What kind of a graphical model is the brain?

Geoffrey E. Hinton
Canadian Institute for Advanced Research &
Department of Computer Science, University of Toronto
10 Kings College Road
Toronto, Canada M5S 3G4
hinton@cs.toronto.edu

Abstract

If neurons are treated as latent variables, our visual systems are non-linear, densely-connected graphical models containing billions of variables and thousands of billions of parameters. Current algorithms would have difficulty learning a graphical model of this scale. Starting with an algorithm that has difficulty learning more than a few thousand parameters, I describe a series of progressively better learning algorithms all of which are designed to run on neuron-like hardware. The latest member of this series can learn deep, multi-layer belief nets quite rapidly. It turns a generic network with three hidden layers and 1.7 million connections into a very good generative model of handwritten digits. After learning, the model gives classification performance that is comparable to the best discriminative methods.

1 Introduction

Our perceptual systems make sense of the visual input using a neural network that contains about 10^{13} synapses. There has been much debate about whether our perceptual abilities should be attributed to a few million generations of blind evolution or to a few hundred million seconds of visual experience. Evolutionary search suffers from an information bottleneck because fitness is a scalar, so my bet is that the main contribution of evolution was to endow us with a learning algorithm that could make use of high-dimensional gradient vectors. These vectors provide millions of bits of information every second thus allowing us to perform a much larger search in one lifetime than evolution could perform in our entire evolutionary history.

So what is this magic learning algorithm? I have been involved in attempts to answer this question using undirected graphical models [Hinton and Sejnowski, 1986], directed graphical models [Hinton *et al.*, 1995], or no graphical model at all [Rumelhart *et al.*, 1986]. These attempts have failed as scientific theories of how the brain learns because they simply do not work well enough. They have, however, produced two neat tricks, one for learning undirected models and one for learning directed models. In this paper, I describe some recent work in collaboration with Simon Osindero and Yee-Whye Teh that combines these two tricks in a surprising way to learn a hybrid generative model that was first proposed by Yee-Whye Teh. In this model, the top two layers form an *undirected* associative memory. The remaining layers form a directed acyclic graph that converts the representation in the associative memory into observable variables such as the pixels of an image. In addition to working well, this hybrid model has some other nice features:

1. The learning finds a fairly good model quickly even in deep directed networks with millions of parameters and many hidden layers. For optimal performance, however, a slower fine-tuning phase is required.
2. The learning algorithm builds a full generative model of the data which makes it easy to see what the distributed representations in the deeper layers have in mind.
3. The inference required for forming a percept is both fast and accurate.
4. The learning algorithm is unsupervised. For labeled data, it learns a model that generates both the label and the data.
5. The learning is local: adjustments to a synapse strength depend only on the states of the pre-synaptic and post-synaptic neuron.
6. The communication is simple: neurons only need to communicate their stochastic binary states.

Section 2 describes a simple learning algorithm for undirected, densely-connected, networks composed of stochastic binary variables some of which are unobserved. Section 3 shows how to make this simple algorithm efficient by restricting the architecture of the network. Section 4 introduces the idea of variational approximations for learning directed graphical models in which correct inference is intractable and describes the "wake-sleep" algorithm that makes use of a variational approximation in a multi-layer, directed network of stochastic binary variables. All of these sections can be safely ignored by people already familiar with these ideas.

Section 5 introduces the novel idea of a "complementary" prior. Complementary priors seem about as probable as father Christmas because, by definition, they exactly cancel the "explaining away" phenomenon that makes inference difficult in

directed models. Section 5.1 includes a simple example of a complementary prior and shows the equivalence between restricted Boltzmann machines and infinite directed networks with tied weights.

Section 6 introduces a fast, greedy learning algorithm for constructing multi-layer directed networks one layer at a time. Using a variational bound it shows that as each new layer is added, the overall generative model improves. The greedy algorithm resembles boosting in its repeated use of the same "weak" learner, but instead of re-weighting each data-vector to ensure that the next step learns something new, it re-represents it. Curiously, the weak learner that is used to construct deep directed nets is itself an undirected graphical model.

Section 7 shows how the weights produced by the efficient greedy algorithm can be fine-tuned using the "up-down" algorithm which is a contrastive version of the wake-sleep algorithm.

Section 8 shows the pattern recognition performance of a network with three hidden layers and about 1.7 million weights on the standard MNIST set of handwritten digits. When no knowledge of geometry is provided and there is no special preprocessing, the generalization performance of the network is 1.25% errors on the official test set. This beats the 1.5% achieved by the best back-propagation nets when they are not hand-crafted for this particular application, and it is quite close to the 1.1% or 1.0% achieved by the best support vector machines.

Finally, section 9 shows what happens in the mind of the model when it is running without being constrained by visual input. The network has a full generative model, so it is easy to look into its mind – we simply generate an image from its high-level representations.

Throughout the paper, we will consider nets composed of stochastic binary variables but the ideas can be generalized to other models in which the log probability of a variable is an additive function of the states of its directly-connected neighbours.

