MIT-CSAIL-TR-2010-037 August 4, 2010

An Efficient Learning Procedure for Deep Boltzmann Machines

Ruslan Salakhutdinov and Geoffrey Hinton

An Efficient Learning Procedure for Deep Boltzmann Machines

Ruslan Salakhutdinov RSALAKHU@MIT.EDU
CSAIL
Massachusetts Institute of Technology
Cambridge, MA, USA

Geoffrey Hinton HINTON@CS.TORONTO.EDU
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada

Abstract

We present a new learning algorithm for Boltzmann Machines that contain many layers of hidden variables. Data-dependent statistics are estimated using a variational approximation that tends to focus on a single mode, and data-independent statistics are estimated using persistent Markov chains. The use of two quite different techniques for estimating the two types of statistic that enter into the gradient of the log likelihood makes it practical to learn Boltzmann Machines with multiple hidden layers and millions of parameters. The learning can be made more efficient by using a layer-by-layer "pre-training" phase that initializes the weights sensibly. The pre-training also allows the variational inference to be initialized sensibly with a single bottom-up pass. We present results on the MNIST and NORB datasets showing that Deep Boltzmann Machines learn very good generative models of hand-written digits and 3-D objects. We also show that the features discovered by Deep Boltzmann Machines are a very effective way to initialize the hidden layers of feed-forward neural nets which are then discriminatively fine-tuned.

1. A Brief History of Boltzmann Machine Learning

The original learning procedure for Boltzmann Machines (see section 2) makes use of the fact that the gradient of the log likelihood with respect to a connection weight has a very simple form: it is the difference of two pair-wise statistics (Hinton and Sejnowski (1983)). The first statistic is data-dependent and is the expectation that a pair of binary stochastic units are both on when a randomly selected training case is clamped on the "visible" units and the states of the "hidden" units are sampled from their posterior distribution. The second, data-independent statistic is the expectation that the two units are both on when the visible units are not constrained by data and the states of the visible and hidden units are sampled from the joint distribution defined by the parameters of the model.

Hinton and Sejnowski (1983) estimated the data-dependent statistics by clamping a training vector on the visible units, initializing the hidden units to random binary states, and using sequential Gibbs sampling of the hidden units (Geman and Geman (1984)) to approach the posterior distribution. They estimated the data-independent statistics in the same way, but with the randomly initialized visible units included in the sequential Gibbs sampling. Inspired by Kirkpatrick et al.

(1983) they used simulated annealing from a high initial temperature to a final temperature of one to speed up convergence to the stationary distribution. They demonstrated that this was a feasible way of learning the weights in small networks, but even with the help of simulated annealing, this learning procedure was much too slow to be practical for learning large, multi-layer Boltzmann machines. Even for small networks, the learning rate must be very small to avoid an unexpected effect: the high variance in the difference of the two estimated statistics has a tendency to drive the parameters to regions where each hidden unit is almost always on or almost always off. These regions act as attractors because the variance in the gradient estimate is lower in these regions, so the weights change much more slowly.

Neal (1992) improved the learning procedure by using persistent Markov chains. To estimate the data-dependent statistics, the Markov chain for each training case is initialized at its previous state for that training case and then just run for a few steps. Similarly, for the data-independent statistics, a number of Markov chains are run for a few steps from their previous states. If the weights have only changed slightly, the chains will already be close to their stationary distributions and a few iterations will suffice to keep them close. In the limit of very small learning rates, therefore, the data-dependent and data-independent statistics will be almost unbiased. Neal did not explicitly use simulated annealing, but the persistent Markov chains implement it implicitly, provided that the weights have small initial values. Early in the learning the chains mix rapidly because the weights are small. As the weights grow, the chains should remain near their stationary distributions in much the same way as simulated annealing should track the stationary distribution as the inverse temperature increases.

Neal (1992) showed that persistent Markov chains work quite well for training a Boltzmann Machine on a fairly small dataset. For large datasets, however, it is much more efficient to update the weights after a small "mini-batch" of training examples, so by the time a training example is revisited, the weights may have changed by a lot and the stored state of the Markov chain for that training case may be far from equilibrium. Also, once the weights become large, the Markov chains used for estimating the data-independent statistics may have a very slow mixing rate since they typically need to sample from a highly multimodal distribution in which widely separated modes have very similar probabilities but the vast majority of the joint states are extremely improbable. This suggests that the learning rates might need to be impractically small for the persistent chains to remain close to their stationary distributions with only a few state updates per weight update. Fortunately, the asymptotic analysis is almost completely irrelevant: there is a subtle reason, explained later in this section, why the learning works well with a learning rate that is much larger than the obvious asymptotic analysis would allow.

In an attempt to reduce the time required by the sampling process, Peterson and Anderson Peterson and Anderson (1987) replaced Gibbs sampling with a simple mean field method that approximates a stationary distribution by replacing stochastic binary values with deterministic real-valued probabilities. More sophisticated deterministic approximation methods were investigated by Galland (1991) and Kappen and Rodriguez (1998), but none of these approximations worked very well for learning for reasons that were not well understood at the time.

It is now well-known that in *directed* graphical models learning typically works quite well when the statistics from the true posterior distribution that are required for exact maximum likelihood learning are replaced by statistics from a simpler approximating distribution, such as a simple mean-field distribution (Zemel (1993); Hinton and Zemel (1994); Neal and Hinton (1998); Jordan et al. (1999)). The reason learning still works is that it follows the gradient of a variational bound (see section 3.1). This bound consists of the log probability that the model assigns to the training data penalized by the sum, over all training cases, of the Kullback-Leibler divergence between the approximating posterior and the true posterior over the hidden variables. Following the gradient of the bound tends to minimize this penalty term thus making the true posterior of the model similar to the approximating distribution.

An undirected graphical model, such as a Boltzmann Machine, has an additional, data-independent term in the maximum likelihood gradient. This term is the derivative of the log partition function and, unlike the data-dependent term, it has a negative sign. This means that if a variational approximation is used to estimate the data-independent statistics, the resulting gradient will tend to change the parameters to make the approximation worse. This probably explains the lack of success in using variational approximations for learning Boltzmann Machines.

The first efficient learning procedure for large-scale Boltzmann machines used an extremely limited architecture, first proposed in Smolensky (1986), that was designed to make inference easy. A Restricted Boltzmann Machine (RBM) has a layer of visible units and a layer of hidden units with no connections between the hidden units. The lack of connections between hidden units eliminates many of the computational properties that make general Boltzmann Machines interesting, but it makes it easy to compute the data-dependent statistics exactly, because the hidden units are independent given a data-vector. If connections between visible units are also prohibited, the data-independent statistics can be estimated by starting Markov chains at hidden states that were inferred from training vectors, and alternating between updating all of the visible units in parallel and updating all of the hidden units in parallel (Hinton (2002)). It is hard to compute how many alternations (half-steps) of these Markov chains are needed to approach the stationary distribution and it is also hard to know how close this approach must be for learning to make progress towards a better model. It is tempting to infer that, if the learning worked, the Markov chains used to estimate the data-independent statistics must have been close to equilibrium, but it turns out that this is quite wrong.

Empirically, learning usually works quite well if the alternating Gibbs sampling is run for only one full step starting from the sampled binary states of the hidden units inferred from a data-vector (Hinton (2002)). This gives very biased estimates of the data-independent statistics, but it greatly reduces the variance in the estimated difference between data-dependent and data-independent statistics (Williams and Agakov (2002)), especially when using mini-batch learning on large datasets. Much of the sampling error in the data-dependent statistics caused by using a small mini-batch is eliminated because the estimate of the data-independent statistics suffers from a very similar sampling error. The reduced variance allows a much higher learning rate. Instead of viewing this learning procedure as a gross approximation to maximum likelihood learning, it can be viewed as a much better approximation to minimizing the difference of two divergences (Hinton (2002)) and so it is called Contrastive Divergence (CD) learning. The quality of the learned model can be improved by using more full steps of alternating Gibbs sampling as the weights increase from their small initial values (Carreira-Perpignan and Hinton (2005)) and with this modification CD learning allows RBMs with millions of parameters to achieve state-of-the art performance[1] on a large collaborative filtering task (Salakhutdinov et al. (2007)).

1. The performance is comparable with the best other single models, such as probabilistic matrix factorization. By averaging many models it is possible to do better and the two systems with the best performance on Netflix both use multiple RBMs among the many models that are averaged.

The architectural limitations of RBMs can be overcome by using them as simple learning modules that are stacked to form a deep, multilayer network. After training each RBM, the activities of its hidden units, when they are being driven by data, are treated as training data for the next RBM (Hinton et al. (2006); Hinton and Salakhutdinov (2006)). However, if multiple layers are learned in this greedy, layer-by-layer way, the resulting composite model is *not* a multilayer Boltzmann Machine (Hinton et al. (2006)). It is a hybrid generative model called a "Deep Belief Net" that has undirected connections between its top two layers and downward directed connections between all adjacent lower layers.

In this paper we present a fairly efficient learning procedure for fully general Boltzmann Machines. To estimate the data-dependent statistics, we use mean-field variational inference and rely on the learning to make the true posterior distributions be close to the factorial distributions assumed by mean-field. To estimate the data-independent statistics we use a relatively small number of persistent Markov chains and rely on a subtle interaction between the learning and the Markov chains to allow a small number of slow mixing chains to sample quickly from a highly multimodal energy landscape. For both sets of statistics, the fact that the parameters are changing is essential for making the sampling methods work.

We then show how to make our learning procedure for general Boltzmann machines considerably more efficient for Deep Boltzmann Machines (DBMs) that have many hidden layers but no connections within each layer and no connections between non-adjacent layers. The weights of a DBM can be initialized by training a stack of RBMs, but with a modification that ensures that the resulting composite model is a Boltzmann Machine rather than a Deep Belief Net (DBN). This pre-training method has the added advantage that it provides a fast, bottom-up inference procedure for initializing the mean-field inference. We use the MNIST and NORB datasets to demonstrate that DBMs learn very good generative models of images of hand-written digits and 3-D objects. Although this paper is primarily about learning generative models, we also show that the weights learned by these models can be used to initialize deep feed-forward neural networks. These feed-forward networks can then be fine-tuned using backpropagation to give much better discriminative performance than randomly initialized networks.

2. Boltzmann Machines (BMs)

A Boltzmann Machine is a network of symmetrically coupled stochastic binary units. It contains a set of visible units $\mathbf{v} \in \{0,1\}^{\mathcal{V}}$, and a set of hidden units $\mathbf{h} \in \{0,1\}^{\mathcal{F}}$ (see Fig. 1, left panel), that learn to model higher-order correlations between the visible units. The energy of the state $\{\mathbf{v},\mathbf{h}\}$ is defined as:

$$E(\mathbf{v},\mathbf{h};\theta) \; = \; -\mathbf{v}^\top W\mathbf{h} - \frac{1}{2}\mathbf{v}^\top L\mathbf{v} - \frac{1}{2}\mathbf{h}^\top J\mathbf{h}, \tag{1}$$

where $\theta = \{W, L, J\}$ are the model parameters[2]: W, L, J represent visible-to-hidden, visible-to-visible, and hidden-to-hidden symmetric interaction terms. The diagonal elements of L and J are

2. We have omitted the bias terms for clarity of presentation. Biases are equivalent to weights on a connection to a unit whose state is fixed at 1, so their derivatives can be inferred from the derivatives for weights by simply setting the state of one of the two units to 1.

set to 0. The probability that the model assigns to a visible vector $\mathbf{v}$ is:

$$P(\mathbf{v};\theta) = \frac{P^*(\mathbf{v};\theta)}{\mathcal{Z}(\theta)} = \frac{1}{\mathcal{Z}(\theta)} \sum_{\mathbf{h}} \exp\left(-E(\mathbf{v},\mathbf{h};\theta)\right), \tag{2}$$

$$\mathcal{Z}(\theta) = \sum_{\mathbf{v}} \sum_{\mathbf{h}} \exp\left(-E(\mathbf{v},\mathbf{h};\theta)\right), \tag{3}$$

where P^* denotes unnormalized probability, and $\mathcal{Z}(\theta)$ is the partition function. The *conditional* distributions over hidden and visible units are given by:

$$p(h_j = 1|\mathbf{v}, \mathbf{h}_{-j}) = g\left(\sum_i W_{ij} v_i + \sum_{m \neq j} J_{jm} h_j\right), \tag{4}$$

$$p(v_i = 1|\mathbf{h}, \mathbf{v}_{-i}) = g\left(\sum_j W_{ij} h_j + \sum_{k \neq i} L_{ik} v_j\right), \tag{5}$$

where $g(x) = 1/(1 + \exp(-x))$ is the logistic function and $\mathbf{x}_{-i}$ denotes a vector $\mathbf{x}$ but with x_i omitted. The parameter updates, originally derived by Hinton and Sejnowski (1983), that are needed to perform gradient ascent in the log-likelihood can be obtained from Eq. 2:

$$\Delta W = \alpha\left(\mathrm{E}_{P_{\mathrm{data}}}[\mathbf{v}\mathbf{h}^\top] - \mathrm{E}_{P_{\mathrm{model}}}[\mathbf{v}\mathbf{h}^\top]\right),$$
$$\Delta L = \alpha\left(\mathrm{E}_{P_{\mathrm{data}}}[\mathbf{v}\mathbf{v}^\top] - \mathrm{E}_{P_{\mathrm{model}}}[\mathbf{v}\mathbf{v}^\top]\right),$$
$$\Delta J = \alpha\left(\mathrm{E}_{P_{\mathrm{data}}}[\mathbf{h}\mathbf{h}^\top] - \mathrm{E}_{P_{\mathrm{model}}}[\mathbf{h}\mathbf{h}^\top]\right),$$

where α is a learning rate. $\mathrm{E}_{P_{\mathrm{data}}}[\cdot]$, the data-dependent term, is an expectation with respect to the completed data distribution $P_{\mathrm{data}}(\mathbf{h},\mathbf{v};\theta) = P(\mathbf{h}|\mathbf{v};\theta)P_{\mathrm{data}}(\mathbf{v})$, with $P_{\mathrm{data}}(\mathbf{v}) = \frac{1}{N}\sum_n \delta(\mathbf{v}-\mathbf{v}^n)$ representing the empirical distribution, and $\mathrm{E}_{P_{\mathrm{model}}}[\cdot]$, the data-independent term, is an expectation with respect to the distribution defined by the model (Eq. 2).

