or Truncated Exponential

Consider a unit with value x_k in an RBM, connected to units $\mathbf{h}$ of the layer above. $p(x_k|\mathbf{h})$ can be obtained by considering the terms in the energy function that contain x_k. These terms can be grouped in $x_k(\mathbf{W}_{\cdot k}^{\mathsf{T}}\mathbf{h}+c_k)$ when the energy function is linear in x_k (as in Equation 7, appendix B), where $\mathbf{W}_{\cdot k}$ is the k-th column of $\mathbf{W}$. If we allow x_k to take any value in interval I, the conditional density of x_k becomes

$$p(x_k|\mathbf{h}) = \frac{e^{x_k(\mathbf{W}_{\cdot j}^{\mathsf{T}}\mathbf{h}+c_k)}\mathbf{1}_{x_k\in I}}{\int_v e^{v(\mathbf{W}_{\cdot j}^{\mathsf{T}}\mathbf{h}+c_k)}\mathbf{1}_{v\in I}dv} \, .$$

When $I=[0,\infty)$, this is an exponential density with parameter $a(\mathbf{h}) = \mathbf{W}_{\cdot j}^{\mathsf{T}}\mathbf{h}+c_k$, and the normalizing integral, equal to $\frac{-1}{a(\mathbf{h})}$, only exists if $a(\mathbf{h}) < 0 \; \forall \mathbf{h}$. Computing the density, the expected value ($\frac{-1}{a(\mathbf{h})}$) and sampling would all be easy, but since the density does not always exist it seems more appropriate to let I be a closed interval, yielding a *truncated exponential* density. For simplicity we consider the case $I=[0,1]$ here, for which the normalizing integral, which always exists, is

$$\frac{e^{-a(\mathbf{h})}-1}{a(\mathbf{h})} \, .$$

The conditional expectation of x_k given $\mathbf{h}$ is interesting because it has a sigmoidal-like saturating and monotone non-linearity:

$$E\left[x_k|\mathbf{h}\right] = \frac{1}{1-e^{-a(\mathbf{h})}} - \frac{1}{a(\mathbf{h})} \, .$$

Note that $E\left[x_k|\mathbf{h}\right]$ does not explode for $a(\mathbf{h})$ near 0, but is instead smooth and in the interval $[0,1]$. A sample from the truncated exponential is easily obtained from a uniform sample U, using the inverse cumulative F^{-1} of the conditional density $p(x_k|\mathbf{h})$:

$$F^{-1}(U) = \frac{\log(1-U\times(1-e^{a(\mathbf{h})}))}{a(\mathbf{h})} \, .$$

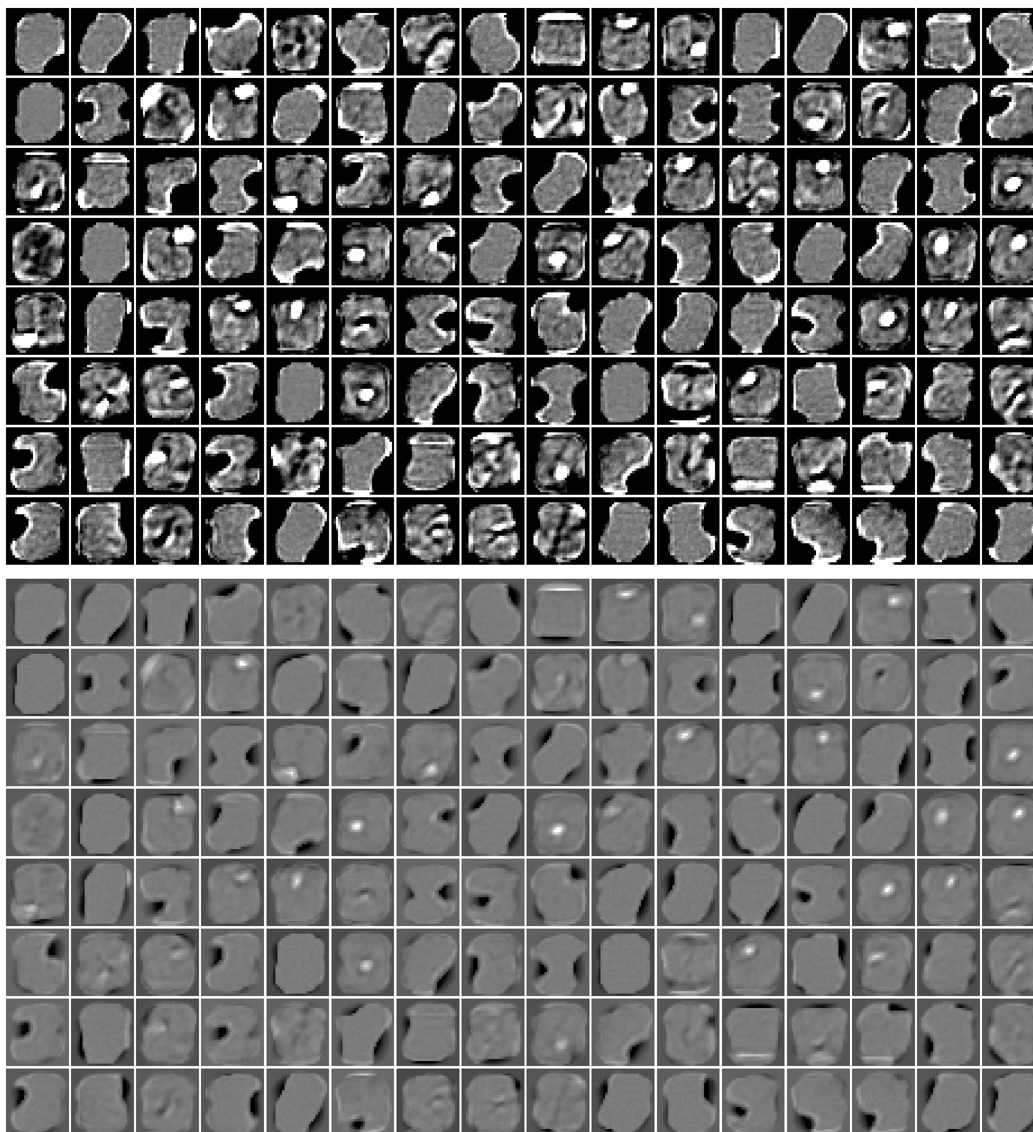


Figure 11: Input weights of a random subset of the hidden units, learned by an RBM with truncated exponential visible units, when trained on samples from the MNIST data set. The top and bottom images correspond to the same filters but with different color scale. On the top, the display setup is the same as for Figures 6 and 7 and, on the bottom, a black and white pixel correspond to weights smaller than -30 and larger than 30 respectively.

The contrastive divergence updates have the same form as for binary units of Equation 11, since the updates only depend on the derivative of the energy with respect to the parameters. Only sampling is changed, according to the unit's conditional density. Figure 11 shows the filters learned by an RBM with truncated exponential visible units, when trained on MNIST samples. Note how these are strikingly different from those obtained with binomial units.

7.2 Quadratic Energy: Gaussian Units

To obtain Gaussian-distributed units, one only needs to add quadratic terms to the energy. Adding $\sum_k d_k^2 x_k^2$ gives rise to a diagonal covariance matrix between units of the same layer, where x_k is the continuous value of a Gaussian unit and d_k^2 is a positive parameter that is equal to the inverse of the variance of x_k. In this case the variance is unconditional, whereas the mean depends on the inputs of the unit: for a visible unit x_k with hidden layer $\mathbf{h}$ and inverse variance d_k^2,

$$E\left[x_k|\mathbf{h}\right] = \frac{a(\mathbf{h})}{2d_k^2} \; .$$

The contrastive divergence updates are easily obtained by computing the derivative of the energy with respect to the parameters. For the parameters in the linear terms of the energy function $\mathbf{b}$, $\mathbf{c}$ and $\mathbf{W}$, the derivatives have the same form as for the case of binary units. For quadratic parameter $d_k > 0$, the derivative is simply $2d_k x_k^2$. Figure 12 shows the filters learned by an RBM with Gaussian visible units, when trained on MNIST samples.