2 The Boltzmann machine learning algorithm

A Boltzmann machine (Hinton and Sejnowski, 1986) is a network of stochastic binary units with symmetric connections. It is usually divided into a set of "visible" units which can have data-vectors clamped on them, and a set of hidden units that act as latent variables (see figure 1). Each unit, i, adopts its "on" state with a probability that is a logistic function of the inputs it receives from other units, j:

$$p(s_i = 1) = \frac{1}{1 + \exp(-b_i - \sum_j s_j w_{ij})} \tag{1}$$

where b_i is the bias of unit i, and w_{ij} is the weight on the symmetric connection between i and j. The weights and biases of a Boltzmann machine define an energy function over global configurations (*i.e.* binary state vectors) of the net. Using α as an index over configurations of the visible units, and β for configurations of the hidden units:

$$E^{\alpha\beta} = -\sum_i b_i s_i^{\alpha\beta} - \sum_{i<j} s_i^{\alpha\beta} s_j^{\alpha\beta} w_{ij} \tag{2}$$

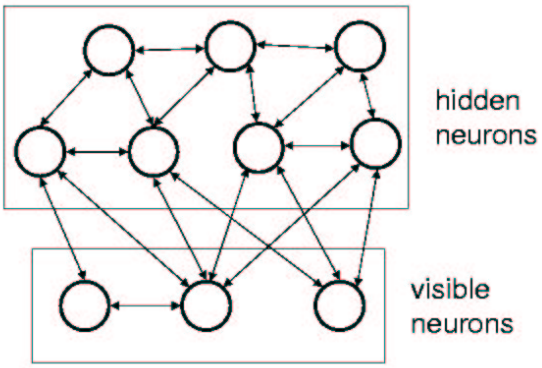

Figure 1: A Boltzmann machine composed of stochastic binary units with symmetric connections. When data is clamped on the visible units, a simple stochastic updating rule infers a configuration of states of the hidden units that is a good interpretation of the data. If no data is clamped and the same updating rule is used for all of the units, the network generates visible vectors from its model.

where $s_i^{\alpha\beta}$ is the binary state of unit i in the global binary configuration $\alpha\beta$. If units are chosen at random and their binary states are updated using the stochastic activation rule in Eq. 1 the Boltzmann machine will eventually converge to a stationary probability distribution in which the probability of finding it in any global state is determined by the energy of that state relative to the energies of all the other global states:

$$P^{\alpha\beta} = \frac{\exp(-E^{\alpha\beta})}{\sum_{\gamma\delta} \exp(-E^{\gamma\delta})} \tag{3}$$

If we sum over all configurations of the hidden units, we get the probability, at thermal equilibrium, of finding the visible units in configuration α

$$P^{\alpha} = \frac{\sum_{\beta} \exp(-E^{\alpha\beta})}{\sum_{\gamma\delta} \exp(-E^{\gamma\delta})} \tag{4}$$

A Boltzmann machine can be viewed as a generative model that assigns a probability, via Eq. 4, to each possible binary state vector over its visible units. By changing the weights and biases, we can change the probability that the model assigns to each possible visible vector. So we can model a set of training vectors by adjusting the weights and biases to maximize the sum of their log probabilities. A nice feature of Boltzmann machines is that the maximum likelihood learning rule for the weights is both simple and local. We can learn a locally optimal set of weights by collecting two sets of statistics:

- In the *positive phase* we clamp a training vector on the visible units and then repeatedly update the hidden units in random order using Eq. 1. Once the distribution over hidden configurations has reached "thermal" equilibrium with the clamped data-vector, we sample the hidden state and record which pairs of units are both

on. By repeating this for the entire training set, we can compute $< s_i s_j >^+$, the average correlation between i and j when data is clamped on the visible units.

- In the *negative phase* we let the network run freely with the visible units unclamped. Their states are updated in just the same way as the hidden units. Once the distribution over global configurations has reached equilibrium, we sample the states of all the units and record which pairs are both on. By repeating this many times, we can compute $< s_i s_j >^-$, the average correlation between i and j when the network is running freely and therefore producing samples from its generative model.

We can then follow the gradient of the log probability of the data by using the simple rule

$$\Delta w_{ij} = \epsilon(< s_i s_j >^+ - < s_i s_j >^-) \quad (5)$$

where ϵ is a learning rate. It is very surprising that the learning rule is so simple because the gradient of the log likelihood with respect to one weight depends in complicated ways on all the other weights. In the back-propagation algorithm, these dependencies are computed explicitly in the backward pass. In the Boltzmann machine they show up as the difference between the local correlations in the positive and negative phases.

Unfortunately, the simplicity and generality of the Boltzmann machine learning algorithm come at a price. It can take a very long time for the network to settle to thermal equilibrium, especially in the negative phase when it is unconstrained by data but needs to be highly multi-modal. Also, the gradient used for learning is very noisy because it is the difference of two noisy expectations. These problems make the general form of the algorithm impractical for large networks with many hidden units.