Exact maximum likelihood learning in this model is intractable. The exact computation of the data-dependent expectation takes time that is exponential in the number of hidden units, whereas the exact computation of the model's expectation takes time that is exponential in the number of hidden and visible units.

Setting both J=0 and L=0 recovers the Restricted Boltzmann Machine (RBM) model (see Fig. 1, right panel). Setting only the hidden-to-hidden connections J=0 recovers a semi-restricted Boltzmann Machine (Osindero and Hinton (2008)) in which inferring the states of the hidden units given the visible states is still very easy but learning is more complicated because it is no longer feasible to infer the states of the visible units exactly when reconstructing the data from the hidden states.

2.1 A Stochastic Approximation Procedure for Estimating the Data-independent Statistics

Markov Chain Monte Carlo (MCMC) methods belonging to the general class of stochastic approximation algorithms of the Robbins–Monro type (Younes (1989); Robbins and Monro (1951)) can be used to approximate the data-independent statistics (Younes (2000); Neal (1992); Yuille (2004);

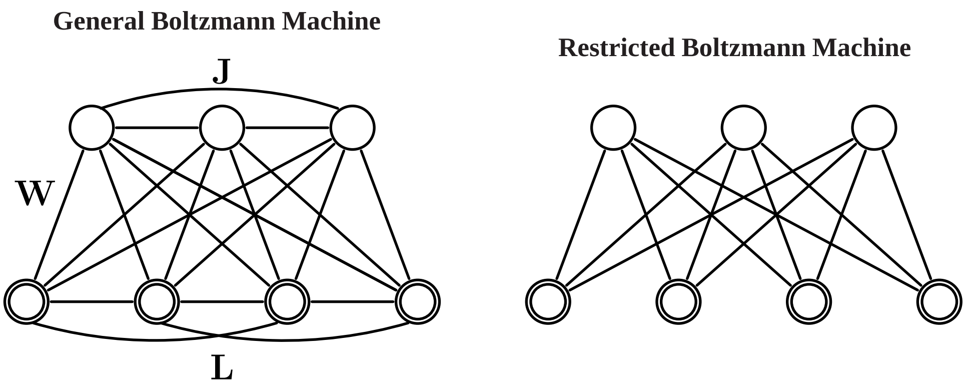

Figure 1: **Left:** A general Boltzmann Machine. The top layer represents a vector of stochastic binary "hidden" variables and the bottom layer represents a vector of stochastic binary "visible" variables. **Right:** A Restricted Boltzmann Machine with no hidden-to-hidden or visible-to-visible connections.

Tieleman (2008)). To be more precise, let us consider the following canonical form of the exponential family associated with the sufficient statistics vector Φ:

$$P(\mathbf{x};\theta) \;=\; \frac{1}{\mathcal{Z}(\theta)} \exp\left(\theta^\top \Phi(\mathbf{x})\right). \tag{6}$$

The derivative of the log-likelihood for an observation $\bar{\mathbf{x}}$ with respect to parameter vector θ takes the form:

$$\frac{\partial \log P(\bar{\mathbf{x}};\theta)}{\partial \theta} \;=\; \Phi(\bar{\mathbf{x}}) - \mathrm{E}_{P_{\mathrm{model}}}[\Phi(\mathbf{x})]. \tag{7}$$

The idea behind learning parameter vector θ using stochastic approximation is straightforward. Let θ^t and $\mathbf{x}^t$ be the current parameters and the state. Then $\mathbf{x}^t$ and θ^t are updated sequentially as follows:

- Given $\mathbf{x}^t$, a new state $\mathbf{x}^{t+1}$ is sampled from a transition operator $T_{\theta^t}(\mathbf{x}^{t+1} \leftarrow \mathbf{x}^t)$ that leaves $P(\cdot;\theta^t)$ invariant (e.g. a Gibbs sampler).

- A new parameter θ^{t+1} is then obtained by replacing the intractable data-independent statistics $\mathrm{E}_{P_{\mathrm{model}}}[\Phi(\mathbf{x})]$ by a point estimate $\Phi(\mathbf{x}^{t+1})$.

In practice, we typically maintain a set of M sample points $X^t = \{\tilde{\mathbf{x}}^{t,1}, ..., \tilde{\mathbf{x}}^{t,M}\}$, which we will often refer to as sample particles. In this case, the intractable data-independent statistics are replaced by the sample averages $1/M \sum_{m=1}^{M} \Phi(\tilde{\mathbf{x}}^{t+1,m})$. The procedure is summarized in Algorithm 1.

The standard proof of convergence of these algorithms relies on the following basic decomposition. First, the gradient of the log-likelihood function takes the form:

$$S(\theta) \;=\; \frac{\partial \log P(\bar{\mathbf{x}};\theta)}{\partial \theta} \;=\; \Phi(\bar{\mathbf{x}}) - \mathrm{E}_{P_{\mathrm{model}}}[\Phi(\mathbf{x})]. \tag{8}$$

Algorithm 1 Stochastic Approximation Algorithm.

```
1: Randomly initialize $\theta^0$ and $M$ sample particles $\{\tilde{\mathbf{x}}^{0,1}, ...., \tilde{\mathbf{x}}^{0,M}\}$.
2: for $t = 0 : T$ (number of iterations) do
3:     for $i = 1 : M$ (number of parallel Markov chains) do
4:         Sample $\tilde{\mathbf{x}}^{t+1,i}$ given $\tilde{\mathbf{x}}^{t,i}$ using transition operator $T_{\theta^t}(\tilde{\mathbf{x}}^{t+1,i} \leftarrow \tilde{\mathbf{x}}^{t,i})$.
5:     end for
6:     Update: $\theta^{t+1} = \theta^t + \alpha_t \left[\Phi(\bar{\mathbf{x}}) - \frac{1}{M}\sum_{m=1}^{M} \Phi(\tilde{\mathbf{x}}^{t+1,m})\right]$.
7:     Decrease $\alpha_t$.
8: end for
```

The parameter update rule then takes the following form:

$$\begin{aligned}\theta^{t+1} &= \theta^t + \alpha_t \left[\Phi(\bar{\mathbf{x}}) - \frac{1}{M}\sum_{m=1}^{M} \Phi(\tilde{\mathbf{x}}^{t+1,m})\right] \qquad (9)\\ &= \theta^t + \alpha_t S(\theta^t) + \alpha_t \left[\mathrm{E}_{P_{\text{model}}}[\Phi(\mathbf{x})] - \frac{1}{M}\sum_{m=1}^{M} \Phi(\tilde{\mathbf{x}}^{t+1,m})\right]\\ &= \theta^t + \alpha_t S(\theta^t) + \alpha_t \epsilon_{t+1}.\end{aligned}$$

The first term is the discretization of the ordinary differential equation $\dot{\theta} = S(\theta)$. The algorithm is therefore a perturbation of this discretization with the noise term ϵ_t. The proof then proceeds by showing that the noise term is not too large.

Precise sufficient conditions that ensure almost sure convergence to an asymptotically stable point of $\dot{\theta} = S(\theta)$ are given in Younes (1989, 2000); Yuille (2004). One necessary condition requires the learning rate to decrease with time, so that $\sum_{t=0}^{\infty} \alpha_t = \infty$ and $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$. This condition can, for example, be satisfied simply by setting $\alpha_t = 1/(t_0 + t)$. Other conditions ensure that the speed of convergence of the Markov chain, governed by the transition operator T_θ, does not decrease too fast as θ tends to infinity, and that the noise term ϵ_t in the update of Eq. 9 is bounded. Typically, in practice, the sequence $|\theta^t|$ is bounded, and the Markov chain, governed by the transition kernel T_θ, is ergodic. Together with the condition on the learning rate, this ensures almost sure convergence of the stochastic approximation algorithm to an asymptotically stable point of $\dot{\theta} = S(\theta)$.

Informally, the intuition behind why this procedure works is the following. As the learning rate becomes sufficiently small compared with the mixing rate of the Markov chain, this "persistent" chain will always stay very close to the stationary distribution, even if it is only run for a few MCMC steps per parameter update. Samples from the persistent chain will be highly correlated for successive parameter updates, but if the learning rate is sufficiently small, the chain will mix before the parameters have changed enough to significantly alter the value of the estimator.

The success of learning relatively small Boltzmann Machines (Neal (1992)) seemed to imply that the learning rate was sufficiently small to allow the chains to stay close to equilibrium as the parameters changed. Recently, however, this explanation has been called into question. After learning an RBM using persistent Markov chains for the data-independent statistics, we tried sampling from the RBM and discovered that even though the learning had produced a good model, the chains mixed extremely slowly. In fact, they mixed so slowly that the appropriate final learning rate, according to the explanation above, would have been smaller than the rate we actually used by several orders of magnitude. So why did the learning work?

Tieleman and Hinton (2009) argue that the fact that the parameters are being updated using the data-independent statistics gathered from the persistent chains means that the mixing rate of the chains *with their parameters fixed* is not what limits the maximum acceptable learning rate. Consider, for example, a persistent chain that is stuck in a deep local minimum of the energy surface. Assuming that this local minimum has very low probability under the posterior distributions that are used to estimate the data-dependent statistics, the effect of the learning will be to raise the energy of the local minimum. After a number of weight updates, the persistent chain will escape from the local minimum not because the chain has had time to mix but because the energy landscape has changed to make the local minimum much less deep. The learning causes the persistent chains to be repelled from whatever state they are currently in and this can cause slow mixing chains to move to other parts of the dynamic energy landscape much faster than would be predicted by the mixing rate with static parameters. Welling (2009) has independently reported a closely related phenomenon which he calls “herding”.

Recently, (Tieleman (2008); Salakhutdinov and Hinton (2009a); Salakhutdinov (2009); Desjardins et al. (2010)) have shown that this stochastic approximation algorithm, also termed Persistent Contrastive Divergence, performs well compared to Contrastive Divergence at learning good generative models in RBMs. Even though the allowable learning rate is much higher than would be predicted from the mixing rate of the persistent Markov chains, it is still considerably lower than the rate used for contrastive divergence learning because the gradient estimate it provides has lower bias but much higher variance, especially when using mini-batch learning rather than full batch learning.

2.2 A Variational Approach to Estimating the Data-Dependent Expectations

As already mentioned, persistent Markov chains are less appropriate for estimating the data-dependent statistics, especially with mini-batch learning on large datasets. Fortunately, variational approximations work well for estimating the data-dependent statistics. Given the data, it is typically quite reasonable for the posterior distribution over latent variables to be unimodal, especially for applications like speech and vision where normal data-vectors really do have a single correct explanation and the data is rich enough to allow a good generative model to infer that explanation.