Gaussian units were previously used as hidden units of an RBM (with multinomial inputs) applied to an information retrieval task (Welling et al., 2005). That same paper also shows how to generalize RBMs to units whose marginal distribution is from any member of the exponential family.

7.3 Impact on Classification Performance

In order to assess the impact of the choice for the visible layer distribution on the ultimate performance of an SRBM network, we trained and compared different deep networks whose first level RBM had binary, truncated exponential or Gaussian input units. These networks all had 3 hidden layers, with 2000 hidden units for each of these layers. The hyperparameters that were optimized are the unsupervised learning rates and number of updates as well as the fine-tuning learning rate. Because the assumption of binary inputs is not unreasonable for the MNIST images, we conducted this experiment on a modified and more challenging version of the data set where the background contains patches of images downloaded from the Internet. Samples from this data set are shown in Figure 13. This data set is part of a benchmark[6] designed by Larochelle et al. (2007). The results are given in Table 5, where we can see that the choice of the input distribution has a significant impact on the classification performance of the deep network. As a comparison, a support vector machine with Gaussian kernel achieves 22.61% error on this data set (Larochelle et al., 2007). Other experimental results with truncated exponential and Gaussian input units are found in Bengio et al. (2007).

8. Generating vs Encoding

Though the SRBM and SAA networks are similar in their motivation, there is a fundamental difference in the type of unsupervised learning used during training. Indeed, the RBM is based on the learning algorithm of a *generative model*, which is trained to be able to generate data similar to those found in the training set. On the other hand, the autoassociator is based on the learning algorithm of an *encoding model* which tries to learn a new representation or code from which the input can be reconstructed without too much loss of information.

6. The benchmark's data sets can be downloaded from `http://www.iro.umontreal.ca/~lisa/icml2007`.

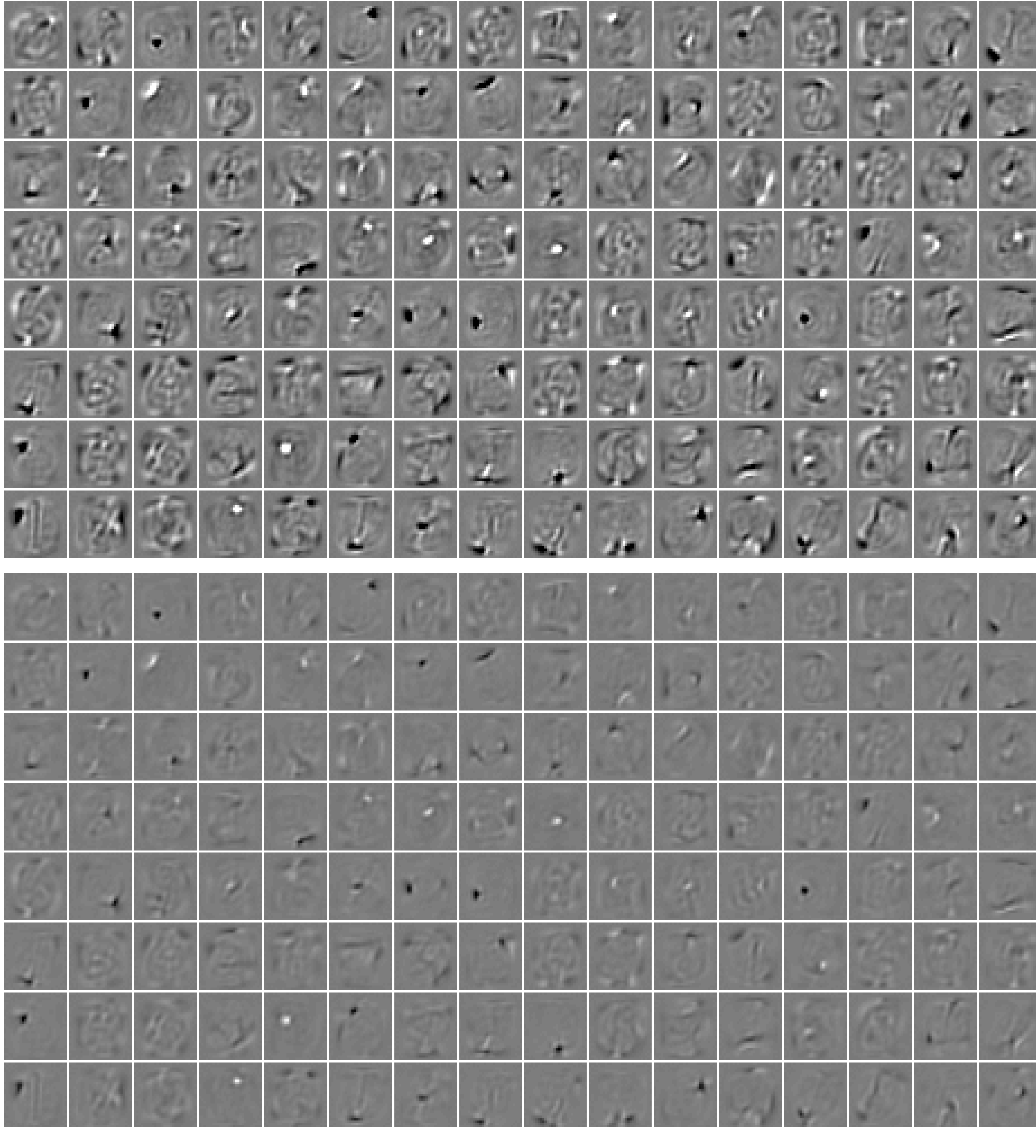


Figure 12: Input weights of a random subset of the hidden units, learned by an RBM with Gaussian visible units, when trained on samples from the MNIST data set. The top and bottom images correspond to the same filters but with different color scale. On top, the display setup is the same as for Figures 6 and 7 and, on the bottom, a black and white pixel correspond to weights smaller than -10 and larger than 10 respectively.

Figure 13: Samples from the modified MNIST digit recognition data set with a background containing image patches. Here, a black pixel corresponds to an input value of 0 and a white pixel corresponds to 1 (the inputs are scaled between 0 and 1).

SRBM input type	Train.	Valid.	Test
Bernoulli	10.50%	18.10%	20.29%
Gaussian	0%	20.50%	21.36%
Truncated exponential	0%	14.30%	14.34%

Table 5: Classification error on MNIST with background containing patches of images (see Figure 13) on the training, validation, and test sets, for different distributions of the input layer for the bottom RBM. The best hyperparameters were selected according to the validation error.

It is not clear which of the two approaches (generating or encoding) is the most appropriate. The advantage of a generative model is that the assumptions that are made are usually clear. However, it is possible that the problem it is trying to solve is harder than it needs to be, since ultimately we are only interested in coming up with good representations or features of the input. For instance, if one is interested in finding appropriate clusters in a very high dimensional space, using a mixture of Gaussians with full covariance matrix can quickly become too computationally intensive, whereas using the simple k-means algorithm might do a good enough job. As for encoding models, they do not require to be interpretable as a generative model and they can be more flexible because any parametric or non-parametric form can be chosen for the encoder and decoder, as long as they are differentiable.