3 Restricted Boltzmann machines and contrastive divergence learning

If we are willing to restrict the architecture of a Boltzmann machine by not allowing connections between hidden units, the positive phase no longer requires any settling. With a data-vector clamped on the visible units, the hidden units are all conditionally independent, so we can apply the update rule in Eq. 1 to all the units at the same time to get an unbiased sample from the posterior distribution over hidden configurations. This makes it easy to measure the first correlation in Eq. 5.

If we also prohibit connections between visible units, we can update all of the visible units in parallel given a hidden configuration. So the second correlation in Eq. 5 can be found by alternating Gibbs sampling as shown in figure 2. Unfortunately we may need to run the alternating Gibbs sampling for a long time before the Markov chain converges to the equilibrium distribution. Fortunately, if we start the Markov chain at the data distribution, learning still works well even if we only run the chain for a few steps [Hinton, 2002]. This gives an efficient learning rule:

$$\Delta w_{ij} = \epsilon(< s_i s_j >^0 - < s_i s_j >^n) \quad (6)$$

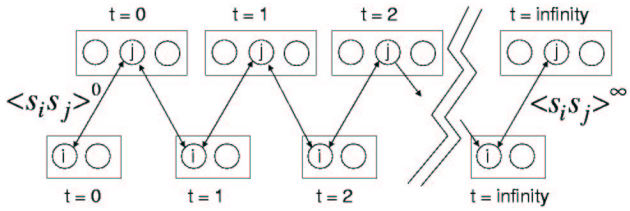

Figure 2: This depicts a Markov chain that uses alternating Gibbs sampling. In one full step of Gibbs sampling, the hidden units in the top layer are all updated in parallel by applying Eq. 1 to the inputs received from the the current states of the visible units in the bottom layer, then the visible units are all updated in parallel given the current hidden states. The chain is initialized by setting the binary states of the visible units to be the same as a data-vector. The correlations in the activities of a visible and a hidden unit are measured after the first update of the hidden units and again at the end of the chain. The difference of these two correlations provides the learning signal for updating the weight on the connection.

where the superscript 0 indicates that the correlation is measured at the start of the chain with a data-vector clamped on the visible units and n indicates that the correlation is measured after n full steps of Gibbs sampling. The angle brackets denote expectations over both the choice of data-vector and the stochastic updating used in the Gibbs sampling. This learning rule does not follow the gradient of the log likelihood, but it does closely approximate the gradient of another function, contrastive divergence, which is the difference of two Kullback-Leibler divergences [Hinton, 2002]. Intuitively, it is not necessary to run the chain to equilibrium in order to see how the data distribution is being systematically distorted by the model. If we just run the chain for a few steps and then lower the energy of the data and raise the energy of whichever configuration the chain preferred to the data, we will make the model more likely to generate the data and less likely to generate the alternatives. An empirical investigation of the relationship between the maximum likelihood and the contrastive divergence learning rules can be found in [Carreira-Perpinan and Hinton, 2005].

Contrastive divergence learning in a restricted Boltzmann machine is efficient enough to be practical [Mayraz and Hinton, 2001]. Variations that use real-valued units and different sampling schemes are described in [Teh *et al.*, 2003] and have been quite successful for modeling the formation of topographic maps [Welling *et al.*, 2003] and for denoising natural images [Roth and Black, 2005]. However, it appears that the efficiency has been bought at a high price: It is not possible to have deep, multilayer nets because these take far too long even to reach *conditional* equilibrium with a clamped data-vector. Also, nets with symmetric connections do not give a causal model in which the data is explained in terms of underlying causes.

The next section describes a simple learning algorithm for an apparently quite different type of network that uses di-

rected connections. This learning algorithm also has deficiencies, but it can be combined with contrastive divergence learning in a surprising way to produce an algorithm that works much better and is significantly more similar to the real brain.

4 Variational learning

Inference in directed graphical models that use non-linear, distributed representations is difficult because of a phenomenon called "explaining away" [Pearl, 1988] which creates dependencies between hidden variables. This is illustrated in figure 3. Radford Neal [Neal, 1992] showed that it was possible to use Gibbs sampling to perform inference correctly in multilayer *directed* networks composed of the same type of binary stochastic units as are used in Boltzmann machines. The communication required is more complicated than in a Boltzmann machine because in addition to seeing the binary states of its ancestors and descendants, a unit needs to see the probability that each of its descendants would be turned on by the current states of all that descendant's ancestors. However, once we have a sample from the posterior distribution over configurations of the hidden units, the maximum likelihood learning rule for updating the directed connection from j to i is very simple:

$$\Delta w_{ji} = \epsilon s_j (s_i - \hat{s}_i) \tag{7}$$

where ϵ is a learning rate and $\hat{s}_i$ is the probability that i would be turned on by the current states of all its ancestors. There is no need for a "negative phase" because directed models do not require the awkward normalizing term that shows up in the denominator of Eq. 3. Radford Neal showed that logistic belief nets are somewhat easier to learn than Boltzmann machines, but the use of Gibbs sampling to get samples from the posterior distribution still makes it very tedious to learn large, deep nets.