In variational learning (Zemel (1993); Hinton and Zemel (1994); Neal and Hinton (1998); Jordan et al. (1999)), the true posterior distribution over latent variables $P(\mathbf{h}|\mathbf{v};\theta)$ for each training vector $\mathbf{v}$, is replaced by an approximate posterior $Q(\mathbf{h}|\mathbf{v};\mu)$ and the parameters are updated to maximize the variational lower bound on the log-likelihood:

$$\begin{aligned}\log P(\mathbf{v};\theta) &\geq \sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{v};\mu)\log P(\mathbf{v},\mathbf{h};\theta) + \mathcal{H}(Q) \\ &= \log P(\mathbf{v};\theta) - \mathrm{KL}\left[Q(\mathbf{h}|\mathbf{v};\mu)||P(\mathbf{h}|\mathbf{v};\theta)\right], \qquad (10)\end{aligned}$$

where $\mathcal{H}(\cdot)$ is the entropy functional.

Variational learning has the nice property that in addition to trying to maximize the log-likelihood of the training data, it tries to find parameters that minimize the Kullback-Leibler divergence between the approximating and true posteriors. Making the true posterior approximately unimodal, even if it means sacrificing some log-likelihood, could be advantageous for a system that will use the posterior to control its actions. Having multiple alternative representations of the same sensory input increases the likelihood compared with a single explanation of the same quality, but it makes it more difficult to associate an appropriate action with that sensory input. Variational inference that uses a factorial distribution to approximate the posterior helps to eliminate this problem. During learning, if the posterior given a training input vector is multimodal, the variational inference will lock onto one mode, and learning will make that mode more probable. Our learning algorithm will therefore tend to find regions in the parameter space in which the true posterior is dominated by a single mode.

For simplicity and speed, we approximate the true posterior using a fully factorized distribution (*i.e.* the naive mean-field approximation), $Q(\mathbf{h};\mu) = \prod_{j=1}^{\mathcal{F}} q(h_i)$, where $q(h_i = 1) = \mu_i$ and $\mathcal{F}$ is the number of hidden units. The lower bound on the log-probability of the data takes the following form:

$$\log P(\mathbf{v};\theta) \geq \frac{1}{2}\sum_{i,k} L_{ik}v_i v_k + \frac{1}{2}\sum_{j,m} J_{jm}\mu_j\mu_m + \sum_{i,j} W_{ij}v_i\mu_j - \log\mathcal{Z}(\theta) + \sum_j \left[\mu_j \log\mu_j + (1-\mu_j)\log(1-\mu_j)\right].$$

The learning proceeds by first maximizing this lower bound with respect to the variational parameters μ for fixed θ, which results in the mean-field fixed-point equations:

$$\mu_j \leftarrow g\left(\sum_i W_{ij}v_i + \sum_{m\neq j} J_{mj}\mu_m\right). \tag{11}$$

This is followed by applying stochastic approximation to update model parameters θ.

We emphasize that variational approximations should not be used for estimating the data-independent statistics in the Boltzmann Machine learning rule, as attempted in Galland (1991), for two separate reasons. First, a factorial approximation cannot model the highly multi-modal, data-independent distribution that is typically required. Second, the minus sign causes the parameters to be adjusted so that the true model distribution becomes as different as possible from the variational approximation.

3. Learning Deep Boltzmann Machines (DBMs)

The algorithm above can learn Boltzmann Machines with any pattern of connectivity between the units, but it can be made particularly efficient in "deep" Boltzmann Machines that have multiple hidden layers but only have connections between adjacent layers as shown in Fig. 2 (Salakhutdinov and Hinton (2009a); Salakhutdinov (2010)). Deep Boltzmann Machines are interesting for several reasons. First, like Deep Belief Networks, DBMs have the ability to learn internal representations that capture very complex statistical structure in the higher layers. As has already been demonstrated for DBNs, this is a promising way of solving object and speech recognition problems (Bengio (2009); Bengio and LeCun (2007b); Hinton et al. (2006); Dahl and Hinton (2009); Dahl (2010)). High-level representations can be built from a large supply of unlabeled data and a much smaller supply of labeled data can then be used to slightly fine-tune the model for a specific discrimination task. Second, again like DBNs, if DBMs are learned in the right way there is a very fast way to initialize the states of the units in all layers by simply doing a single bottom-up pass using twice the weights to compensate for the initial lack of top-down feedback. Third, unlike DBNs and unlike many other models with deep architectures (Ranzato et al. (2007); Vincent et al. (2008); Serre

Algorithm 2 Learning Procedure for a General Boltzmann Machine

```
1: Given: a training set of $N$ binary data vectors $\{\mathbf{v}\}_{n=1}^{N}$, and $M$, the number of samples.
2: Randomly initialize parameter vector $\theta^0$ and $M$ samples: $\{\tilde{\mathbf{v}}^{0,1}, \tilde{\mathbf{h}}^{0,1}\}, ..., \{\tilde{\mathbf{v}}^{0,M}, \tilde{\mathbf{h}}^{0,M}\}$.
3: for $t = 0$ to $T$ (number of iterations) do

4:     // Variational Inference:
5:     for each training example $\mathbf{v}^n$, $n = 1$ to $N$ do
6:         Randomly initialize $\mu$ and run mean-field updates until convergence:
               $\mu_j \leftarrow g\left(\sum_i W_{ij} v_i + \sum_{m \neq j} J_{mj} \mu_m\right)$.
7:         Set $\mu^n = \mu$.
8:     end for

9:     // Stochastic Approximation:
10:    for each sample $m = 1$ to $M$ do
11:        Sample $(\tilde{\mathbf{v}}^{t+1,m}, \tilde{\mathbf{h}}^{t+1,m})$ given $(\tilde{\mathbf{v}}^{t,m}, \tilde{\mathbf{h}}^{t,m})$ by running a Gibbs sampler (Eqs. 4, 5).
12:    end for

13:    // Parameter Update:
14:    $W^{t+1} = W^t + \alpha_t\left(\frac{1}{N}\sum_{n=1}^{N} \mathbf{v}^n (\mu^n)^\top - \frac{1}{M}\sum_{m=1}^{M} \tilde{\mathbf{v}}^{t+1,m} (\tilde{\mathbf{h}}^{t+1,m})^\top\right)$.
15:    $J^{t+1} = J^t + \alpha_t\left(\frac{1}{N}\sum_{n=1}^{N} \mu^n (\mu^n)^\top - \frac{1}{M}\sum_{m=1}^{M} \tilde{\mathbf{h}}^{t+1,m} (\tilde{\mathbf{h}}^{t+1,m})^\top\right)$.
16:    $L^{t+1} = L^t + \alpha_t\left(\frac{1}{N}\sum_{n=1}^{N} \mathbf{v}^n (\mathbf{v}^n)^\top - \frac{1}{M}\sum_{m=1}^{M} \tilde{\mathbf{v}}^{t+1,m} (\tilde{\mathbf{v}}^{t+1,m})^\top\right)$.
17:    Decrease $\alpha_t$.
18: end for
```

et al. (2007)), the approximate inference procedure, after the initial bottom-up pass, can incorporate top-down feedback, allowing DBMs to use higher-level knowledge to resolve uncertainty about intermediate level features (Salakhutdinov and Larochelle (2010)).

Let us consider a three-hidden-layer DBM, as shown in Fig. 2, right panel, with no within-layer connections. The energy of the state $\{\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3\}$ is defined as:

$$E(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3; \theta) = -\mathbf{v}^\top W^1 \mathbf{h}^1 - \mathbf{h}^{1\top} W^2 \mathbf{h}^2 - \mathbf{h}^{2\top} W^3 \mathbf{h}^3, \tag{12}$$

where $\theta = \{W^1, W^2, W^3\}$ are the model parameters, representing visible-to-hidden and hidden-to-hidden symmetric interaction terms.

The probability that the model assigns to a visible vector $\mathbf{v}$ is:

$$P(\mathbf{v}; \theta) = \frac{1}{\mathcal{Z}(\theta)} \sum_{\mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3} \exp\left(-E(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3; \theta)\right). \tag{13}$$

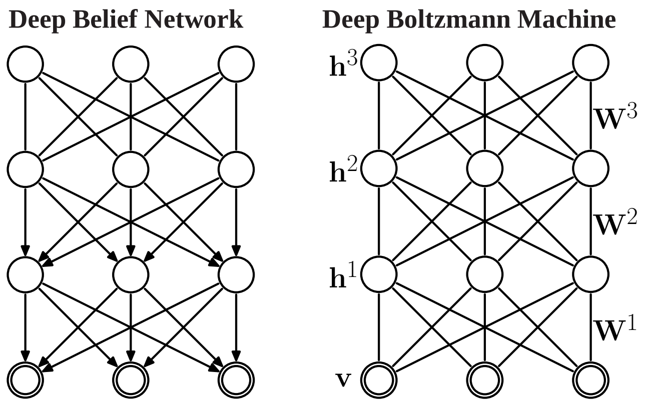

Figure 2: **Left:** Deep Belief Network (DBN), with the top two layers forming an undirected graph and the remaining layers form a belief net with directed, top-down connections **Right:** Deep Boltzmann Machine (DBM), with both visible-to-hidden and hidden-to-hidden connections but with no within-layer connections. All the connections in a DBM are undirected.

The conditional distributions over the visible and the three sets of hidden units are given by logistic functions:

$$p(h_j^1 = 1|\mathbf{v}, \mathbf{h}^2) = g\left(\sum_i W_{ij}^1 v_i + \sum_m W_{jm}^2 h_m^2\right), \tag{14}$$

$$p(h_m^2 = 1|\mathbf{h}^1, \mathbf{h}^3) = g\left(\sum_j W_{jm}^2 h_j^1 + \sum_l W_{ml}^3 h_l^3\right), \tag{15}$$

$$p(h_l^3 = 1|\mathbf{h}^2) = g\left(\sum_m W_{ml}^3 h_m^2\right), \tag{16}$$

$$p(v_i = 1|\mathbf{h}^1) = g\left(\sum_j W_{ij}^1 h_j^1\right). \tag{17}$$

As mentioned before, the learning procedure for general Boltzmann Machines described above can be applied to DBMs that start with randomly initialized weights, but it works much better if the weights are initialized sensibly. With small random weights, hidden units in layers that are far from the data are very under-constrained so there is no consistent learning signal for their weights. With larger random weights the initialization imposes a strong random bias on the feature detectors learned in the hidden layers. Even when the ultimate goal is some unknown discrimination task, it is much better to bias these feature detectors towards ones that form a good generative model of the data. We now describe how this can be done.

3.1 Greedy Layerwise Pre-training of DBNs

Hinton et al. (2006) introduced a greedy, layer-by-layer unsupervised learning algorithm that consists of learning a stack of RBMs one layer at a time. After greedy learning, the whole stack can be viewed as a single probabilistic model called a Deep Belief Network. Surprisingly, this composite model is *not* a Deep Boltzmann Machine. The top two layers form a Restricted Boltzmann Machine, but the lower layers form a *directed* sigmoid belief network (see Fig. 2, left panel).

After learning the first RBM in the stack, the generative model can be written as:

$$P(\mathbf{v};\theta) \;=\; \sum_{\mathbf{h}^1} P(\mathbf{h}^1;W^1)P(\mathbf{v}|\mathbf{h}^1;W^1), \tag{18}$$

where $P(\mathbf{h}^1;W^1) = \sum_{\mathbf{v}} P(\mathbf{h}^1,\mathbf{v};W^1)$ is a prior over $\mathbf{h}^1$ that is implicitly defined by W^1. Using the same parameters to define both the prior over $\mathbf{h}^1$ and the likelihood term $P(\mathbf{v}|\mathbf{h}^1)$ seems like an odd thing to do for those who are more familiar with directed graphical models, but it makes inference much easier and it is only a temporary crutch: the prior over $\mathbf{h}^1$ defined by W^1 will be thrown away and replaced by a better prior defined by the weights, W^2, of the next RBM in the stack.

The second RBM in the stack attempts to learn a better overall model by leaving $P(\mathbf{v}|\mathbf{h}^1;W^1)$ fixed and replacing $P(\mathbf{h}^1;W^1)$ by $P(\mathbf{h}^1;W^2) = \sum_{\mathbf{h^2}} P(\mathbf{h}^1,\mathbf{h}^2;W^2)$, where W^2 is initialized at $(W^1)^\top$ and then improved by following the gradient of a variational lower bound on the log probability of the training data with respect to W^2. The variational bound was first derived using coding arguments in Hinton and Zemel (1994) and applies to DBMs as well as to DBNs. For a dataset containing N training examples, it has the form:

$$\begin{aligned}\sum_{n=1}^{N} \log P(\mathbf{v}^n;\theta) &\geq \sum_n \mathrm{E}_{Q(\mathbf{h}^1|\mathbf{v}^n)}\left[\log(P(\mathbf{v}^n|\mathbf{h}^1;W^1))\right] - \sum_n \mathrm{KL}\left(Q(\mathbf{h}^1|\mathbf{v}^n)||P(\mathbf{h}^1;W^2)\right) \\ &= \sum_n \left[\sum_{\mathbf{h}^1} Q(\mathbf{h}^1|\mathbf{v}^n)\left[\log(P(\mathbf{v}|\mathbf{h}^1;W^1))\right] + \mathcal{H}(Q)\right] + \sum_n \sum_{\mathbf{h}^1} Q(\mathbf{h}^1|\mathbf{v}^n)\log P(\mathbf{h}^1;W^2), \end{aligned} \tag{19}$$

where $\mathcal{H}(\cdot)$ is the entropy functional and $Q(\mathbf{h}^1|\mathbf{v})$ is any approximation to the posterior distribution over hidden vectors for the DBN containing hidden layers $\mathbf{h}^1$ and $\mathbf{h}^2$. The approximation we use is the true posterior over $\mathbf{h}^1$ for the first RBM, $P(\mathbf{h}^1|\mathbf{v},W^1)$. As soon as W^2 ceases to be identical to $(W^1)^\top$ this is no longer the true posterior for the DBN.