Another interesting connection between reconstruction error in autoassociators and CD in RBMs was mentioned earlier: the reconstruction error can be seen as an estimator of the log-likelihood gradient of the RBM which has more bias but less variance than the CD update rule (Bengio and Delalleau, 2007). In that paper it is shown how to write the RBM log-likelihood gradient as a series expansion where each term is associated with a sample of the contrastive divergence Gibbs chain. Because the terms become smaller and converge to zero, this justifies taking a truncation of the series as an estimator of the gradient. The reconstruction error gradient is a mean-field (i.e., biased) approximation of the first term, whereas CD-1 is a sampling (i.e., high-variance) approximation of the first two terms, and similarly CD-k involves the first $2k$ terms.

This suggests combining the reconstruction error and contrastive divergence for training RBMs. During unsupervised pre-training, we can use the updates given by both algorithms and combine them by associating a coefficient to each of them. This is actually equivalent to applying the updates one after the other but using different learning rates for both. We tested this idea in the MNIST data set split of Section 6.1, where we had to validate separately the learning rates for the RBM and the autoassociator updates. This combination improved on the results of the SRBM and the SAA networks, obtaining 1.02% and 1.09% on the validation and test set respectively. This improvement was confirmed in a more complete experiment on 6 other folds with mutually exclusive test sets of 10 000 examples, where the mixed gradient variant gave on average a statistically significant improvement of 0.1% on a SRBM network. One possible explanation for the improvement brought by this combination is that it uses a better trade-off between bias and variance in estimating the log-likelihood gradient.

Another deterministic alternative to CD is mean-field CD (MF-CD) of Welling and Hinton (2002), and is equivalent to the pseudocode code in Appendix B, with the statements $\mathbf{h}^0 \sim p(\mathbf{h}|\mathbf{x}^0)$ and $\mathbf{v}^1 \sim p(\mathbf{x}|\mathbf{h}^0)$ changed to $\mathbf{h}^0 \leftarrow \text{sigm}(\mathbf{b}+\mathbf{W}\mathbf{v}^0)$ and $\mathbf{v}^1 \leftarrow \text{sigm}(\mathbf{c}+\mathbf{W}^\top\mathbf{h}^0)$ respectively. MF-CD can be used to test another way to change the bias/variance trade-off, either as a gradient estimator alone, or by combining it to the CD-1 gradient estimate (in the same way we combined the autoassociator gradient with CD-1, previous paragraph). On the MNIST split of Section 6.1, SRBM networks with MF-CD and combined CD-1/MF-CD[7] achieved 1.26% and 1.17% on the test set respectively. The improvement brought by combining MF-CD with CD-1 was not found to be statistically significant, based on similar experiments on the 6 other folds.

This suggests that something else than the bias/variance trade-off is at play in the improvements observed when combining CD-1 with the autoassociator gradient. A hypothesis that should be explored is that whereas there is no guarantee that an RBM will encode in its hidden representation all the information in the input vector, an autoassociator is trying to achieve this. In fact an RBM trained by maximum likelihood would be glad to completely ignore the inputs if these were independent of each other. Minimizing the reconstruction error would prevent this, and may be useful in the context where the representations are later used for supervised classification (which is the case here).

9. Continuous Training of all Layers of a Deep Network

The layer-wise training algorithm for networks of depth l actually has $l+1$ separate training phases: first the l phases for the unsupervised training of each layer, and then the final supervised fine-tuning phase to adjust all the parameters simultaneously. One element that we would like to dispense with

7. The weight of the CD-1 and MF-CD gradient estimates was considered as a hyperparameter.

is having to decide the number of unsupervised training iterations for each layer before starting the fine-tuning. One possibility is then to execute all phases simultaneously, that is, train all layers based on both their greedy unsupervised and global supervised gradients. The advantage is that we can now have a single stopping criterion (for the whole network). However, computation time is slightly greater, since we do more computations initially on the upper layers, which might be wasted before the lower layers converge to a decent representation, but time is saved on optimizing hyperparameters. When this continuous training variant is used on the MNIST data set with the same experimental setup as in Section 6.1, we reach 1.6% and 1.5% error on the test set respectively for the SRBM network and the SAA network, so unsupervised learning still brings better generalization in this setting. This variant may be more appealing for on-line training on very large data sets, where one would never cycle back on the training data.

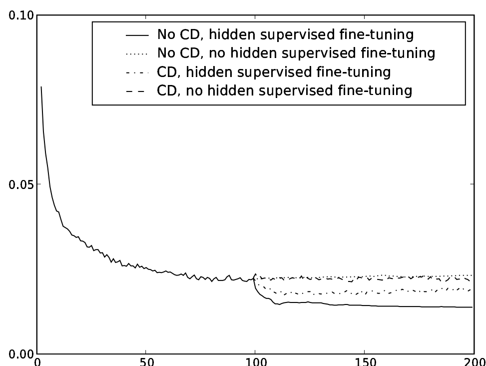
However, there seems to be a price to pay in terms of classification error, with this online variant. In order to investigate what could be the cause, we experimented with a 2-phase algorithm designed to shed some light on the contribution of different factors to this decrease. In the first phase, all layers of networks were simultaneously trained according to their unsupervised criterion *without* fine-tuning. The output layer is still trained according to the supervised criterion, however, unlike in Section 6.1, the gradient is not backpropagated into the rest of the network. This allows us to monitor the discriminative capacity of the top hidden layer. This first phase also enables us to verify whether the use of the supervised gradient too early during training explains the decrease in performance (recall the poor results obtained with purely supervised greedy layer-wise training). Then, in the second phase, 2 options were considered:

1. fine-tune the whole network according to the supervised criterion and stop layer-wise unsupervised learning;
2. fine-tune the whole network and maintain layer-wise unsupervised learning (as in the previous experiment).

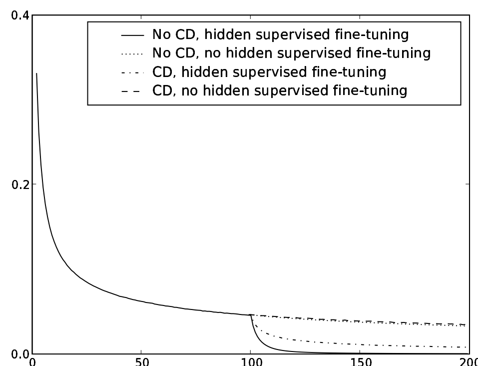
Figures 14(a) and 15(a) show examples of the progression of the test classification error for such an experiment with the SRBM and SAA networks respectively. As a baseline for the second phase, we also give the performance of the networks when unsupervised learning is stopped and only the parameters of the output layer are trained. These specific curves do not correspond to the best values of the hyperparameters, but are representative of the global picture we observed on several runs with different hyperparameter values.

We observe that the best option is to perform fine-tuning without layer-wise unsupervised learning, even when supervised learning is not introduced at the beginning. Also, though performing unsupervised and supervised learning at the same time outperforms unsupervised learning without fine-tuning, it appears to yield over-regularized networks, as indicated by the associated curves of the training negative log-likelihood of the target classes for both networks (see Figures 14(b) and 15(b)). Indeed, we see that by maintaining some unsupervised learning, the networks are not able to optimize as well their supervised training objective. From other runs with different learning rates, we have observed that this effect becomes less visible when the supervised learning rate gets larger, which reduces the relative importance of the unsupervised updates. But then the unsupervised updates usually bring significant instabilities in the learning process, making even the training cost oscillate.