Rich Zemel and I realised that it might still be possible to learn a belief net that contained a layer of binary stochastic hidden units even when the cost of computing the posterior distribution, or sampling from it, was prohibitive. Instead of trying to perform maximum likelihood learning, we adopted a coding perspective and attempted to learn a model that would minimize the description length of the data [Zemel and Hinton, 1995]. The idea is that the sender and receiver both have access to the model and instead of communicating a data-vector directly, the sender first communicates a hidden configuration of the model. This costs some bits, but it also gives the receiver a good idea of what data to expect. Given these expectations, the data-vector can be communicated more cheaply[1]. So it appears that the expected cost of communicating a data-vector is:

$$C(x) = -\sum_{\alpha} Q(y_\alpha|x) \left[\log p(y_\alpha) + \log p(x|y_\alpha)\right] \tag{8}$$

[1]Shannon showed that, using an efficient block coding scheme, the cost of communicating a discrete value to a receiver is asymptotically equal to the negative log probability of that value under a probability distribution that has already been agreed upon by the sender and the receiver.

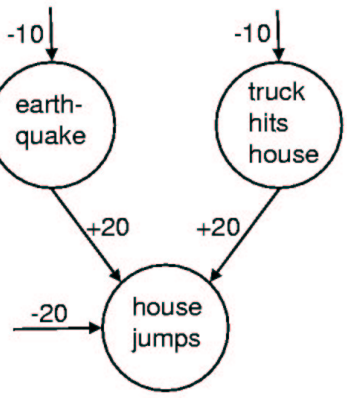

Figure 3: A simple logistic belief net containing two independent, rare causes that become highly anti-correlated when we observe the house jumping. The bias of -10 on the earthquake node means that, in the absence of any observation, this node is e^{10} times more likely to be off than on. If the earthquake node is on and the truck node is off, the jump node has a total input of 0 which means that it has an even chance of being on. This is a much better explanation of the observation that the house jumped than the odds of e^{-20} which apply if neither of the hidden causes is active. But it is wasteful to turn on both hidden causes to explain the observation because the odds on them both happening are $e^{-10} \times e^{-10} = e^{-20}$.

where y_α is a configuration of the hidden units, $Q(y_\alpha|x)$ is the probability that the sender will choose to use y_α in order to communicate data-vector x, $\log p(y_\alpha)$ is the cost of communicating y_α to a receiver who already has the model and $\log p(x|y_\alpha)$ is the cost of communicating x to a receiver who has both the model and the hidden configuration y_α. Rich soon discovered that it was better to minimize a different function and we eventually understood why.

Suppose there are two different hidden configurations that give the same communication cost for a data-vector. The sender can flip a coin to decide which one to use. After receiving the data-vector, the receiver can figure out what choices were available to the sender and he can therefore figure out the value of the random bit produced by the coin. So if there are two equally good hidden configurations, the sender can communicate one additional bit from a random bit stream by his choice of configuration. In general, the number of extra bits is equal to the entropy of the sender's distribution across hidden configurations. All of these extra bits can be used to communicate some other bit string, so we need to subtract them from the communication cost of the data-vector:

$$\begin{aligned} C(x) &= -\sum_{\alpha} Q(y_\alpha|x) \left[\log p(y_\alpha) + \log p(x|y_\alpha)\right] \\ &\quad - \left(-\sum_{\alpha} Q(y_\alpha|x) \log Q(y_\alpha|x)\right) \end{aligned} \tag{9}$$

If the sender picks hidden configurations from their true posterior distribution, the communication cost is minimized and is equal to the negative log probability of the data-vector

under the model. But if it is hard to sample from the true posterior, the sender can use any other distribution he chooses. The communication cost goes up, but it is still perfectly well-defined. The sender could, for example, insist on using a factorial distribution in which the states of the hidden units are chosen independently. The communication cost will then be an upper bound on the negative log probability of the data under the model and by minimizing the communication cost we will either push down the negative log probability of the data or we will make the bound tighter. The looseness of the bound is just the Kullback-Liebler divergence between the distribution used by the sender and the true posterior, $P(y_\alpha|x)$.

$$-\log p(x) = C(x) - \sum_\alpha Q(y_\alpha|x) \log \frac{Q(y_\alpha|x)}{p(y_\alpha|x)} \qquad (10)$$

The use of an approximate posterior distribution to bound $\log p(x)$ [Neal and Hinton, 1998] is now a standard way to learn belief nets in which inference is intractable [Jordan *et al.*, 1999].

4.1 The wake-sleep algorithm

A simple way to make use of variational learning in a multilayer logistic belief net is to use a set of "recognition" connections that compute a factorial approximation to the posterior distribution in one layer when given the binary states of the units in the layer below [Hinton *et al.*, 1995]. These recognition connections will not, in general, have the same values as the corresponding generative connections. Given a set of recognition weights, it is easy to update the generative weights to follow the gradient of the description cost in Eq. 9. We use a data-vector to set the states of the visible units and then we use the recognition connections to compute a probability for each unit in the first hidden layer. Then we use these probabilities to pick independent binary states for all the units in that layer. This is repeated for each hidden layer in turn until we have a sample from the approximate posterior. Given this sample the learning rule for the generative, top-down weights is given by Eq. 7. This is the "wake" phase of the wake-sleep algorithm.