Changing W^2 only affects the last sum in Eq. 19 so maximizing the bound, summed over all the training cases, *w.r.t.* W^2 amounts to learning a better model of the mixture, over all N training cases, of the true posteriors of the first RBM over $\mathbf{h}^1$. Each of these posteriors is factorial, but their mixture $\frac{1}{N}\sum_n Q(\mathbf{h}^1|\mathbf{v}^n)$, which we call the aggregated posterior, is typically very far from factorial.

If the second RBM is initialized to be the same as the first RBM but with its visible and hidden units interchanged, the second RBM's model of its visible vectors is identical to the first RBM's model of its hidden vectors, so using the second RBM to define $P(\mathbf{h}^1)$ does not change the model provided $W^2 = (W^1)^\top$. Changing W^2 so that the second RBM becomes a better model of the aggregated posterior over $\mathbf{h}^1$ is then guaranteed to improve the variational bound for the whole DBN on the log likelihood of the training data.

This argument can be applied recursively to learn as many layers of features as desired. Each RBM in the stack performs exact inference while it is being learned, but once its implicit prior over its hidden vectors has been replaced by a better prior defined by the higher-level RBM, the simple inference procedure ceases to be exact. As the stack gets deeper, the simple inference procedure used for the earlier layers can be expected to become progressively less correct. Nevertheless, each time a new layer is added, the variational bound for the deeper system is better than the bound for its predecessor. When a third hidden layer is added, for example, the bound of Eq. 19 gets replaced by a bound in which the last sum:

$$\sum_n \sum_{\mathbf{h}^1} Q(\mathbf{h}^1|\mathbf{v}^n) \log P(\mathbf{h}^1; W^2) \tag{20}$$

is replaced by

$$\sum_n \sum_{\mathbf{h}^1} Q(\mathbf{h}^1|\mathbf{v}^n) \Big(\mathrm{E}_{Q(\mathbf{h}^2|\mathbf{h}^1)} \left[\log(P(\mathbf{h}^1|\mathbf{h}^2; W^2)) \right] - \mathrm{KL}(Q(\mathbf{h}^2|\mathbf{h}^1)||P(\mathbf{h}^2; W^3)) \Big). \tag{21}$$

When the second RBM is learned, the log probability of the training data also improves because the bound starts off tight, but this is not guaranteed for deeper layers since, for these layers, the variational bound does not start off tight. The bound could therefore improve as the log probability of the training data falls provided the bound becomes tighter at a higher rate than the log probability falls.

The improvement of the bound is only guaranteed if each RBM in the stack starts with the same weights as the previous RBM and follows the gradient of the log likelihood, using the posterior distributions over the hidden units of the previous RBM as its data. In practice we violate this condition by using gross approximations to the gradient such as contrastive divergence. The real value of deriving the variational bound is to allow us to understand why it makes sense to use the aggregated posterior distributions of one RBM as the training data for the next RBM and why the combined model is a Deep Belief Net rather than a Deep Boltzmann Machine.

3.2 Greedy Layerwise Pre-training of DBMs

Even though the simple way of stacking RBMs leads to a Deep Belief Net, it is possible to modify the procedure so that stacking produces a Deep Boltzmann Machine. We start by giving an intuitive argument about how to combine RBMs to get a Deep Boltzmann Machine using three different operations, one for the bottom layer, one for the top layer, and one operation that is repeated for all the intermediate layers. We then show that two of the three intuitively derived operations that we used to pre-train a DBM in our experiments are guaranteed to improve a variational bound and the remaining operation is a close approximation to a method that is guaranteed to improve a bound.

After training the 2nd layer RBM in a DBN, there are two different ways of computing a factorial approximation to the true posterior $P(\mathbf{h}^1|\mathbf{v}; W^1, W^2)$. The obvious way is to ignore the 2nd layer RBM and use the $P(\mathbf{h}^1|\mathbf{v}; W^1)$ defined by the first RBM. An alternative method is to first sample $\mathbf{h}^1$ from $P(\mathbf{h}^1|\mathbf{v}; W^1)$, then sample $\mathbf{h}^2$ from $P(\mathbf{h}^2|\mathbf{h}^1; W^2)$, and then use the $P(\mathbf{h}^1|\mathbf{h}^2; W^2)$ defined by the second RBM. The sampling noise in the second method can be reduced by using a further approximation in which the sampled binary values are replaced by their probabilities. The second method will tend to over-emphasize the prior for $\mathbf{h}^1$ defined by W^2 whereas the first method will tend to under-emphasize this prior in favor of the earlier prior defined by W^1 that it replaced.

Given these two different approximations to the posterior, it would be possible to take a geometric average of the two distributions. This can be done by first performing a bottom-up pass to infer $\mathbf{h}^2$ then using $1/2 W^1$ and $1/2 W^2$ to infer $\mathbf{h}^1$ from both $\mathbf{v}$ and $\mathbf{h}^2$. Notice that $\mathbf{h}^2$ is inferred from $\mathbf{v}$ so it is not legitimate to sum the full top-down and bottom-up influences. This would amount to "double-counting" the evidence provided by $\mathbf{v}$ and would give a distribution that was much too

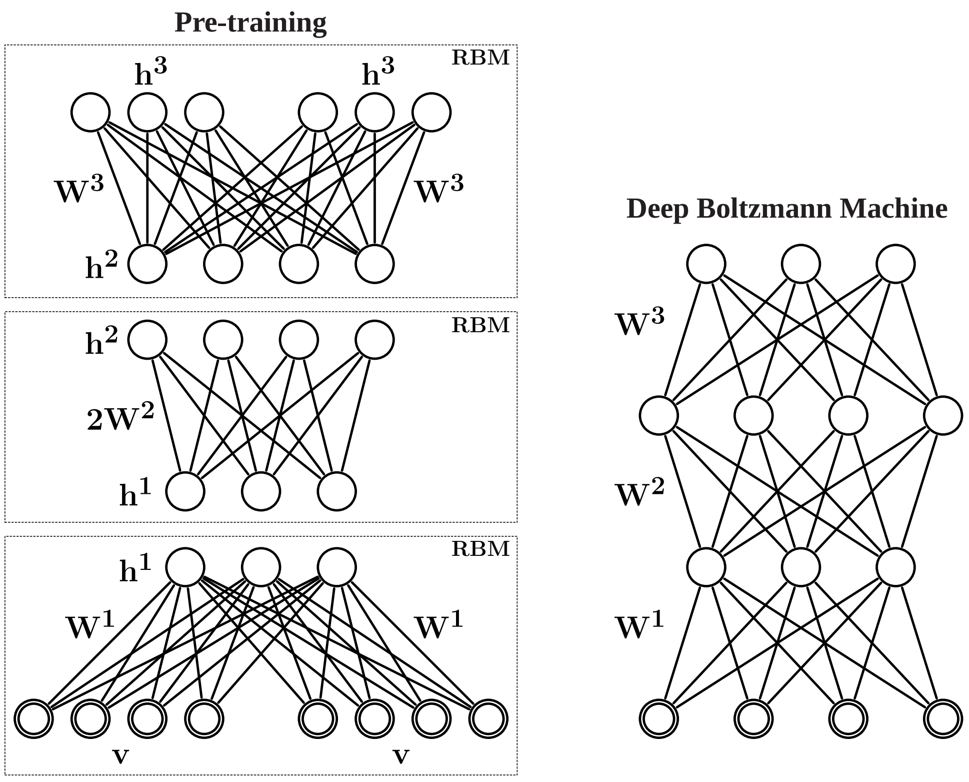

Figure 3: Pre-training a DBM with three hidden layers consists of learning a stack of RBMs that are then composed to create a DBM. The first and last RBMs in the stack need to be modified by copying the visible or hidden units.

sharp. Experiments with trained DBNs confirm that averaging the top-down and bottom-up inputs works well for inference and adding them works badly.

This reasoning can be extended to a much deeper stack of greedily trained RBMs. The initial, bottom-up inference that is performed in a DBN can be followed by a stage in which all of the weights are halved and the states of the units in the intermediate layers are resampled by summing the top-down and bottom-up inputs to a layer. If we alternate between resampling the odd-numbered layers and resampling the even-numbered layers, this corresponds to block Gibbs sampling in a Deep Boltzmann Machine with the visible and top-layer units clamped. So after learning a stack of RBMs we can either compose them to form a DBN or we can halve all the weights and compose them to form a DBM. Moreover, given the way the DBM was created, there is a very fast way to initialize all of the hidden layers when given a data-vector: simply perform a bottom-up pass using twice the weights of the DBM to compensate for the lack of top-down input.

There is an annoying problem with this method of pre-training a DBM. For the intermediate layers, initializing at the halved weights is fine because it can be viewed as taking the geometric mean of the bottom-up and top-down models, but for the visible layer and the top layer it is not legitimate because they only receive input from one other layer. Both the top and the bottom layer need to be updated when estimating the data-independent statistics and we cannot use weights that are bigger in one direction than the other because this does not correspond to Gibbs sampling in any energy function. So we need to use a special trick when pre-training the first and last RBMs in

Algorithm 3 Greedy Pre-training Algorithm for a Deep Boltzmann Machine with L-layers.

```
1: Make two copies of the visible vector and tie the visible-to-hidden weights $W^1$. Fit $W^1$ of the 1st layer RBM to data.
2: Freeze $W^1$ that defines the 1st layer of features, and use samples $\mathbf{h}^l$ from $P(\mathbf{h}^1|\mathbf{v}, 2W^1)$ (Eq. 22) as the data for training the next layer RBM with weight vector $2W^2$.
3: Freeze $W^2$ that defines the 2nd layer of features and use the samples $\mathbf{h}^2$ from $P(\mathbf{h}^2|\mathbf{h}^1, 2W^2)$ as the data for training the 3rd layer RBM with weight vector $2W^3$.
4: Proceed recursively for the next layers $L - 1$.
5: When learning the top-level RBM, double the number of hidden units and tie the visible-to-hidden weights $W^L$.
6: Use the weights $\{W^1, W^2, ...., W^L\}$ to compose a Deep Boltzmann Machine.
```

the stack. For the first RBM, we make two copies of the visible units and tie the weights to the two copies as shown in Fig. 3, left panel. Even though the copies are always identical in the data, we do not insist that they have the same state vectors when reconstructing them from the hidden units because this constraint would effectively double the top-down weights to a visible unit.

In this modified RBM with tied parameters, the conditional distributions over the hidden and visible states are defined as:

$$p(h_j^1 = 1|\mathbf{v}, \mathbf{v}') = g\left(\sum_i W_{ij}^1 (v_i + v_i')\right), \tag{22}$$

$$p(v_i = 1|\mathbf{h}^1) = p(v_i' = 1|\mathbf{h}^1) = g\left(\sum_j W_{ij}^1 h_j^1\right). \tag{23}$$

Contrastive Divergence learning still works well and the modified RBM is good at reconstructing its training data.

Conversely, for the top-level RBM, we make two copies of the hidden units with tied weights and do not insist that they have the same state vectors. For a DBM with three hidden layers, the conditional distributions for this model take the form:

$$p(h_m^2 = 1|\mathbf{h}^3) = g\left(\sum_l W_{ml}^3 h_l^{3(a)} + \sum_l W_{ml}^3 h_l^{3(b)}\right), \tag{24}$$

$$p(h_l^{3(a)} = 1|\mathbf{h}^2) = p(h_l^{3(b)} = 1|\mathbf{h}^2) = g\left(\sum_m W_{ml}^3 h_m^2\right). \tag{25}$$

For the intermediate RBM we simply define W^2 to be half of the weight matrix learned by the RBM, so the conditional distributions take the form:

$$p(h_j^1 = 1|\mathbf{h}^2) = g\left(2\sum_m W_{jm}^2 h_m^2\right), \tag{26}$$

$$p(h_m^2 = 1|\mathbf{h}^1) = g\left(2\sum_j W_{jm}^2 h_j^1\right). \tag{27}$$

When these three modules are composed to form a single system, the total input coming into the first and second hidden layers is halved, which leads to the following conditional distributions over $\mathbf{h}^1$ and $\mathbf{h}^2$:

$$p(h_j^1 = 1|\mathbf{v}, \mathbf{h}^2) \;=\; g\left(\sum_i W_{ij}^1 v_i + \sum_m W_{jm}^2 h_m^2\right), \tag{28}$$

$$p(h_m^2 = 1|\mathbf{h}^1, \mathbf{h}^3) \;=\; g\left(\sum_j W_{jm}^2 h_j^1 + \sum_l W_{ml}^3 h_l^3\right). \tag{29}$$

The conditional distributions over $\mathbf{v}$ and $\mathbf{h}^3$ remain the same as defined by Eqs. 23, 25.