Another interesting observation is that, when layer-wise unsupervised learning is performed, the classification error is less stable in an SRBM network than in an SAA network, as indicated by

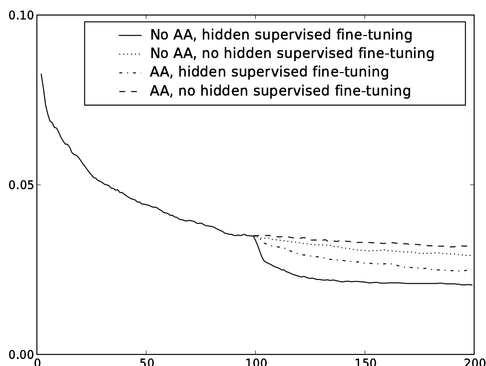


(a) SRBM network, test classification error curves

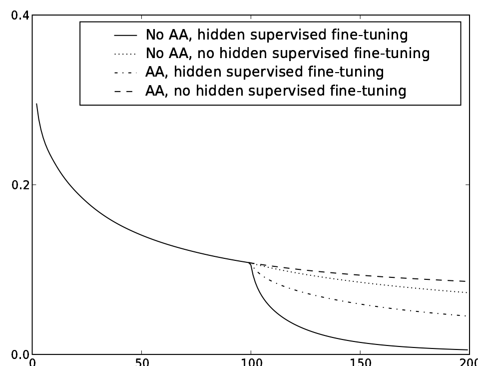


(b) SRBM network, train NLL error curves.

Figure 14: Example of learning curves of the 2-phase experiment of Section 9. During the first half of training, all hidden layers are trained according to CD and the output layer is trained according to the supervised objective, for all curves. In the second phase, all combinations of two possibilities are displayed: CD training is performed at all hidden layers ("CD") or not ("No CD"), and all hidden layers are fine-tuned according to the supervised objective ("hidden supervised fine-tuning") or not ("no hidden supervised fine-tuning").



(a) SAA network, test classification error curves



(b) SAA network, train NLL error curves.

Figure 15: Same as Figure 14, but with autoassociators ("AA") used for layer-wise unsupervised learning.

the dented learning curves in Figure 14(b), whereas the curves in Figure 15(b) are smoother. This may be related to the better performance of the SAA network (1.5%) versus the SRBM network (1.6%) when combining unsupervised and supervised gradients, in the experiment reported at the beginning of this section. Autoassociator learning might hence be more appropriate here, possibly because its training objective, that is, the discovery of a representation that preserves the information in the input, is more compatible with the supervised training objective, which asks that the network discovers a representation that is predictive of the input's class. This hypothesis is related to the one presented at the end of Section 8 regarding the apparent improvement brought by minimizing the reconstruction error in addition to CD-1 updates.

These experiments show that one can eliminate the multiple unsupervised phases: each layer can be pre-trained in a way that simply ignores what the layer above are doing. However, it appears that a final phase involving only supervised gradient yields the best performance. A plausible explanation of these results, and in particular the quick improvement when the unsupervised updates are removed, is that the unsupervised pre-training brings the parameters near a good solution for the supervised criterion, but far enough from that solution to yield a significantly higher classification error. Note that in a setting where there is little labeled data but a lot of unlabelled examples, the additional regularization introduced by maintaining some unsupervised learning might be beneficial (Salakhutdinov and Hinton, 2007b).

10. Conclusion

In this paper, we discussed in detail three principles for training deep neural networks, which are (1) pre-training one layer at a time in a greedy way (2) using unsupervised learning at each layer in a way that preserves information from the input and disentangles factors of variation and (3) fine-tuning the whole network with respect to the ultimate criterion of interest. We also presented experimental evidence that supports the claim that they are key ingredients for reaching good results. Moreover, we presented a series of experimental results that shed some light on many aspects of deep networks: confirming that the unsupervised procedure helps the optimization of the deep architecture, while initializing the parameters in a region near which a good solution of the supervised task can be found. Our experiments showed cases where greater depth clearly helps, but too much depth could be slightly detrimental. We found that CD-1 can be improved by combining it with the gradient of reconstruction error, and that this is not just due to the use of a lower-variance update. We showed that the choice of input distribution in RBMs could be important for continuous-valued input and yielded different types of filters at the first layer. Finally we studied variants more amenable to online learning in which we show that if different training phases can be combined, the best results were obtained with a final fine-tuning phase involving only the supervised gradient.

There are many questions and issues that remain to be addressed and that we intend to investigate in future work. As pointed out in Section 8, the most successful unsupervised learning approach seems to fall in between generative and encoding approaches. This raises questions about what are the properties of a learning algorithm that learns good representations for deep networks. Finding good answers to these questions would have a direct positive impact on the performance of deep networks. Finally, better model selection techniques that would permit to reduce the number of hyperparameters would be beneficial and will need to be developed for deep network learning algorithms to become easier to use.

Acknowledgments

The author are particularly grateful for the inspiration from and constructive discussions with Dan Popovici, Aaron Courville, Olivier Delalleau, James Bergstra, and Dumitru Erhan. The authors also want to thank the editor and reviewers for their helpful comments and suggestions. This research was performed thanks to funding from NSERC, MITACS, and the Canada Research Chairs.

Appendix A. Pseudocode for Greedy Layer-Wise Training Paradigm

Input: training set $\mathcal{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$, pre-training learning rate $\varepsilon_{\text{pre-train}}$ and fine-tuning learning rate $\varepsilon_{\text{fine-tune}}$

Initialize weights $\mathbf{W}^i_{jk} \sim U(-a^{-0.5}, a^{-0.5})$ with $a = \max(|\widehat{\mathbf{h}}^{i-1}|, |\widehat{\mathbf{h}}^{i}|)$ and set biases $\mathbf{b}^i$ to 0

% Pre-training phase
for $i \in \{1, \ldots, l\}$ **do**
  while Pre-training stopping criterion is not met **do**
    Pick input example $\mathbf{x}_t$ from training set
    $\widehat{\mathbf{h}}^0(\mathbf{x}_t) \leftarrow \mathbf{x}_t$
    for $j \in \{1, \ldots, i-1\}$ **do**
      $\mathbf{a}^j(\mathbf{x}_t) = \mathbf{b}^j + \mathbf{W}^j \widehat{\mathbf{h}}^{j-1}(\mathbf{x}_t)$
      $\widehat{\mathbf{h}}^j(\mathbf{x}_t) = \text{sigm}\left(\mathbf{a}^j(\mathbf{x}_t)\right)$
    end for
    Using $\widehat{\mathbf{h}}^{i-1}(\mathbf{x}_t)$ as input example, update weights $\mathbf{W}^i$ and biases $\mathbf{b}^{i-1}$, $\mathbf{b}^i$ with learning rate $\varepsilon_{\text{pre-train}}$ according to a layer-wise unsupervised criterion (see pseudocodes in appendices B and C)
  end while
end for

% Fine-tuning phase
while Fine-tuning stopping criterion is not met **do**
  Pick input example $(\mathbf{x}_t, y_t)$ from training set

  % Forward propagation
  $\widehat{\mathbf{h}}^0(\mathbf{x}_t) \leftarrow \mathbf{x}_t$
  for $i \in \{1, \ldots, l\}$ **do**
    $\mathbf{a}^i(\mathbf{x}_t) = \mathbf{b}^i + \mathbf{W}^i \widehat{\mathbf{h}}^{i-1}(\mathbf{x}_t)$
    $\widehat{\mathbf{h}}^i(\mathbf{x}_t) = \text{sigm}\left(\mathbf{a}^i(\mathbf{x}_t)\right)$
  end for
  $\mathbf{a}^{l+1}(\mathbf{x}_t) = \mathbf{b}^{l+1} + \mathbf{W}^{l+1} \widehat{\mathbf{h}}^{l}(\mathbf{x}_t)$
  $\mathbf{o}(\mathbf{x}_t) = \widehat{\mathbf{h}}^{l+1}(\mathbf{x}_t) = \text{softmax}\left(\mathbf{a}^{l+1}(\mathbf{x}_t)\right)$