The "correct" way to learn the recognition weights is to follow the derivative of the cost in Eq. 9. The recognition weights only affect the Q terms; they do not affect the p terms. However, the derivatives *w.r.t* the Q terms are messy because Q comes outside the $\log$. So we make a further approximation. In the "sleep" phase, we perform an ancestral pass to generate a sample from the generative model. Starting at the top layer, we pick binary states for the units from their independent priors. Then we pick states for the units in each lower layer using the probabilities computed by applying the generative weights to the states in the layer above. Once we have completed an ancestral pass, we have both a visible vector and its true hidden causes. So we can adjust the recognition weights to be better at recovering the hidden causes from the states of the units in the layer below:

$$\Delta w_{ij} = \epsilon s_i (s_j - \hat{s}_j) \qquad (11)$$

where $\hat{s}_j$ is the probability that j would be turned on by the current states of all its descendants.

The wake-sleep algorithm works quite well, but the sleep phase is not exactly following the gradient of the variational bound. As a result, it does the wrong thing when a data-vector can be generated by two quite different hidden configurations: Instead of picking one of these hidden configurations and sticking with it, it averages the configurations to produce a vague factorial distribution that gives significant probability to many poor configurations.

5 Complementary priors

The phenomenon of explaining away makes inference difficult in directed networks. It is comforting that we can still improve the parameters even when inference is done incorrectly, but it would be much better to find a way of eliminating explaining away altogether, even in models whose hidden variables have highly correlated effects on the visible variables. Most people who use directed graphical models regard this as impossible.

In a logistic belief net with one hidden layer, the prior distribution over the hidden variables is factorial because their binary states are chosen independently when the model is used to generate data. The non-independence in the posterior distribution is created by the likelihood term coming from the data. Perhaps we could eliminate explaining away in the first hidden layer by using extra hidden layers to create a "complementary" prior that has exactly the opposite correlations to those in the likelihood term. Then, when the likelihood term is multiplied by the prior, we will get a posterior that is exactly factorial. This seems pretty implausible, but figure 4 shows a simple example of a logistic belief net with replicated weights in which the priors are complementary *at every hidden layer*. This net has some interesting properties.

5.1 An infinite directed model with tied weights

We can generate data from the infinite directed net by starting with a random configuration at an infinitely deep hidden layer and then performing an ancestral pass all the way down to the visible variables. Clearly, the distribution that we will get over the visible variables is exactly the same as the distribution produced by the Markov chain in figure 2 so the infinite directed net with tied weights is equivalent to a restricted Boltzmann machine.[2]

We can sample from the true posterior distribution over all of the hidden layers of the infinite directed net by starting with a data vector on the visible units and then using the transposed weight matrices to infer the factorial distributions over each hidden layer in turn. At each hidden layer we sample from the factorial posterior before computing the factorial posterior for the layer above. This is exactly the same process as starting a restricted Boltzmann machine at the data and letting it settle to equilibrium. It is also exactly the same as the inference procedure used in the wake-sleep algorithm, but in this net it gives unbiased samples because the complementary prior at each layer ensures that the posterior distribution really is factorial.

[2] We can interpret any undirected model that uses Gibbs sampling as an infinite directed model with tied weights.

Since we can easily sample from the true posterior, it is easy to learn the weights in the infinite directed net. Let us start by computing the derivative for a generative weight, w_{ij}^{00}, from a unit j in layer H_0 to unit i in layer V_0 (see figure 4). In a logistic belief net, the maximum likelihood learning rule for a single data-vector, x, is:

$$\frac{\partial \log p(x)}{\partial w_{ij}^{00}} = <s_j^0(s_i^0 - \hat{s}_i^0)> \tag{12}$$

where $< \cdot >$ denotes an average over the sampled states and $\hat{s}_i^0$ is the probability that unit i would be turned on if the visible vector was stochastically reconstructed from the sampled hidden states. Computing the posterior distribution over the second hidden layer, V_1, from the sampled binary states in the first hidden layer, H_0, is exactly the same process as reconstructing the data, so s_i^1 is a sample from $\hat{s}_i^0$ and the learning rule becomes:

$$\frac{\partial \log p(x)}{\partial w_{ij}^{00}} = <s_j^0(s_i^0 - s_i^1)> \tag{13}$$

Since the weights are replicated, the full derivative for a generative weight is obtained by summing the derivatives of the generative weights between all pairs of layers:

$$\begin{aligned}\frac{\partial \log p(x)}{\partial w_{ij}} &= <s_j^0(s_i^0 - s_i^1)> \\ &\quad + <s_i^1(s_j^0 - s_j^1)> \\ &\quad + <s_j^1(s_i^1 - s_i^2)> \\ &\quad + \ldots\end{aligned}$$

All of the vertically aligned terms cancel leaving the Boltzmann machine learning rule of Eq. 5. The same weights are also used for inference and one might expect this to contribute extra derivatives. Fortunately, all of these derivatives are zero. The inference is exact so, to first order, small changes in the inferred posteriors make no change in the log probability of the data. The only reason the recognition weights ever change is because they are tied to the generative weights.