Observe that the conditional distributions defined by this composite model are exactly the same as the conditional distributions defined by the DBM (Eqs. 14, 15, 16, 17). Therefore, after greedily pre-training a stack of RBMs with appropriate modifications of the first and last RBM, they can be composed to create a Deep Boltzmann Machine.

Greedily pre-training the weights of a DBM in this way serves two purposes. First, it initializes the weights to reasonable values. Second, it ensures that there is a fast way of performing approximate inference by a single bottom-up pass using twice the weights. This eliminates the need to store the hidden states that were inferred last time a training case was used (Neal (1992)) or to use simulated annealing from random initial states of the hidden units (Hinton and Sejnowski (1983)). This fast approximate inference is used to initialize the mean-field, iterative inference which then converges much faster than mean-field with random initialization. Since the mean-field inference uses real-valued probabilities rather than sampled binary states, we also use probabilities rather than sampled binary states for the initial bottom-up inference.

3.3 A Variational Bound for Greedy Layerwise Pre-training of DBMs

The explanation of DBM pre-training given in the previous section is motivated by the need to end up with a deep network that has symmetric weights between all adjacent pairs of layers. The pre-training is motivated by intuitive arguments about combining top-down and bottom-up effects. However, unlike the pre-training of a DBN, it lacks a proof that each time a layer is added to the DBM, the variational bound for the deeper DBM is better than the bound for the previous DBM. We now show that the method we use for training the first and last RBMs in the stack is a correct way of improving a variational bound, and the method for adding the intermediate layers is quite close to being correct.

The basic idea for pre-training a DBM is to start by learning a model in which the prior over hidden vectors, $p(\mathbf{h}^1; W^1)$, is the normalized product of two identical distributions. Then one of these distributions is discarded and replaced by the square root of a better prior $p(\mathbf{h}^1; W^2)$ that has been trained to fit the aggregated posterior of the first model. To be more precise, let us first consider a simpler case of pre-training a two-hidden-layer DBM.

Pre-training a two-hidden-layer DBM

Suppose we start by training the RBM with tied weights shown in Fig. 4a that has two sets of visible units, each of which sees one copy of the data. After learning, we can write down the variation lower

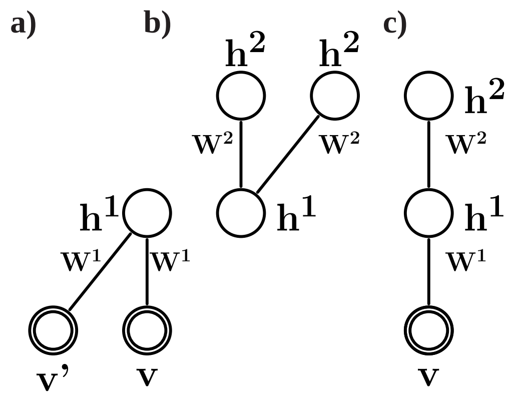

Figure 4: Pre-training a Deep Boltzmann Machine with two hidden layers.

bound of Eq. 19:

$$\sum_n \log P(\mathbf{v}^n) \geq \sum_n \mathrm{E}_{Q(\mathbf{h}^1|\mathbf{v}^n)} \left[\log(P(\mathbf{v}^n|\mathbf{h}^1; W^1))\right] - \sum_n \mathrm{KL}\left(Q(\mathbf{h}^1|\mathbf{v}^n)||P(\mathbf{h}^1; W^1)\right). \quad (30)$$

The model's marginal distribution over $\mathbf{h}^1$ is the product of two identical distributions, one defined by an RBM composed of $\mathbf{h}^1$ and $\mathbf{v}$ and the other defined by an identical RBM composed of $\mathbf{h}^1$ and $\mathbf{v}^{'}$:

$$p(\mathbf{h}^1; W^1) = \frac{1}{\mathcal{Z}}\Big(\sum_{\mathbf{v}'} e^{\mathbf{v}'^\top W^1 \mathbf{h}^1}\Big)\Big(\sum_{\mathbf{v}} e^{\mathbf{v}^\top W^1 \mathbf{h}^1}\Big), \quad (31)$$

where $\mathcal{Z}$ is the normalizing constant[3]. The idea is to keep one of these RBMs and replace the other by a square root a better prior $P(\mathbf{h}^1; W^2)$. To do so we train the second-layer RBM with two sets of hidden units to be a better model the aggregated posterior $\frac{1}{N}\sum_n Q(\mathbf{h}^1|\mathbf{v}^n)$ of the first model (see Fig. 4b), so that:

$$\sum_n \mathrm{KL}(Q(\mathbf{h}^1|\mathbf{v}^n; W^1)||P(\mathbf{h}^1; W^2)) \leq \sum_n \mathrm{KL}(Q(\mathbf{h}^1|\mathbf{v}^n; W^1)||P(\mathbf{h}^1; W^1)). \quad (32)$$

Similar to Eq. 31, the distribution over $\mathbf{h}^1$ defined by the second-layer RBM is also the product of two identical distributions, one for each set of hidden units. This implies that taking a square root amounts to simply keeping one such distribution.

Once the two RBMs are composed to form a two-hidden-layer DBM model (see Fig. 4c), the marginal distribution over $\mathbf{h}^1$ is the geometric mean of the two probability distributions: $P(\mathbf{h}^1; W^1), P(\mathbf{h}^1; W^2)$ defined by the first and second-layer RBMs (*i.e.* the renormalized pairwise products of the square roots of the two probabilities for each event):

$$P(\mathbf{h}^1; W^1, W^2) = \frac{1}{\mathcal{Z}}\Big(\sum_{\mathbf{v}} e^{\mathbf{v}^\top W^1 \mathbf{h}^1}\Big)\Big(\sum_{\mathbf{h}^2} e^{\mathbf{h}^{1\top} W^2 \mathbf{h}^2}\Big). \quad (33)$$

3. The biases learned for $\mathbf{h}^1$ are shared equally between the two RBMs.

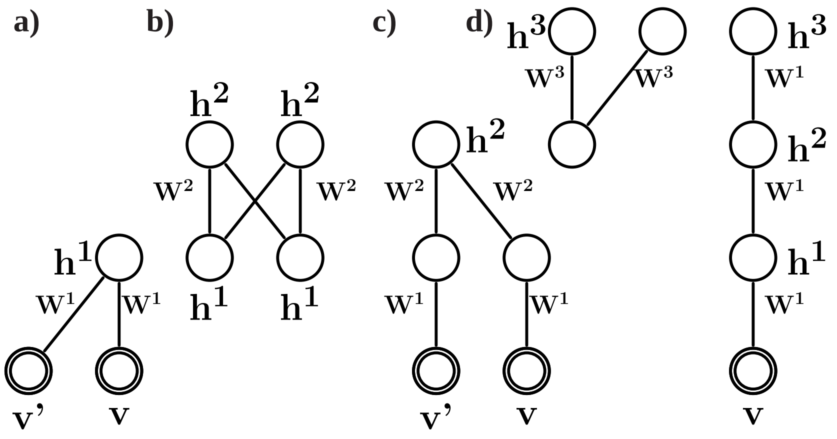

Figure 5: Pre-training a Deep Boltzmann Machine with three hidden layers.

The variational lower bound of Eq. 30 improves because replacing half of the prior by a better model reduces the Kullback-Leibler divergence. Indeed, Eq. 32 immediately implies:

$$\sum_n \mathrm{KL}(Q(\mathbf{h}^1|\mathbf{v}^n;W^1)||gm(P(\mathbf{h}^1;W^1),P(\mathbf{h}^1;W^2)) \leq \sum_n \mathrm{KL}(Q(\mathbf{h}^1|\mathbf{v}^n;W^1)||P(\mathbf{h}^1;W^1)), \tag{34}$$

where $gm(\cdot,\cdot)$ is the geometric mean of two probability distributions. Due to the convexity of asymmetric divergence, this is guaranteed to improve the variational bound of the training data by at least half as much as fully replacing the original prior. It is also guaranteed to loosen the variational bound by at most half as much as fully replacing the original prior, assuming that inference is still performed assuming the original prior.

This argument shows that the apparently unprincipled hack of using two sets of visible units or two sets of hidden units to cope with the "end effects" when creating a DBM from a stack of RBMs is actually exactly the right thing to do in order to improve a variational bound[4].

Pre-training a Deep Boltzmann Machine

When pre-training a DBM with more than two layers. the discarded half of the previous prior is replaced by half of another *Boltzmann machine* that is also a product of two identical Boltzmann Machines and has been trained to be a better model of the aggregated posterior over $\mathbf{h}^1$. There are two different cases to consider. The easy case is when we add the final, L^{th} layer of the DBM. Similar to the two-hidden-layer construction, we simply replace half of the previous prior by half of the distribution defined by an RBM with the architecture shown at Fig. 5d.

Adding non-final layers in a way that guarantees that the new variational bound is better than the previous one is a bit more complicated. First we need to train an RBM to be a better model of the aggregated posterior of the previous RBM. Then we need to modify this trained RBM to get the

4. An RBM with two sets of visible units learns a different model than an RBM with only one set of visible units and the variational bound is on the log probability of this different model

square root of the distribution that it defines over its visible units. Then we need to make another modification to get the square root of the prior that it defines over its hidden units so that, when we add the next layer, we can replace the other half of that prior by a better distribution. The right way to do this is to train an RBM that has two copies of its visible units and two copies of its hidden units with all four weight matrices tied as shown in Fig. 5b. Then we make two copies of the previous DBM after modifying it to take the square root of the prior over its hidden units. We use one copy of the hidden units of the new RBM as the top layer of the new DBM as shown in Fig. 5c. This creates a DBM with a better variational bound for the distribution that it defines over each set of its visible units. Finally, in preparation for adding the next layer, we throw away one copy of the previous DBM, thus taking the square root of the prior over the top layer.

Instead of using this correct method, we approximated it by training an ordinary RBM with no duplicate sets of units on the aggregated posterior from the existing DBM. Then we simply halved its weights and biases. Halving the weights takes the square root of the *joint* distribution over pairs of visible and hidden vectors and it also takes the square root of the conditional distribution over visible vectors given a hidden vector (or vice versa), but it does not take the square root of the marginal distributions over the visible or the hidden vectors. This is most easily seen by considering the ratios of the probabilities of two visible vectors, $\mathbf{v}_\alpha$ and $\mathbf{v}_\beta$. Before halving the weights, their probability ratio in the marginal distribution over visible vectors is given by:

$$\frac{P(\mathbf{v}_\alpha)}{P(\mathbf{v}_\beta)} = \frac{\sum_{\mathbf{h}} e^{-E(\mathbf{v}_\alpha,\mathbf{h})}}{\sum_{\mathbf{h}} e^{-E(\mathbf{v}_\beta,\mathbf{h})}}. \tag{35}$$

In the RBM with halved weights, all of the exponents are halved which takes the square root of every individual term in each sum, but this does not take the square root of the ratio of the sums. This argument shows that the apparently unproblematic idea of halving the weights of all the intermediate RBMs in the stack is not the right thing to do if we want to ensure that as each layer is added, the new variational bound is better than the old one. Nevertheless, this method is fast and works quite well in practice and it is the method we used.

In the correct method of adding an intermediate hidden layer, throwing away one set of hidden units halves the total expected energy of all terms involving hidden units, but it also halves the entropy, so it successfully halves the free energy of each visible vector which is what is required to take the square root of the marginal distribution over visible vectors[5].

4. Evaluating Deep Boltzmann Machines

Assessing the generalization performance of DBMs plays an important role in model selection, model comparison, and controlling model complexity. In this section we discuss two ways of evaluating the generalization capabilities of DBMs: generative and discriminative.