  % Backward gradient propagation and parameter update
  $\frac{\partial \log o_{y_t}(\mathbf{x}_t)}{\partial a_j^{l+1}(\mathbf{x}_t)} \leftarrow \mathbf{1}_{y_t=j} - o_j(\mathbf{x}_t)$ **for** $j \in \{1, \ldots, K\}$

$\mathbf{b}^{l+1} \leftarrow \mathbf{b}^{l+1} + \varepsilon_{\text{fine-tune}} \frac{\partial \log o_{y_t}(\mathbf{x}_t)}{\partial \mathbf{a}^{l+1}(\mathbf{x}_t)}$
$\mathbf{W}^{l+1} \leftarrow \mathbf{W}^{l+1} + \varepsilon_{\text{fine-tune}} \frac{\partial \log o_{y_t}(\mathbf{x}_t)}{\partial \mathbf{a}^{l+1}(\mathbf{x}_t)} \widehat{\mathbf{h}}^{l}(\mathbf{x}_t)^\top$
for $i \in \{1,\ldots,l\}$, in decreasing order **do**
$\frac{\partial \log o_{y_t}(\mathbf{x}_t)}{\partial \widehat{\mathbf{h}}^{i}(\mathbf{x}_t)} \leftarrow \left(\mathbf{W}^{i+1}\right)^\top \frac{\partial \log o_{y_t}(\mathbf{x}_t)}{\partial \mathbf{a}^{i+1}(\mathbf{x}_t)}$
$\frac{\partial \log o_{y_t}(\mathbf{x}_t)}{\partial a^{i}_{j}(\mathbf{x}_t)} \leftarrow \frac{\partial \log o_{y_t}(\mathbf{x}_t)}{\partial \widehat{h}^{i}_{j}(\mathbf{x}_t)} \widehat{h}^{i}_{j}(\mathbf{x}_t)\left(1-\widehat{h}^{i}_{j}(\mathbf{x}_t)\right)$ **for** $j \in \{1,\ldots,|\widehat{\mathbf{h}}^{i}|\}$
$\mathbf{b}^{i} \leftarrow \mathbf{b}^{i} + \varepsilon_{\text{fine-tune}} \frac{\partial \log o_{y_t}(\mathbf{x}_t)}{\partial \mathbf{a}^{i}}$
$\mathbf{W}^{i} \leftarrow \mathbf{W}^{i} + \varepsilon_{\text{fine-tune}} \frac{\partial \log o_{y_t}(\mathbf{x}_t)}{\partial \mathbf{a}^{i}} \widehat{\mathbf{h}}^{i-1}(\mathbf{x}_t)^\top$
end for
end while

In the first step of the gradient computation, one has to be careful to compute the gradient of the cost with respect to $\mathbf{a}^{l+1}(\mathbf{x}_t)$ at once, in order not to lose numerical precision during the computation. In particular, computing $\frac{\partial \log o_{y_t}(\mathbf{x}_t)}{\partial \mathbf{o}(\mathbf{x}_t)}$ first, then $\frac{\partial \mathbf{o}(\mathbf{x}_t)}{\partial \mathbf{a}^{l+1}(\mathbf{x}_t)}$ and applying chain-rule, leads to numerical instability and sometimes parameter value explosion (NaN).

Appendix B. Restricted Boltzmann Machines and Deep Belief Networks

In this section, we give a brief overview of restricted Boltzmann machines and deep belief networks.

B.1 Restricted Boltzmann Machine

A restricted Boltzmann machine is an energy-based generative model defined over a visible layer $\mathbf{v}$ (sometimes called input) and a hidden layer $\mathbf{h}$ (sometimes called hidden factors or representation). Given an energy function energy$(\mathbf{v},\mathbf{h})$ on the whole set of visible and hidden units, the joint probability is given by

$$p(\mathbf{v},\mathbf{h}) = \frac{e^{-\text{energy}(\mathbf{v},\mathbf{h})}}{Z} \tag{6}$$

where Z ensures that $p(\mathbf{v},\mathbf{h})$ is a valid distribution and sums to one. See Figure 3 for an illustration of an RBM.

Typically we take $h_i \in \{0,1\}$, but other choices are possible. For now, we consider only binary units, that is, $v_i \in \{0,1\}$ (the continuous case will be discussed in Section 7), where the energy function has the form

$$\text{energy}(\mathbf{v},\mathbf{h}) = -\mathbf{h}^\top W\mathbf{v} - c^\top\mathbf{v} - b^\top\mathbf{h} = -\sum_k c_k v_k - \sum_j b_j h_j - \sum_{jk} W_{jk} v_k h_j\ . \tag{7}$$

When considering the marginal distribution over $\mathbf{v}$, we obtain a mixture distribution

$$p(\mathbf{v}) = \sum_{\mathbf{h}} p(\mathbf{v},\mathbf{h}) = \sum_{\mathbf{h}} p(\mathbf{v}|\mathbf{h})p(\mathbf{h})$$

with a number of parameters linear in the number of hidden units H, while having a number of components exponential in H. This is because $\mathbf{h}$ can take as many as 2^H possible values. The 2^H distributions $p(\mathbf{v}|\mathbf{h})$ will in general be different, but they are tied. Though computing exactly the marginal $p(\mathbf{v})$ for large values of H is impractical, a good estimator of the log-likelihood gradient

have been found with the contrastive divergence algorithm described in the next section. An important property of RBMs is that the posterior distribution over one layer given the other is tractable and fast to compute, as opposed to mixture models with very many components in general. Indeed one can show that

$$p(\mathbf{v}|\mathbf{h}) = \prod_k p(v_k|\mathbf{h}) \quad \text{where} \quad p(v_k = 1|\mathbf{h}) = \text{sigm}(c_k + \sum_j W_{jk} h_j)\ , \tag{8}$$

$$p(\mathbf{h}|\mathbf{v}) = \prod_j p(h_j|\mathbf{v}) \quad \text{where} \quad p(h_j = 1|\mathbf{v}) = \text{sigm}(b_j + \sum_k W_{jk} v_k)\ . \tag{9}$$

Because of the particular parametrization of RBMs, inference of the "hidden factors" $\mathbf{h}$ given the observed input vector $\mathbf{v}$ is very easy because those factors are conditionally independent given $\mathbf{v}$. On the other hand, unlike in many factor models (such as ICA Jutten and Herault, 1991; Comon, 1994; Bell and Sejnowski, 1995 and sigmoidal belief networks Dayan et al., 1995; Hinton et al., 1995; Saul et al., 1996), these factors are generally not marginally independent (when we integrate $\mathbf{v}$ out). Notice the similarity between Equations 9 and 1, which makes it possible to relate the weights and biases of an RBM with those of a deep neural network.

B.2 Learning in a Restricted Boltzmann Machine

To train an RBM, we would like to compute the gradient of the negative log-likelihood of the data with respect to the RBM's parameters. However, given an input example $\mathbf{v}_0$, the gradient with respect to a parameter θ in an energy-based model

$$\frac{\partial}{\partial\theta}(-\log p(\mathbf{v}_0)) = E_{p(\mathbf{h}|\mathbf{v}_0)}\left[\frac{\partial \text{energy}(\mathbf{v}_0, \mathbf{h})}{\partial\theta}\right] - E_{p(\mathbf{v},\mathbf{h})}\left[\frac{\partial \text{energy}(\mathbf{v}, \mathbf{h})}{\partial\theta}\right] \tag{10}$$

necessitates a sum over all possible assignments for $\mathbf{h}$ (first expectation of Equation 10) and another sum over all assignments for $\mathbf{v}$ and $\mathbf{h}$ (second expectation). The first expectation is not problematic in an RBM because the posterior $p(\mathbf{h}|\mathbf{v}_0)$ and $\frac{\partial \text{energy}(\mathbf{v}_0,\mathbf{h})}{\partial\theta}$ factorize. However, the second expectation requires a prohibitive exponential sum over the possible configurations for $\mathbf{v}$ or $\mathbf{h}$.