6 A greedy learning algorithm for transforming representations

An efficient way to learn a complicated model is to combine a set of simpler models that are learned sequentially. To force each model in the sequence to learn something different from the previous models, the data is modified in some way after each model has been learned. In boosting [Freund, 1995], each model in the sequence is trained on re-weighted data that emphasizes the cases that the preceding models got wrong. In one version of principal components analysis, the variance in a modeled direction is removed thus forcing the next modeled direction to lie in the orthogonal subspace. In projection pursuit [Friedman and Stuetzle, 1981], the data is transformed by nonlinearly distorting one direction in the data-space to remove all non-Gaussianity in that direction. The idea behind our greedy algorithm is to allow each model in the sequence to receive a different representation of the data. The model

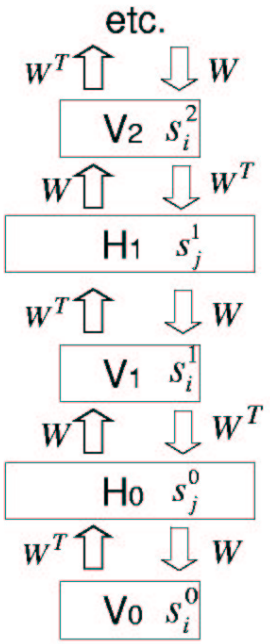

Figure 4: An infinite logistic belief net with tied weights.

performs a non-linear transformation on its input vectors and produces as output the vectors that will be used as input for the next model in the sequence.

Figure 5 shows a multilayer generative model in which the top two layers interact via undirected connections and all of the other connections are directed. There are no intra-layer connections and, to simplify the analysis, all layers have the same number of units. It is possible to learn sensible (though not optimal) values for the parameters W_0 by assuming that the parameters between higher layers will be used to construct a complimentary prior for W_0. This is equivalent to assuming that all of the weight matrices are constrained to be equal. The task of learning W_0 under this assumption reduces to the task of learning an RBM and although this is still difficult, good approximate solutions can be found rapidly by minimizing contrastive divergence (Hinton, 2002). Once W_0 has been learned, the data can be mapped through W_0^T to create higher-level "data" at the first hidden layer.

If the RBM is a perfect model of the original data, the higher-level "data" will already be modeled perfectly by the higher-level weight matrices. Generally, however, the RBM will not be able to model the original data perfectly and we can make the generative model better using the following greedy algorithm:

1. Learn W_0 assuming all the weight matrices are tied.
2. Freeze W_0 and commit ourselves to using W_0^T to infer factorial approximate posterior distributions over the states of the variables in the first hidden layer.
3. Keeping all the higher weight matrices tied to each other, but untied from W_0, learn an RBM model of the

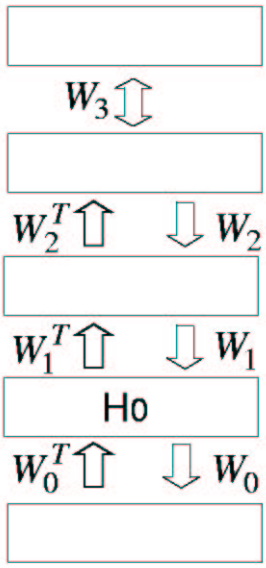

Figure 5: A hybrid network. The top two layers have undirected connections and form an associative memory. The layers below have directed, top-down, generative connections that can be used to map a state of the associative memory to an image. There are also directed, bottom-up, recognition connections that are used to infer a factorial representation in one layer from the binary activities in the layer below. In the greedy initial learning the recognition connections are tied to the generative connections.

higher-level "data" that is produced by using W_0^T to transform the original data.

If this greedy algorithm changes the higher-level weight matrices, it is guaranteed to improve the generative model. The log probability of a single data-vector, x, under the multilayer generative model is bounded by

$$\log p(x) \geq \sum_{\alpha} Q(y_\alpha|x)\left(\log p(y_\alpha) + \log p(x|y_\alpha)\right) - \sum_{\alpha} Q(y_\alpha|x)\log Q(y_\alpha|x)$$

where y_α is a binary configuration of the units in the first hidden layer, $p(y_\alpha)$ is the prior probability of y_α under the model defined by the weights above H_0 and $Q(\cdot|x)$ is any probability distribution over y. The bound becomes an equality if and only if $Q(\cdot|x)$ is the true posterior distribution.

When all of the weight matrices are tied together, the factorial distribution over H_0 produced by applying W_0^T to a data-vector is the true posterior distribution, so at step 2 of the greedy algorithm $\log p(x)$ is equal to the bound. Step 2 freezes both $Q(\cdot|x)$ and $p(x|y_\alpha)$ and with these terms fixed, the derivative of the bound is the same as the derivative of

$$\sum_{\alpha} Q(y_\alpha|x)\log p(y_\alpha) \tag{14}$$

So maximizing the bound *w.r.t.* the weights in the higher layers is exactly equivalent to maximizing the log probability of a dataset in which y_α occurs with probability $Q(y_\alpha|x)$. If the bound becomes tighter, it is possible for $\log p(x)$ to fall even though the lower bound on it increases, but $\log p(x)$ can never fall below its value at step 2 of the greedy algorithm because the bound is tight at this point and the bound always increases.