4.1 Evaluating DBMs as Generative Models

We first focus on evaluating generalization performance of DBMs as density models. For many specific tasks, such as classification or information retrieval, performance of DBMs can be directly evaluated (see section 4.2). More broadly, however, the ability of DBMs to generalize can be evaluated by computing the probability of held-out input vectors, which is independent of any specific

5. We also halve the visible biases.

application. An unfortunate limitation of DBMs is that the probability of data under the model is known only up to a computationally intractable partition function. A good estimate of the partition function would allow us to assess generalization performance of DBMs as density models.

Recently, Salakhutdinov and Murray (2008) showed that a Monte Carlo based method, Annealed Importance Sampling (AIS) (Neal (2001)), can be used to efficiently estimate the partition function of an RBM. In this section we show how AIS can be used to estimate the partition functions of DBMs. Together with variational inference this will allow us to obtain good estimates of the lower bound on the log-probability of the training and test data.

Suppose we have two distributions defined on some space $\mathcal{X}$ with probability density functions: $P_A(\mathbf{x}) = P_A^*(\mathbf{x})/Z_A$ and $P_B(\mathbf{x}) = P_B^*(\mathbf{x})/Z_B$. Typically $P_A(\mathbf{x})$ is defined to be some simple distribution, with known partition function Z_A, from which we can easily draw *i.i.d.* samples. AIS estimates the ratio Z_B/Z_A by defining a sequence of intermediate probability distributions: $P_0, ..., P_K$, with $P_0 = P_A$ and $P_K = P_B$, which satisfy $P_k(\mathbf{x}) \neq 0$ whenever $P_{k+1}(\mathbf{x}) \neq 0$. For each intermediate distribution we must be able to easily evaluate the unnormalized probability $P_k^*(\mathbf{x})$, and we must also be able to sample $\mathbf{x}'$ given $\mathbf{x}$ using a Markov chain transition operator $T_k(\mathbf{x}'; \mathbf{x})$ that leaves $P_k(\mathbf{x})$ invariant. One general way to define this sequence is to set:

$$P_k(\mathbf{x}) \propto P_A^*(\mathbf{x})^{1-\beta_k} P_B^*(\mathbf{x})^{\beta_k}, \tag{36}$$

with $0 = \beta_0 < \beta_1 < ... < \beta_K = 1$ chosen by the user.

Using the special layer-by-layer structure of DBMs, we can derive an efficient AIS scheme for estimating the model's partition function. Let us consider a three-hidden-layer Boltzmann Machine (see Fig. 3, right panel) whose energy is defined as:

$$E(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3; \theta) = -\mathbf{v}^\top W^1 \mathbf{h}^1 - \mathbf{h}^{1\top} W^2 \mathbf{h}^2 - \mathbf{h}^{2\top} W^3 \mathbf{h}^3. \tag{37}$$

By explicitly summing out the 1st and the 3rd layer hidden units $\{\mathbf{h}^1, \mathbf{h}^3\}$, we can easily evaluate an unnormalized probability $P^*(\mathbf{v}, \mathbf{h}^2; \theta)$. We can therefore run AIS on a much smaller state space $\mathbf{x} = \{\mathbf{v}, \mathbf{h}^2\}$ with $\mathbf{h}^1$ and $\mathbf{h}^3$ analytically summed out. The sequence of intermediate distributions, parameterized by β, is defined as follows:

$$\begin{aligned} P_k(\mathbf{v}, \mathbf{h}^2; \theta) &= \sum_{\mathbf{h}^1, \mathbf{h}^3} P_k(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3; \theta) \\ &= \frac{1}{\mathcal{Z}_k} \prod_j \left(1 + e^{\beta_k \left(\sum_i v_i W_{ij}^1 + \sum_m h_m^2 W_{jm}^2\right)}\right) \prod_l \left(1 + e^{\beta_k \left(\sum_m h_m^2 W_{ml}^3\right)}\right). \end{aligned}$$

We gradually change β_k (the inverse temperature) from 0 to 1, annealing from a simple "uniform" model to the final complex model. Using Eqs. 14, 15, 16, 17, it is straightforward to derive a Gibbs transition operator that leaves $P_k(\mathbf{v}, \mathbf{h}^2; \theta)$ invariant:

$$p(h^1_j = 1|\mathbf{v}, \mathbf{h}^2) = g\left(\beta_k\left(\sum_i W^1_{ij} v_i + \sum_m W^2_{jm} h^2_m\right)\right), \tag{38}$$

$$p(h^2_m = 1|\mathbf{h}^1, \mathbf{h}^3) = g\left(\beta_k\left(\sum_j W^2_{jm} h^1_j + \sum_l W^3_{ml} h^3_l\right)\right), \tag{39}$$

$$p(h^3_l = 1|\mathbf{h}^2) = g\left(\beta_k \sum_m W^3_{ml} h^2_m\right), \tag{40}$$

$$p(v_i = 1|\mathbf{h}^1) = g\left(\beta_k \sum_j W^1_{ij} h^1_j\right). \tag{41}$$

Once we obtain an estimate of the global partition function $\hat{\mathcal{Z}}$, we can estimate, for a given test case $\mathbf{v}^*$, the variational lower bound of Eq. 10:

$$\begin{aligned}\log P(\mathbf{v}^*; \theta) &\geq -\sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{v}^*; \mu) E(\mathbf{v}^*, \mathbf{h}; \theta) + \mathcal{H}(Q) - \log \mathcal{Z}(\theta) \\ &\approx -\sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{v}^*; \mu) E(\mathbf{v}^*, \mathbf{h}; \theta) + \mathcal{H}(Q) - \log \hat{\mathcal{Z}},\end{aligned}$$

where we defined $\mathbf{h} = \{\mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3\}$. For each test vector under consideration, this lower bound is maximized with respect to the variational parameters μ using the mean-field update equations.

Furthermore, by explicitly summing out the states of the hidden units $\{\mathbf{h}^2, \mathbf{h}^3\}$, we can obtain a tighter variational lower bound on the log-probability of the test data. Of course, we can also adopt AIS to estimate $P^*(\mathbf{v}) = \sum_{\mathbf{h}^1,\mathbf{h}^2,\mathbf{h}^3} P^*(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3)$, and together with an estimate of the global partition function we can actually estimate the true log-probability of the test data. This however, would be computationally very expensive, since we would need to perform a separate AIS run for each test case. As an alternative, we could adopt a variation of the Chib-style estimator, proposed by Murray and Salakhutdinov (2009). In the case of Deep Boltzmann Machines, where the posterior over the hidden units tends to be unimodal, their proposed Chib-style estimator can provide good estimates of $\log P^*(\mathbf{v})$ in a reasonable amount of computer time.

In general, when learning a Deep Boltzmann Machine with more than two hidden layers, and no within-layer connections, we can explicitly sum out either odd or even layers. This will result in a better estimate of the model's partition function and tighter lower bounds on the log-probability of the test data.

4.2 Evaluating DBMs as Discriminative Models

After learning, the stochastic activities of the binary features in each layer can be replaced by deterministic, real-valued probabilities, and a Deep Boltzmann Machine with two hidden layers can be used to initialize a multilayer neural network in the following way. For each input vector $\mathbf{v}$, the mean-field inference is used to obtain an approximate posterior distribution $Q(\mathbf{h}^2|\mathbf{v})$. The marginals $q(h^2_j = 1|\mathbf{v})$ of this approximate posterior, together with the data, are used to create an "augmented" input for this deep multilayer neural network as shown in Fig. 6. Standard backpropagation of error derivatives can then be used to discriminatively fine-tune the model.

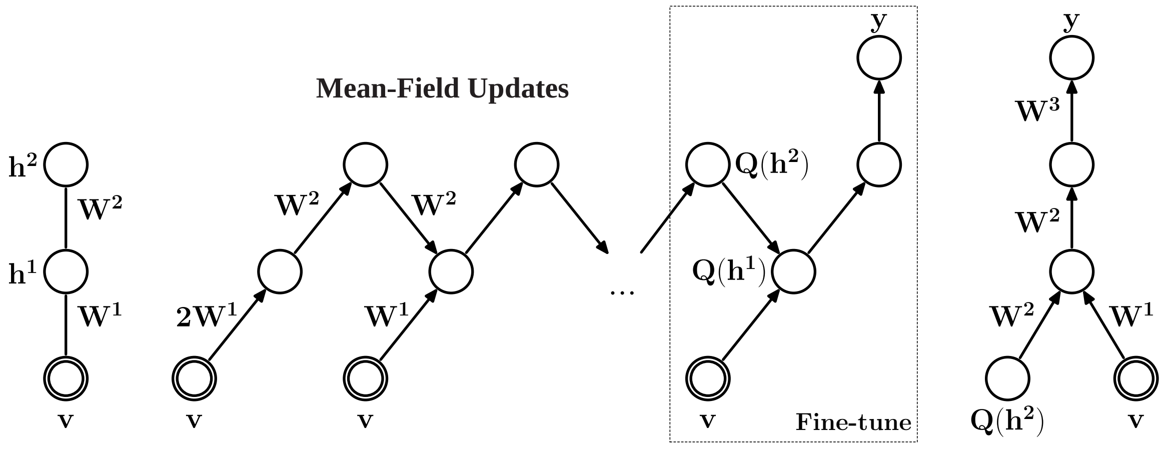

Figure 6: **Left:** A two-hidden-layer Boltzmann Machine. **Right:** After learning, DBM is used to initialize a multilayer neural network. The marginals of approximate posterior $q(h_j^2 = 1|\mathbf{v})$ are used as additional inputs. The network is fine-tuned by backpropagation.

The unusual representation of the input is a by-product of converting a DBM into a deterministic neural network. In general, the gradient-based fine-tuning may choose to ignore $Q(\mathbf{h}^2|\mathbf{v})$, i.e. drive the 1st layer connections W^2 to zero, which will result in a standard neural network. Conversely, the network may choose to ignore the input by driving the 1st layer weights W^1 to zero, and make its predictions based on only the approximate posterior. However, the network typically makes use of the entire augmented input for making predictions.

5. Experimental Results

In our experiments we used the MNIST and NORB datasets. To speed-up learning, we subdivided datasets into mini-batches, each containing 100 cases, and updated the weights after each mini-batch. The number of sample particles, used for approximating the model's expected sufficient statistics, was also set to 100. For the stochastic approximation algorithm, we always used 5 Gibbs updates. Each model was trained using 300,000 weight updates. The initial learning rate was set 0.005 and was decreased as 10/(2000+t), where t is the number of updates so far. For discriminative fine-tuning of DBMs we used the method of conjugate gradients on larger mini-batches of 5000 with three line searches performed for each mini-batch. Details of pre-training and fine-tuning, along with details of Matlab code that we used for learning and fine-tuning Deep Boltzmann Machines, can be found at http://web.mit.edu/$\sim$rsalakhu/www/software.html.

5.1 MNIST

The MNIST digit dataset contains 60,000 training and 10,000 test images of ten handwritten digits (0 to 9), with 28×28 pixels. Intermediate intensities between 0 and 255 were treated as probabilities and each time an image was used we sampled binary values from these probabilities independently for each pixel.

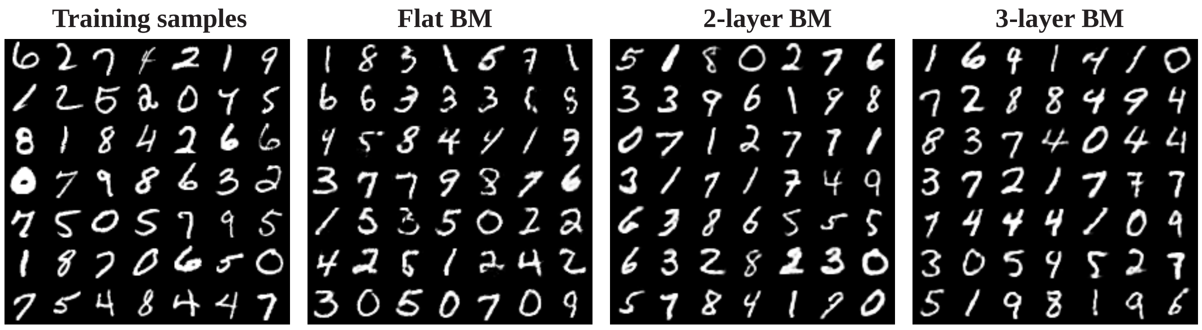

Figure 7: Random samples from the training set, and samples generated from three Boltzmann Machines by running the Gibbs sampler for 100,000 steps. The images shown are the *probabilities* of the binary visible units given the binary states of the hidden units

Table 1: Results of estimating partition functions of BM models, along with the estimates of lower bound on the average training and test log-probabilities. For all BMs we used 20,000 intermediate distributions. Results were averaged over 100 AIS runs.