Fortunately, there exists an approximation for this gradient given by the contrastive divergence (CD) algorithm (Hinton, 2002), which has been shown to work well empirically (Carreira-Perpiñan and Hinton, 2005). There are two key elements in this approximation. First, consider that in order to estimate the second term of Equation 10, we could replace the expectation by a unique evaluation of the gradient $\frac{\partial \text{energy}(\mathbf{v},\mathbf{h})}{\partial\theta}$ at a particular pair of values $(\mathbf{v}, \mathbf{h})$. This pair should ideally be sampled from the distribution $p(\mathbf{v}, \mathbf{h})$, which would make the estimation of the gradient unbiased. However, sampling exactly from an RBM distribution is not as easy as in a directed graphical model. Instead, we have to rely on sampling methods such as Markov Chain Monte Carlo methods. For an RBM, we can use Gibbs sampling based on the conditional distributions of Equations 8 and 9, but this method can be costly if the Markov chain mixes slowly. So the second key idea is to run only a few iterations of Gibbs sampling and use the data sample $\mathbf{v}_0$ as the initial state for the chain at the visible layer. It turns out that applying only one iteration of the Markov chain works well in practice. This corresponds to the following sampling procedure:

$$\mathbf{v}_0 \overset{p(\mathbf{h}_0|\mathbf{v}_0)}{\longrightarrow} \mathbf{h}_0 \overset{p(\mathbf{v}_1|\mathbf{h}_0)}{\longrightarrow} \mathbf{v}_1 \overset{p(\mathbf{h}_1|\mathbf{x}_1)}{\longrightarrow} \mathbf{h}_1$$

where $\stackrel{p(\mathbf{h}_i|\mathbf{v}_i)}{\longrightarrow}$ and $\stackrel{p(\mathbf{v}_{i+1}|\mathbf{h}_i)}{\longrightarrow}$ represent the operations of sampling from $p(\mathbf{h}_i|\mathbf{v}_i)$ and $p(\mathbf{v}_{i+1}|\mathbf{h}_i)$ respectively. Estimation of the gradient using the above sampling procedure is noted CD-1, with CD-k referring to the contrastive divergence algorithm, performing k iterations of the Markov chain up to $\mathbf{v}_k$. Training with CD-k has been shown to empirically approximate well training with the exact log-likelihood gradient (Carreira-Perpiñan and Hinton, 2005). Furthermore, it can be shown that the CD-k update is an unbiased estimator of the truncation of a series expansion of the log-likelihood gradient (Bengio and Delalleau, 2007), where the truncated part of the series converges to 0 as k increases.

Now let us consider the estimation of the gradient on a weight W_{jk}. We have

$$\frac{\partial \text{energy}(\mathbf{v},\mathbf{h})}{\partial W_{jk}} = -h_j v_k$$

which means that the CD-1 estimate of the gradient becomes

$$-E_{p(\mathbf{h}|\mathbf{v}_0)}\left[h_j v_{0k}\right] + E_{p(\mathbf{h}|\mathbf{v}_1)}\left[h_{1j} v_{1k}\right] = -p(h_j|\mathbf{v}_0)v_{0k} + p(h_j|\mathbf{v}_1)v_{1k} \ . \tag{11}$$

This is similar to what was presented in Hinton et al. (2006) except that we clarify here that we take the expected value of $\mathbf{h}$ given $\mathbf{v}$ instead of averaging over samples. These estimators have the same expected value because

$$E_{p(\mathbf{v},\mathbf{h})}\left[h_j v_k\right] = E_{p(\mathbf{v})}\left[E_{p(\mathbf{h}|\mathbf{v})}\left[h_j v_k\right]\right] = E_{p(\mathbf{v})}\left[p(h_j|\mathbf{v})v_k\right] \ .$$

Using $p(\mathbf{h}|\mathbf{v}_k)$ instead of $\mathbf{h}_k$ is also what is found in the Matlab code distributed with Hinton and Salakhutdinov (2006). Note that it is still necessary to sample $\mathbf{h}_0 \sim p(\mathbf{h}|\mathbf{v}_0)$ in order to sample $\mathbf{v}_1$, but it is not necessary to sample $\mathbf{h}_1$. The above gradient estimator can then be used to perform stochastic gradient descent by iterating through all vectors $\mathbf{v}_0$ of the training set and performing a parameter update using that gradient estimator in an on-line fashion. Gradient estimators for the biases b_k and c_j can as easily be derived from Equation 10.

Notice also that, even if $\mathbf{v}_0$ is not binary, the formula for the CD-1 estimate of the gradient does not change and is still computed essentially in the same way: only the sampling procedure for $p(\mathbf{v}|\mathbf{h})$ changes (see Section 7 for more details about dealing with continuous-valued inputs). The CD-1 training update for a given training input is detailed by the pseudocode in the next section.

In our implementation of the greedy layer-wise initialization phase, we use the deterministic sigmoidal outputs of the previous level as training vector for the next level RBM. By interpreting these real-valued components as probabilities, learning such a distribution for binary inputs can be seen as a crude "mean-field" way of dealing with probabilistic binary inputs (instead of summing or sampling across input configurations).

B.3 Pseudocode for Contrastive Divergence (CD-1) Training Update

Input: training input $\mathbf{x}$, RBM weights $\mathbf{W}^i$ and biases $\mathbf{b}^{i-1}, \mathbf{b}^i$ and learning rate ε
Notation: $a \sim p(\cdot)$ means set a equal to a random sample from $p(\cdot)$

% Set RBM parameters
$\mathbf{W} \leftarrow \mathbf{W}^i$, $\mathbf{b} \leftarrow \mathbf{b}^i$, $\mathbf{c} \leftarrow \mathbf{b}^{i-1}$

% Positive phase
$\mathbf{v}^0 \leftarrow \mathbf{x}$
$\widehat{\mathbf{h}}^0 \leftarrow \text{sigm}(\mathbf{b} + \mathbf{W}\mathbf{v}^0)$

% Negative phase
$\mathbf{h}^0 \sim p(\mathbf{h}|\mathbf{v}^0)$ according to Equation 9
$\mathbf{v}^1 \sim p(\mathbf{v}|\mathbf{h}^0)$ according to Equation 8
$\widehat{\mathbf{h}}^1 \leftarrow \text{sigm}(\mathbf{b} + \mathbf{W}\mathbf{v}^1)$

% Update
$\mathbf{W}^i \leftarrow \mathbf{W}^i + \varepsilon \left(\widehat{\mathbf{h}}^0 \left(\mathbf{v}^0\right)^\top - \widehat{\mathbf{h}}^1 \left(\mathbf{v}^1\right)^\top\right)$
$\mathbf{b}^i \leftarrow \mathbf{b}^i + \varepsilon \left(\widehat{\mathbf{h}}^0 - \widehat{\mathbf{h}}^1\right)$
$\mathbf{b}^{i-1} \leftarrow \mathbf{b}^{i-1} + \varepsilon \left(\mathbf{v}^0 - \mathbf{v}^1\right)$

B.4 Deep Belief Network

We wish to make a quick remark on the distinction between the SRBM network and the more widely known deep belief network (DBN) (Hinton et al., 2006), which is not a feed-forward neural network but a multi-layer generative model. The SRBM network was initially derived (Hinton et al., 2006) from the DBN, for which stacking RBMs also provides a good initialization.