The greedy algorithm can clearly be applied recursively, so if we use the full maximum likelihood Boltzmann machine learning algorithm to learn each set of tied weights and then we untie the bottom layer of the set from the weights above, we can learn the weights one layer at a time with a guarantee[3] that we will never decrease the log probability of the data under the full generative model. In practice, we replace maximum likelihood Boltzmann machine learning algorithm by contrastive divergence learning because it works well and is much faster. The use of contrastive divergence voids the guarantee, but it is still reassuring to know that extra layers are guaranteed to improve imperfect models if we learn each layer with sufficient patience.

To guarantee that the generative model is improved by greedily learning more layers, it is convenient to consider models in which all layers are the same size so that the higher-level weights can be initialized to the values learned before they are untied from the weights in the layer below. The same greedy algorithm, however, can be applied even when the layers are different sizes.

7 Back-Fitting with the up-down algorithm

Learning the weight matrices one layer at a time is efficient but far from optimal. Once the weights in higher layers have been learned, neither the weights nor the simple inference procedure are optimal for the lower layers. The suboptimality produced by greedy learning is relatively innocuous for supervised methods like boosting. Labels are often scarce and each label may only provide a few bits of constraint on the parameters, so over-fitting is typically more of a problem than under-fitting. Going back and refitting the earlier models may, therefore, cause more harm than good. Unsupervised methods, however, can use very large unlabeled datasets and each case may be very high-dimensional thus providing many bits of constraint on a generative model. Under-fitting is then a serious problem which can be alleviated by a subsequent stage of back-fitting in which the weights that were learned first are revised to fit in better with the weights that were learned later.

After greedily learning good initial values for the weights in every layer, we untie the "recognition" weights that are used for inference from the "generative" weights that define the model, but retain the restriction that the posterior in each layer must be approximated by a factorial distribution in which the variables within a layer are conditionally independent given the values of the variables in the layer below. A variant of the wake-sleep algorithm described in section 4.1 can then be used to allow the higher-level weights to influence the lower level ones. In the "up-pass", the recognition weights are used in a bottom-up pass that stochastically picks

[3]The guarantee is on the *expected* change in the log probability.

a state for every hidden variable. The generative weights on the directed connections are then adjusted using the maximum likelihood learning rule in Eq. 7. Because weights are no longer tied to the weights above them, $\hat{s}_i$ must be computed using the states of the variables in the layer above i and the generative weights from these variables to i. The weights on the undirected connections at the top level are learned as before by fitting the top-level RBM to the posterior distribution of the penultimate layer.

The "down-pass" starts with a state of the top-level associative memory and uses the top-down generative connections to stochastically activate each lower layer in turn. During the down-pass, the top-level undirected connections and the generative directed connections are not changed. Only the bottom-up recognition weights are modified. This is equivalent to the sleep phase of the wake-sleep algorithm if the associative memory is allowed to settle to its equilibrium distribution before initiating the down-pass. But if the associative memory is initialized by an up-pass and then only allowed to run for a few iterations of alternating Gibbs sampling before initiating the down-pass, this is a "contrastive" form of the wake-sleep algorithm which eliminates the need to sample from the equilibrium distribution of the associative memory. The contrastive form also fixes several other problems of the sleep phase. It ensures that the recognition weights are being learned for representations that resemble those used for real data and it also helps to eliminate the problem of mode averaging. If, given a particular data vector, the current recognition weights always pick a particular mode at the level above and ignore other very different modes that are equally good at generating the data, the learning in the down-pass will not try to alter those recognition weights to recover any of the other modes as it would if the sleep phase used a pure ancestral pass.

By using a top-level associative memory we also eliminate a problem in the wake phase: Independent top-level units seem to be required to allow an ancestral pass, but they mean that the variational approximation is very poor for the top layer of weights.

8 Performance on the MNIST database

8.1 Training the network

The MNIST database of handwritten digits contains 60,000 training images and 10,000 test images. Results for many different pattern recognition techniques are already published for this database so it is ideal for evaluating new pattern recognition methods. The network[4] shown in figure 6 was trained on 44,000 of the training images that were divided into 440 balanced mini-batches each containing 10 examples of each digit class. The weights were updated after each mini-batch.