	Estimates		Avg. log-prob.	
	$\log \hat{\mathcal{Z}}$	$\log(\hat{\mathcal{Z}} \pm \hat{\sigma})$	Test	Train
Flat BM	198.29	$198.17, 198.40$	-84.67	-84.35
2-layer BM	356.18	$356.06, 356.29$	-84.62	-83.61
3-layer BM	456.57	$456.34, 456.75$	-85.10	-84.49

In our first experiment we trained a fully connected “flat” BM on the MNIST dataset. The model had 500 hidden units and 784 visible units. To estimate the model’s partition function we used 20,000 β_k spaced uniformly from 0 to 1. Results are shown in table 1. The estimate of the lower bound on the average test log-probability was -84.67 per test case, which is slightly better compared to the lower bound of -85.97, achieved by a carefully trained two-hidden-layer Deep Belief Network (Salakhutdinov and Murray (2008)).

In our second experiment, we trained two Deep Boltzmann Machines: one with two hidden layers (500 and 1000 hidden units), and the other with three hidden layers (500,500, and 1000 hidden units), as shown in Fig. 8. To estimate the model’s partition function, we also used 20,000 intermediate distributions spaced uniformly from 0 to 1. Table 1 shows that the estimates of the lower bound on the average test log-probability were -84.62 and -85.10 for the 2- and 3-layer Boltzmann Machines respectively. Observe that the two DBMs that contain over 0.9 and 1.15 million parameters do not appear to suffer much from overfitting. The difference between the estimates of the training and test log-probabilities was about 1 nat. Figure 7 further shows samples generated from all three models by randomly initializing all binary states and running the Gibbs sampler for 100,000 steps. Certainly, all samples look like the real handwritten digits. We also emphasize that without greedy pre-training, we could not successfully learn good DBM models of MNIST digits.

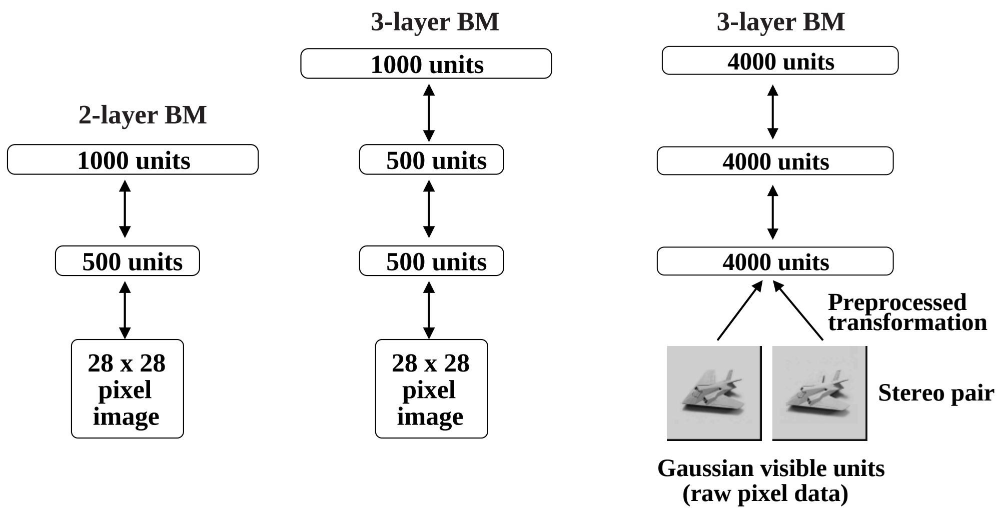

Figure 8: **Left:** The architectures of two Deep Boltzmann Machines used in MNIST experiments. **Right:** The architecture of Deep Boltzmann Machine used in NORB experiments.

To estimate how loose the variational bound is, we randomly sampled 100 test cases, 10 of each class, and ran AIS to estimate the true test log-probability[6] for the 2-layer Boltzmann Machine. The estimate of the variational bound was -83.35 per test case, whereas the estimate of the true test log-probability was -82.86. The difference of about 0.5 nats shows that the bound is rather tight.

For a simple comparison we also trained several mixture of Bernoullis models with 10, 100, and 500 components. The corresponding average test log-probabilities were -168.95, -142.63, and -137.64. Compared to DBMs, a mixture of Bernoullis performs very badly. The difference of over 50 nats per test case is striking.

Finally, after discriminative fine-tuning, the two-hidden-layer Boltzmann Machine achieves an error rate of 0.95% on the full MNIST test set. This is, to our knowledge, the best published result on the permutation-invariant version of the MNIST task[7]. The 3-layer BM gives a slightly worse error rate of 1.01%. The flat BM, on the other hand, gives considerably worse error rate of 1.27%. This is compared to 1.4% achieved by SVMs (Decoste and Schölkopf (2002)), 1.6% achieved by randomly initialized backprop, 1.2% achieved by the Deep Belief Network, described in (Hinton et al. (2006); Hinton and Salakhutdinov (2006)), and 0.97% obtained by using a combination of discriminative and generative fine-tuning on the same DBN (Hinton (2007)).

To test discriminative performance of DBMs when the number of labeled examples is small, we randomly sampled 1%, 5%, and 10% of the handwritten digits in each class and treated them as labeled data. Table 2 shows that after discriminative fine-tuning, a two-hidden-layer BM achieves error rates of 4.82%, 2.72%, and 2.46%. Deep Boltzmann Machines clearly outperform regularized nonlinear NCA (Salakhutdinov and Hinton (2007)), as well as linear NCA (Goldberger et al. (2004)), an autoencoder (Hinton and Salakhutdinov (2006)), and K-nearest neighbours, particularly when the number of labeled examples is only 600.

6. Note that computationally, this is equivalent to estimating 100 partition functions.
7. In the permutation-invariant version, the pixels of every image are subjected to the same, random permutation which makes it hard to use prior knowledge about images.

Table 2: Classification error rates on MNIST test set when only a small fraction of labeled data is available.

	Two-Layer DBM	Regularized Nonlinear NCA	Linear NCA	Autoencoder	KNN
1% (600)	4.82%	8.81%	19.37%	9.62%	13.74%
5% (3000)	2.72%	3.24%	7.23%	5.18%	7.19%
10% (6000)	2.46%	2.58%	4.89%	4.46%	5.87%
100% (60000)	0.95%	1.00%	2.45%	2.41%	3.09%

5.2 NORB

Results on MNIST show that Deep Boltzmann Machines can significantly outperform many other models on the well-studied but relatively simple task of handwritten digit recognition. In this section we present results on NORB, which is a considerably more difficult dataset than MNIST. NORB (LeCun et al. (2004)) contains images of 50 different 3D toy objects with 10 objects in each of five generic classes: cars, trucks, planes, animals, and humans. Each object is photographed from different viewpoints and under various lighting conditions. The training set contains 24,300 stereo image pairs of 25 objects, 5 per class, while the test set contains 24,300 stereo pairs of the remaining, different 25 objects. The goal is to classify each previously unseen object into its generic class. From the training data, 4,300 were set aside for validation.

Each image has 96×96 pixels with integer greyscale values in the range [0,255]. To speed-up experiments, we reduced the dimensionality by using a foveal representation of each image in a stereo pair. The central 64×64 portion of an image is kept at its original resolution. The remaining 16 pixel-wide ring around it is compressed by replacing non-overlapping square blocks of pixels in the ring with a single scalar given by the average pixel-value of a block. We split the ring into four smaller ones: the outermost ring consists of 8×8 blocks, followed by a ring of 4×4 blocks, and finally two innermost rings of 2×2 blocks. The resulting dimensionality of each training vector, representing a stereo pair, was $2 \times 4488 = 8976$. A random sample from the training data used in our experiments is shown in Fig. 9, left panel.

To model raw pixel data, we use an RBM with Gaussian visible and binary hidden units. Gaussian RBMs have been previously successfully applied for modeling greyscale images, such as images of faces (Hinton and Salakhutdinov (2006)). However, learning an RBM with Gaussian units can be slow, particularly when the input dimensionality is quite large. Here we follow the approach of Nair and Hinton (2009) by first learning a Gaussian RBM and then treating the activities of its hidden layer as "preprocessed" data. Effectively, the learned low-level RBM acts as a preprocessor that converts greyscale pixels into a binary representation, which we then use for learning a Deep Boltzmann Machine.

The number of hidden units for the preprocessing RBM was set to 4000 and the model was trained using Contrastive Divergence learning for 500 epochs. We then trained a two-hidden-layer DBM with each layer containing 4000 hidden units, as shown in Fig. 8, right panel. Note that the entire model was trained in a completely unsupervised way. After the subsequent discriminative fine-tuning, the "unrolled" DBM achieves a misclassification error rate of 10.8% on the full test set. This is compared to 11.6% achieved by SVMs (Bengio and LeCun (2007a)), 22.5% achieved

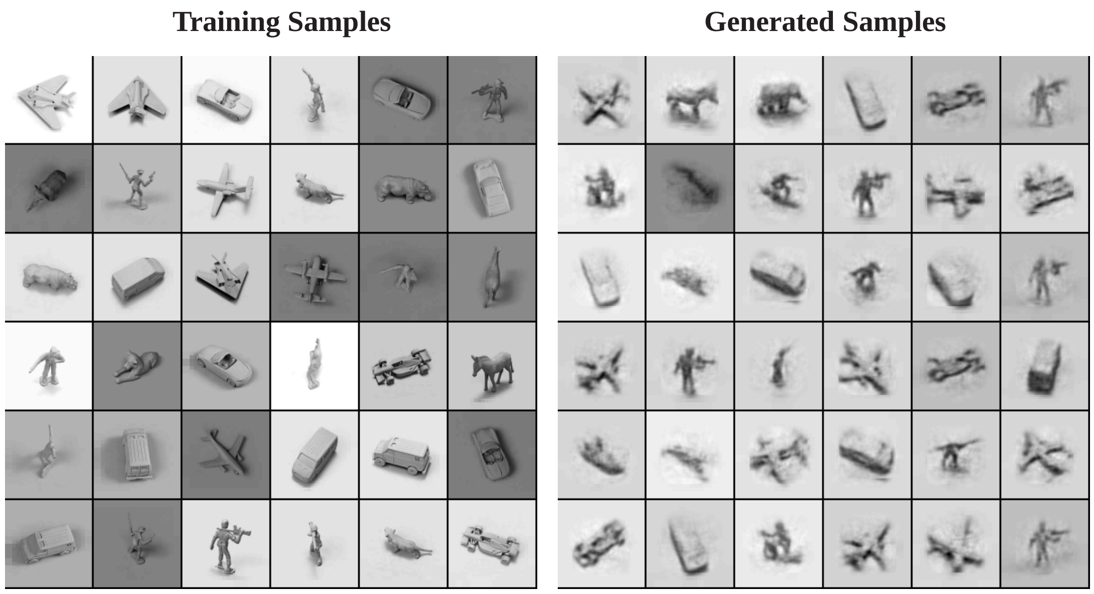

Figure 9: Random samples from the training set, and samples generated from a three-hidden-layer Deep Boltzmann Machine by running the Gibbs sampler for 10,000 steps.

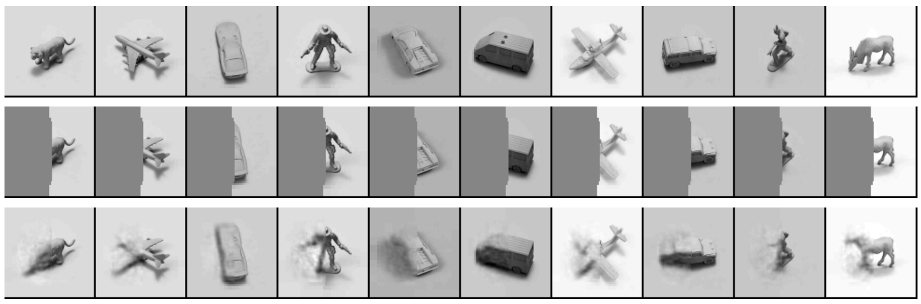

Figure 10: Performance of the three-hidden-layer DBM on the image inpainting task. **Top:** Ten objects randomly sampled from the test set. **Middle:** Partially occluded input images. **Bottom:** Inferred images were generated by running a Gibbs sampler for 1000 steps.

by logistic regression, and 18.4% achieved by the K-nearest neighbours (LeCun et al. (2004)). To show that DBMs can benefit from additional *unlabeled* training data, we augmented the training data with additional unlabeled data by applying simple pixel translations, creating a total of 1,166,400 training instances[8]. After learning a good generative model, the discriminative fine-tuning (using only the 24,300 labeled training examples without any translation) reduces the misclassification error down to 7.2%. Figure 9 shows samples generated from the model by running prolonged Gibbs sampling. Note that the model was able to capture a lot of regularities in this high-dimensional, richly structured data, including different object classes, various viewpoints and lighting conditions.