A DBN is a generative model with several layers of stochastic units. It actually corresponds to a sigmoid belief network (Neal, 1992) of $l-1$ hidden layers, where the prior over its top hidden layer $\mathbf{h}^{l-1}$ (second factor of Equation 12) is an RBM, which itself has a hidden layer $\mathbf{h}^l$. More precisely, it defines a distribution over an input layer $\mathbf{x}$ and l layers of binary stochastic units $\mathbf{h}^i$ as follows:

$$p(\mathbf{x}, \mathbf{h}^1, \ldots, \mathbf{h}^l) = \left(\prod_{i=1}^{l-1} p(\mathbf{h}^{i-1}|\mathbf{h}^i)\right) p(\mathbf{h}^{l-1}, \mathbf{h}^l) \tag{12}$$

where hidden units are conditionally independent given the units in the above layer

$$p(\mathbf{h}^{i-1}|\mathbf{h}^i) = \prod_k p(h_k^{i-1}|\mathbf{h}^i)\ .$$

To process binary values, Bernoulli layers can be used, which correspond to equations

$$p(h_k^{i-1} = 1|\mathbf{h}^i) = \text{sigm}\left(b_k^{i-1} + \sum_j W_{jk}^i h_j^i\right)$$

where $\mathbf{h}^0 = \mathbf{x}$ is the input. We also have

$$p(\mathbf{h}^{l-1}, \mathbf{h}^l) \propto e^{\sum_j c_j^{l-1} h_j^{l-1} + \sum_k b_k^l h_k^l + \sum_{jk} W_{jk}^l h_j^{l-1} h_k^l} \tag{13}$$

for the top RBM. Note that Equation 13 can be obtained from Equations 6 and 7, by naming $\mathbf{v}$ as $\mathbf{h}^{l-1}$, and $\mathbf{h}$ as $\mathbf{h}^l$.

We emphasize the distinction between $\mathbf{h}^i$ and $\widehat{\mathbf{h}}^i(\mathbf{x})$, where the former is a random variable and the latter is the representation of an input $\mathbf{x}$ at the i-th hidden layer of the network obtained from the repeated application of Equation 1.

To train such a generative model, Hinton et al. (2006) proposed the pre-training phase of the SRBM network. When all layers of a DBN have the same size, it was actually shown that this initialization improves a lower bound on the likelihood of the data as the DBN is made deeper. After this pre-training phase is over, Hinton et al. (2006) propose a variant of the Wake-Sleep algorithm for sigmoid belief networks (Hinton et al., 1995) to fine-tune the generative model.

By noticing the similarity between the process of approximating the posterior $p(\mathbf{h}^i|\mathbf{x})$ in a deep belief network and computing the hidden layer representation of an input $\mathbf{x}$ in a deep network, Hinton (2006) then proposed the use of the greedy layer-wise pre-training procedure for deep belief networks to initialize a deep feed-forward neural network, which corresponds to the SRBM network described in this paper.

Appendix C. Pseudocode of Autoassociator Training Update

Input: training input $\mathbf{x}$, autoassociator weights $\mathbf{W}^i$ and biases $\mathbf{b}^{i-1}, \mathbf{b}^i$ and learning rate ε

% Set autoassociator parameters
$\mathbf{W} \leftarrow \mathbf{W}^i$, $\mathbf{b} \leftarrow \mathbf{b}^i$, $\mathbf{c} \leftarrow \mathbf{b}^{i-1}$

% Forward propagation
$\mathbf{a}(\mathbf{x}) \leftarrow \mathbf{b} + \mathbf{W}\mathbf{x}$
$\mathbf{h}(\mathbf{x}) \leftarrow \mathrm{sigm}(\mathbf{a}(\mathbf{x}))$
$\widehat{\mathbf{a}}(\mathbf{x}) \leftarrow \mathbf{c} + \mathbf{W}^\mathsf{T}\mathbf{h}(\mathbf{x})$
$\widehat{\mathbf{x}} \leftarrow \mathrm{sigm}(\widehat{\mathbf{a}}(\mathbf{x}))$

% Backward gradient propagation
$\frac{\partial C(\widehat{\mathbf{x}},\mathbf{x})}{\partial \widehat{\mathbf{a}}(\mathbf{x})} \leftarrow \widehat{\mathbf{x}} - \mathbf{x}$
$\frac{\partial C(\widehat{\mathbf{x}},\mathbf{x})}{\partial \widehat{h}(\mathbf{x})} \leftarrow \mathbf{W} \frac{\partial C(\widehat{\mathbf{x}},\mathbf{x})}{\partial \widehat{\mathbf{a}}(\mathbf{x})}$
$\frac{\partial C(\widehat{\mathbf{x}},\mathbf{x})}{\partial a_j(\mathbf{x})} \leftarrow \frac{\partial C(\widehat{\mathbf{x}},\mathbf{x})}{\partial \widehat{h}_j(\mathbf{x})} \widehat{h}_j(\mathbf{x}) \left(1 - \widehat{h}_j(\mathbf{x})\right)$ **for** $j \in \{1, \ldots, |\widehat{\mathbf{h}}(\mathbf{x})|\}$

% Update
$\mathbf{W}^i \leftarrow \mathbf{W}^i - \varepsilon \left(\frac{\partial C(\widehat{\mathbf{x}},\mathbf{x})}{\partial \mathbf{a}(\mathbf{x})} \mathbf{x}^\mathsf{T} + \widehat{\mathbf{h}}(\mathbf{x}) \frac{\partial C(\widehat{\mathbf{x}},\mathbf{x})}{\partial \widehat{\mathbf{a}}(\mathbf{x})}^\mathsf{T} \right)$
$\mathbf{b}^i \leftarrow \mathbf{b}^i - \varepsilon \frac{\partial C(\widehat{\mathbf{x}},\mathbf{x})}{\partial \mathbf{a}(\mathbf{x})}$
$\mathbf{b}^{i-1} \leftarrow \mathbf{b}^{i-1} - \varepsilon \frac{\partial C(\widehat{\mathbf{x}},\mathbf{x})}{\partial \widehat{\mathbf{a}}(\mathbf{x})}$

References

Rie Kubota Ando and Tong Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6:1817–1853, 2005.

Peter Auer, Mark Herbster, and Manfred K. Warmuth. Exponentially many local minima for single neurons. In M. Mozer, D. S. Touretzky, and M. Perrone, editors, *Advances in Neural Information Processing System 8*, pages 315–322. MIT Press, Cambridge, MA, 1996.

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

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

Yoshua Bengio. Learning deep architectures for AI. Technical Report 1312, Université de Montréal, dept. IRO, 2007.

Yoshua Bengio and Olivier Delalleau. Justifying and generalizing contrastive divergence. Technical Report 1311, Dept. IRO, Université de Montréal, 2007.

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

Yoshua Bengio, Olivier Delalleau, and Nicolas Le Roux. The curse of highly variable functions for local kernel machines. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 107–114. MIT Press, Cambridge, MA, 2006.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 153–160. MIT Press, 2007.

Guillaume Bouchard and Bill Triggs. The tradeoff between generative and discriminative classifiers. In *IASC International Symposium on Computational Statistics (COMPSTAT)*, pages 721–728, Prague, August 2004. URL `http://lear.inrialpes.fr/pubs/2004/BT04`.

Miguel A. Carreira-Perpiñan and Geoffrey E. Hinton. On contrastive divergence learning. In Robert G. Cowell and Zoubin Ghahramani, editors, *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics, Jan 6-8, 2005, Savannah Hotel, Barbados*, pages 33–40. Society for Artificial Intelligence and Statistics, 2005.