8. We thank Vinod Nair for sharing his code for blurring and translating NORB images.

Finally, we tested the ability of the DBM to perform an image inpainting task. To this end, we randomly selected 10 objects from the *test set* and simulated the occlusion by zeroing out the left half of the image (see Fig. 10). We emphasize that the test objects are different from the training objects (i.e. the model never sees images of ’cowboy’, but it sees other images belonging to the ’person’ category). We next sampled the “missing” pixels conditioned on the non-occluded pixels of the image using 1000 Gibbs updates. Figure 10, bottom row, shows that the model was able to coherently infer occluded parts of the test images. In particular, observe that even though the model never sees an image of the cowboy, it correctly infers that it should have two legs and two arms.

Surprisingly, even though the Deep Boltzmann Machine contains about 68 million parameters, it significantly outperforms many of the competing models. Clearly, unsupervised learning helps generalization because it ensures that most of the information in the model parameters comes from modeling the input data. The very limited information in the labels is used only to slightly adjust the layers of features already discovered by the Deep Boltzmann Machine.

6. Discussion

A major difference between DBNs and DBMs is that the procedure for adding an extra layer to a DBN replaces the whole prior over the previous top layer whereas the procedure for adding an extra layer to a DBM only replaces half of the prior. So in a DBM, the weights of the bottom level RBM end up doing much more of the work than in a DBN where the weights are only used to define $p(\mathbf{v}|\mathbf{h}^1; W^1)$ (in the composite generative model). This suggests that adding layers to a DBM will give diminishing improvements in the variational bound much more quickly than adding layers to a DBN. There is, however, a simple way to pre-train a DBM so that more of the modeling work is left to the higher layers.

Suppose we train an RBM with one set of hidden units and four sets of visible units and we constrain the four weight matrices (and visible biases) to be identical. Then we use the hidden activities as data to train an RBM with one set of visible units and four sets of hidden units, again constrained to have identical weight matrices. Now we can combine the two RBMs into a DBM with two hidden layers by using one set of visible units from the first RBM and three of the four sets of hidden units from the second RBM. In this DBM, $^3/_4$ of the first RBM’s prior over the first hidden layer has been replaced by the prior defined by the second RBM. It remains to be seen whether this makes DBMs work better.

Using multiple copies of a layer with identical weights and biases is exactly equivalent to using a single copy but taking multiple samples from the distribution over that layer. Now we can imagine taking, say, $1^1/_3$ samples by taking 2 samples and randomly keeping the second one with a probability of $^1/_3$[9]. Suppose we train the bottom-level RBM by taking 4 samples from its visible units and 1 sample from its hidden units. We then train a higher level RBM on the aggregated posterior of the first RBM taking $1^1/_3$ samples from its hidden units and 1 sample from its “visible” units. Then we compose the two RBMs without changing their weights, but taking only one sample from each layer in the composite model. Again, we have a DBM in which the higher level RBM has replaced $^3/_4$ of the prior defined by the lower level RBM. The higher level RBM will learn a different model using $1^1/_3$ samples than it would have learned using 4 samples, but both methods of replacing $^3/_4$ of the prior are valid. It remains to be seen which works best.

9. The random decisions should be made independently for each unit to avoid high variance in the total output of a layer.

In this paper we have focussed on Boltzmann machines with binary units. The learning methods we have described can be extended to learn Deep Boltzmann machines built with RBM modules that contain real-valued (Marks and Movellan (2001)), count (Salakhutdinov and Hinton (2009b)), or tabular data provided the distributions are in the exponential family (Welling et al. (2005)). However, it often requires additional insights to get the basic RBM learning module to work well with non-binary units. For example, it ought to be possible to learn the variance of the noise model of the visible units in a Gaussian-Bernoulli RBM, but this is typically very difficult for reasons explained in Hinton (2010). For modeling the NORB data we used fixed variances of 1 which is clearly much too big for data that has been normalized so that the pixels have a variance of 1. Recent work shows that Gaussian visible units work much better with rectified linear hidden units (Nair and Hinton (2010)) and using this type of hidden unit it is straightforward to learn the variance of the noise model of each visible unit.

7. Summary

We presented a novel combination of variational and Markov Chain Monte Carlo algorithms for training Boltzmann Machines. When applied to pre-trained Deep Boltzmann Machines with several hidden layers and millions of weights, this combination is a very effective way to learn good generative models. We demonstrated the performance of the algorithm using the MNIST hand-written digits and the NORB stereo images of 3-D objects with highly variable viewpoint and lighting.

A simple variational approximation works well for estimating the data-dependent statistics because learning based on these estimates encourages the true posterior distributions over the hidden variables to be close to their variational approximations. Persistent Markov chains work well for estimating the data-independent statistics because learning based on these estimates encourages the persistent chains to explore the state space much more rapidly than would be predicted by their mixing rates.

Pre-training a stack of RBMs using contrastive divergence can be used to initialize the weights of a Deep Boltzmann Machine to sensible values. Unlike pre-training a DBN, the RBMs in the stack need to be trained with two copies of one or both layers of units. This makes it possible to take the square root of one or both of the marginal distributions over a layer of the trained RBM (by throwing away one copy of the other layer). The RBMs can then be composed to form a Deep Boltzmann Machine. The pre-training ensures that the variational inference can be initialized sensibly by a single bottom-up pass from the data-vector using twice the bottom-up weights to compensate for the lack of top-down input on the initial pass.

We have further showed how Annealed Importance Sampling, along with variational inference, can be used to estimate a variational lower bound on the log-probability that a Deep Boltzmann Machine assigns to test data. This allowed us to directly assess the performance of Deep Boltzmann Machines as generative models of data. Finally, we showed how to use a Deep Boltzmann Machine to initialize the weights of a feedforward neural network that can then be discriminatively fine-tuned. These networks give excellent discriminative performance, especially when there is very little labeled training data but a large supply of unlabeled data.

Acknowledgments

This research was supported by NSERC and by gifts from Google and Microsoft.

References

Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2009.

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

Y. Bengio and Y. LeCun. Scaling learning algorithms towards AI. *Large-Scale Kernel Machines*, 2007b.

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

A. R. Mohamed G. Dahl and G. E. Hinton. Deep belief networks for phone recognition. In *NIPS 22 workshop on deep learning for speech recognition*, 2009.

G. Dahl. Two deep learning architectures for acoustic modeling. In *Master's Thesis, Department of Computer Science, University of Toronto*, 2010.

D. Decoste and B. Schölkopf. Training invariant support vector machines. *Machine Learning*, 46 (1/3):161, 2002.

G. Desjardins, A. Courville, Y. Bengio, P. Vincent, and O. Delalleau. Tempered Markov chain Monte Carlo for training of restricted Boltzmann machines. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 13, 2010.

C. Galland. Learning in deterministic Boltzmann machine networks. In *PhD Thesis*, 1991.

S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 6(6):721–741, November 1984.

J. Goldberger, S. T. Roweis, G. E. Hinton, and R. R. Salakhutdinov. Neighbourhood components analysis. In *Advances in Neural Information Processing Systems*, 2004.

G. E. Hinton. To recognize shapes, first learn to generate images. *Computational Neuroscience: Theoretical Insights into Brain Function.*, 2007.

G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1711–1800, 2002.

G. E. Hinton. A practical guide to training restricted Boltzmann machines. Technical Report UTML TR 2010-003, Department of Computer Science, Machine Learning Group, University of Toronto, 2010.

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

G. E. Hinton and T. Sejnowski. Optimal perceptual inference. In *IEEE conference on Computer Vision and Pattern Recognition*, 1983.

G. E. Hinton and R. S. Zemel. Autoencoders, minimum description length and Helmholtz free energy. In *Advances in Neural Information Processing Systems*, volume 6, pages 3–10, 1994.

G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. In *Machine Learning*, volume 37, pages 183–233, 1999.

H.J. Kappen and F.B. Rodriguez. Boltzmann machine learning using mean field theory and linear response correction. In *Advances in Neural Information Processing Systems*, 1998.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220: 671–680, 1983.

Y. LeCun, F. J. Huang, and L. Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *CVPR (2)*, pages 97–104, 2004.

T. K. Marks and J. R. Movellan. Diffusion networks, product of experts, and factor analysis. In *Proc. Int. Conf. on Independent Component Analysis*, pages 481–485, 2001.

I. Murray and R. R. Salakhutdinov. Evaluating probabilities under high-dimensional latent variable models. In *Advances in Neural Information Processing Systems*, volume 21, 2009.

V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proc. 27th International Conference on Machine Learning*, 2010.

V. Nair and G. E. Hinton. Implicit mixtures of restricted Boltzmann machines. In *Advances in Neural Information Processing Systems*, volume 21, 2009.

R. M. Neal. Annealed importance sampling. *Statistics and Computing*, 11:125–139, 2001.

R. M. Neal. Connectionist learning of belief networks. *Artif. Intell*, 56(1):71–113, 1992.

R. M. Neal and G. E. Hinton. A view of the EM algorithm that justifies incremental, sparse and other variants. In M. I. Jordan, editor, *Learning in Graphical Models*, pages 355–368. Kluwer Academic Press, 1998.

S. Osindero and G. E. Hinton. Modeling image patches with a directed hierarchy of Markov random fields. In *Advances in Neural Information Processing Systems*, Cambridge, MA, 2008. MIT Press.

C. Peterson and J. R. Anderson. A mean field theory learning algorithm for neural networks. *Complex Systems*, 1:995–1019, 1987.

M. A. Ranzato, F. Huang, Y. Boureau, and Y. LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. *IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

H. Robbins and S. Monro. A stochastic approximation method. *Ann. Math. Stat.*, 22:400–407, 1951.

R. R. Salakhutdinov. Learning deep Boltzmann machines using adaptive MCMC. In *Proceedings of the International Conference on Machine Learning*, volume 27. ACM, 2010.

R. R. Salakhutdinov. Learning in Markov random fields using tempered transitions. In *Advances in Neural Information Processing Systems*, volume 22, 2009.

R. R. Salakhutdinov and G. E. Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 11, 2007.

R. R. Salakhutdinov and G. E. Hinton. Deep Boltzmann machines. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 12, 2009a.

R. R. Salakhutdinov and G. E. Hinton. Replicated softmax: An undirected topic model. In *Advances in Neural Information Processing Systems*, volume 22, 2009b.

R. R. Salakhutdinov and H. Larochelle. Efficient learning of deep Boltzmann machines. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 13, 2010.

R. R. Salakhutdinov and I. Murray. On the quantitative analysis of deep belief networks. In *Proceedings of the International Conference on Machine Learning*, volume 25, pages 872 – 879, 2008.

R. R. Salakhutdinov, A. Mnih, and G. E. Hinton. Restricted Boltzmann machines for collaborative filtering. In Zoubin Ghahramani, editor, *Proceedings of the International Conference on Machine Learning*, volume 24, pages 791–798. ACM, 2007.

T. Serre, A. Oliva, and T. A. Poggio. A feedforward architecture accounts for rapid categorization. *Proceedings of the National Academy of Sciences*, 104:6424–6429, 2007.

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

T. Tieleman. Training restricted Boltzmann machines using approximations to the likelihood gradient. In *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2008)*. ACM, 2008.

T. Tieleman and G.E. Hinton. Using Fast Weights to Improve Persistent Contrastive Divergence. In *Proceedings of the 26th international conference on Machine learning*, pages 1033–1040. ACM New York, NY, USA, 2009.

P. Vincent, H. Larochelle, Y. Bengio, and P. Manzagol. Extracting and composing robust features with denoising autoencoders. In William W. Cohen, Andrew McCallum, and Sam T. Roweis, editors, *Proceedings of the Twenty-Fifth International Conference*, volume 307, pages 1096–1103, 2008.

M. Welling. Herding dynamical weights to learn. In *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009.

M. Welling, M. Rosen-Zvi, and G. E. Hinton. Exponential family harmoniums with an application to information retrieval. In *Advances in Neural Information Processing Systems*, pages 1481–1488, Cambridge, MA, 2005. MIT Press.

C. Williams and F. Agakov. An analysis of contrastive divergence learning in gaussian boltzmann machines. In *Technical Report EDI-INF-RR-0120, Institute for Adaptive and Neural Computation, University of Edinburgh*, 2002.

L. Younes. On the convergence of Markovian stochastic algorithms with rapidly decreasing ergodicity rates, March 17 2000.

L. Younes. Parameter inference for imperfectly observed Gibbsian fields. *Probability Theory Rel. Fields*, 82:625–645, 1989.

A. L. Yuille. The convergence of contrastive divergences. In *Advances in Neural Information Processing Systems*, 2004.

R. S. Zemel. A minimum description length framework for unsupervised learning. In *Ph.D. Thesis, Department of Computer Science, University of Toronto*, 1993.