Hsin Chen and Alan F. Murray. A continuous restricted Boltzmann machine with an implementable training algorithm. *IEE Proceedings of Vision, Image and Signal Processing*, 150(3):153–158, 2003.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML 2008)*, pages 160–167, 2008. URL `http://www.kyb.tuebingen.mpg.de/bs/people/weston/papers/unified\-nlp.pdf`.

Pierre Comon. Independent component analysis - a new concept? *Signal Processing*, 36:287–314, 1994.

Garrison W. Cottrell, Paul Munro, and David Zipser. Learning internal representations from gray-scale images: An example of extensional programming. In *Ninth Annual Conference of the Cognitive Science Society*, pages 462–473, Seattle 1987, 1987. Lawrence Erlbaum, Hillsdale.

Peter Dayan, Geoffrey Hinton, Radford Neal, and Rich Zemel. The Helmholtz machine. *Neural Computation*, 7:889–904, 1995.

David DeMers and Garrison W. Cottrell. Non-linear dimensionality reduction. In C.L. Giles, S.J. Hanson, and J.D. Cowan, editors, *Advances in Neural Information Processing Systems 5*, pages 580–587, San Mateo CA, 1993. Morgan Kaufmann.

Scott E. Fahlman and Christian Lebiere. The cascade-correlation learning architecture. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 524–532, Denver, CO, 1990. Morgan Kaufmann, San Mateo.

Ildiko E. Frank and Jerome H. Friedman. A statistical view of some chemometrics regression tools. *Technometrics*, 35(2):109–148, 1993.

Brendan J. Frey. *Graphical models for machine learning and digital communication*. MIT Press, 1998.

Kenji Fukumizu and Shun-ichi Amari. Local minima and plateaus in hierarchical structures of multilayer perceptrons. *Neural Networks*, 13(3):317–327, 2000.

Raia Hadsell, Ayse Erkan, Pierre Sermanet, Marco Scoffier, Urs Muller, and Yann LeCun. Deep belief net learning in a long-range vision system for autonomous off-road driving. In *Proc. Intelligent Robots and Systems (IROS'08)*, 2008. URL `http://www.cs.nyu.edu/~raia/docs/iros08-farod.pdf`.

Johan 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.

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

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

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

Geoffrey E. Hinton. To recognize shapes, first learn to generate images. Technical Report UTML TR 2006-003, University of Toronto, 2006.

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

Geoffrey E. Hinton, Peter Dayan, Brendan J. Frey, and Radford M. Neal. The wake-sleep algorithm for unsupervised neural networks. *Science*, 268:1558–1161, 1995.

Goeffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

Alex Holub and Pietro Perona. A discriminative framework for modelling object classes. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1*, pages 664–671, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2372-2. doi: http://dx.doi.org/10.1109/CVPR.2005.25.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

Tommi S. Jaakkola and David Haussler. Exploiting generative models in discriminative classifiers. In M.S. Kearns, S.A. Solla, and D.A. Cohn, editors, *Advances in Neural Information Processing Systems 11*. MIT Press, Cambridge, MA, 1999.

Tony Jebara. *Machine Learning: Discriminative and Generative (The Kluwer International Series in Engineering and Computer Science)*. Springer, December 2003. ISBN 1402076479. URL http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/1402076479.

Christian Jutten and Jeanny Herault. Blind separation of sources, part I: an adaptive algorithm based on neuromimetic architecture. *Signal Processing*, 24:1–10, 1991.

Hugo Larochelle and Yoshua Bengio. Classification using discriminative restricted boltzmann machines. In Andrew McCallum and Sam Roweis, editors, *Proceedings of the 25th Annual International Conference on Machine Learning (ICML 2008)*, pages 536–543. Omnipress, 2008.

Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In Zoubin Ghahramani, editor, *Twenty-fourth International Conference on Machine Learning (ICML 2007)*, pages 473–480. Omnipress, 2007. URL http://www.machinelearning.org/proceedings/icml2007/papers/331.pdf.

Julia A. Lasserre, Christopher M. Bishop, and Thomas P. Minka. Principled hybrids of generative and discriminative models. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 87–94, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2597-0. doi: http://dx.doi.org/10.1109/CVPR.2006.227.

Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

Régis Lengellé and Thierry Denoeux. Training MLPs layer by layer using an objective function for internal representations. *Neural Networks*, 9:83–97, 1996.

Javier R. Movellan, Paul Mineiro, and R. J. Williams. A monte-carlo EM approach for partially observable diffusion processes: theory and applications to neural networks. *Neural Computation*, 14:1501–1544, 2002.

Radford M. Neal. Connectionist learning of belief networks. *Artificial Intelligence*, 56:71–113, 1992.

Andrew Y. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *NIPS*, pages 841–848, 2001.

Simon Osindero and Geoffrey E. Hinton. Modeling image patches with a directed hierarchy of markov random field. In *Neural Information Processing Systems Conference (NIPS) 20*, 2008.

Marc'Aurelio Ranzato, Fu-Jie Huang, Y-Lan Boureau, and Yann LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Proc. Computer Vision and Pattern Recognition Conference (CVPR'07)*. IEEE Press, 2007a.

Marc'Aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann 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*. MIT Press, 2007b.

Marc'Aurelio Ranzato, Y-Lan Boureau, and Yann 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*. MIT Press, Cambridge, MA, 2008. URL `http://www.cs.nyu.edu/~ranzato/publications/ranzato-nips07.pdf`.

Ruslan Salakhutdinov and Geoffrey Hinton. Using deep belief nets to learn covariance kernels for gaussian processes. In J. C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*. MIT Press, Cambridge, MA, 2008. URL `http://www.csri.utoronto.ca/~hinton/absps/dbngp.pdf`.

Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. In *Proceedings of the 2007 Workshop on Information Retrieval and applications of Graphical Models (SIGIR 2007)*, Amsterdam, 2007a. Elsevier.

Ruslan Salakhutdinov and Geoffrey Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. In *Proceedings of AISTATS 2007*, San Juan, Porto Rico, 2007b. Omnipress.

Ruslan Salakhutdinov and Iain Murray. On the quantitative analysis of deep belief networks. In *Proceedings of the International Conference on Machine Learning*, volume 25, 2008.

Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 791–798, New York, NY, USA, 2007. ACM.

Lawrence K. Saul, Tommi Jaakkola, and Michael I. Jordan. Mean field theory for sigmoid belief networks. *Journal of Artificial Intelligence Research*, 4:61–76, 1996.

Eric Saund. Dimensionality-reduction using connectionist networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(3):304–314, 1989.

Paul 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.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In Andrew McCallum and Sam Roweis, editors, *Proceedings of the 25th Annual International Conference on Machine Learning (ICML 2008)*, pages 1096–1103. Omnipress, 2008. URL `http://icml2008.cs.helsinki.fi/papers/592.pdf`.

Ingo Wegener. *The Complexity of Boolean Functions*. John Wiley & Sons, 1987.

Max Welling and Geoffrey E. Hinton. A new learning algorithm for mean field boltzmann machines. In *ICANN '02: Proceedings of the International Conference on Artificial Neural Networks*, pages 351–357, London, UK, 2002. Springer-Verlag. ISBN 3-540-44074-7.

Max Welling, Michal Rosen-Zvi, and Geoffrey E. Hinton. Exponential family harmoniums with an application to information retrieval. In L.K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*. MIT Press, 2005.

Jason Weston, Frédéric Ratle, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML 2008)*, 2008. URL `http://www.kyb.tuebingen.mpg.de/bs/people/weston/papers/deep-embed.pdf`.

Andrew Yao. Separating the polynomial-time hierarchy by oracles. In *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science*, pages 1–10, 1985.