In the initial phase of training, the greedy algorithm described in section 6 was used to train each layer of weights

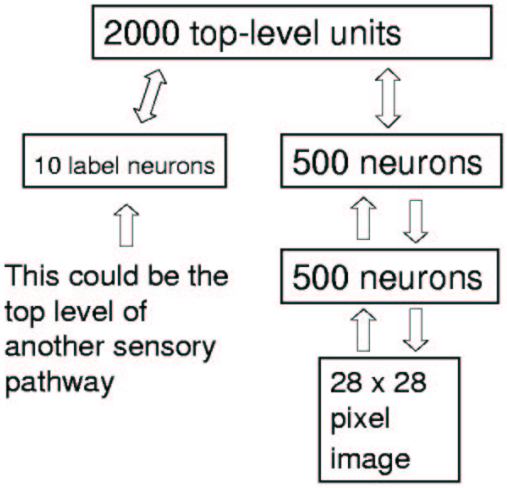

Figure 6: The network used to model the joint distribution of digit images and digit labels.

separately, starting at the bottom. Each layer was trained for 30 sweeps through the training set (called "epochs"). During training, the units in the "visible" layer of each RBM had real-valued activities between 0 and 1. These were the normalized pixel intensities when learning the the bottom layer of weights. For training higher layers of weights, the real-valued activities of the visible units in the RBM were the activation probabilities of the hidden units in the lower-level RBM. The hidden layer of each RBM used stochastic binary values when that RBM was being trained. The greedy training took a few hours in Matlab on a 3GHz Xeon processor and when it was done, the error-rate on the test set was 2.49% (see below for details of how the network is tested).

When training the top layer of weights (the ones in the associative memory) the labels were provided as part of the input. The labels were represented by turning on one unit in a "softmax" group of 10 units. When the activities in this group were reconstructed from the activities in the layer above, exactly one unit was allowed to be active and the probability of picking unit i was given by:

$$p_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{15}$$

where x_i is the total input received by unit i. Curiously, the learning rules are unaffected by the competition between units in a softmax group, so the synapses do not need to know which unit is competing with which other unit. The competition affects the probability of a unit turning on, but it is only this probability that affects the learning.

After the greedy layer-by-layer training, the network was trained, with a different learning rate and weight-decay, for 300 epochs using the up-down algorithm described in section 7. The learning rate, momentum, and weight-decay[5] were

[4] Preliminary experiments with 16×16 images of handwritten digits from the USPS database showed that a good way to model the joint distribution of digit images and their labels was to use an architecture of this type, but for 16×16 images, only $3/5$ as many units were used in each hidden layer.

[5] No attempt was made to use different learning rates or weight-decays for different layers and the learning rate and momentum were

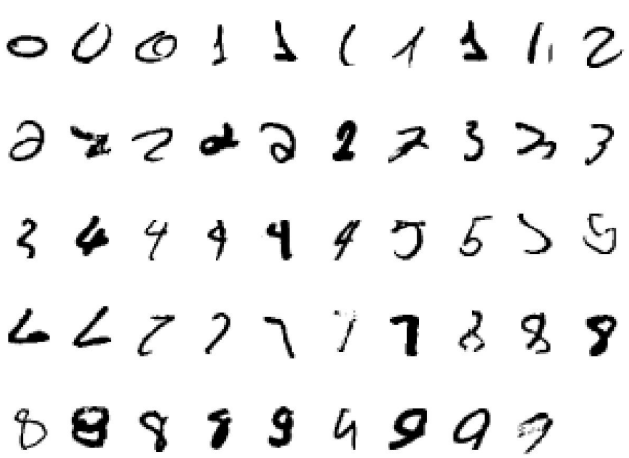

Figure 7: All 49 cases in which the network guessed right but had a second guess whose probability was within 0.3 of the probability of the best guess. The true classes are arranged in standard scan order.

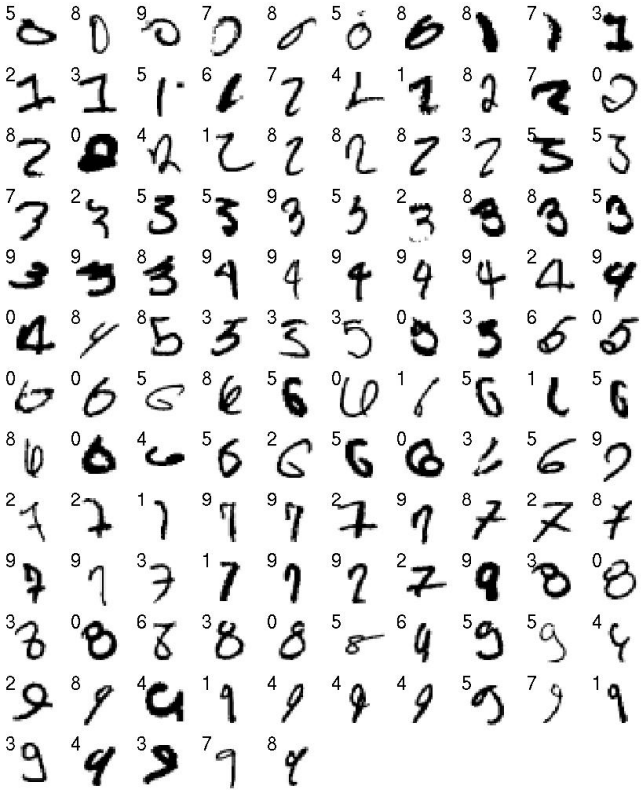

Figure 8: The 125 test cases that the network got wrong. Each case is labeled by the network's guess. The true classes are arranged in standard scan order.

chosen by training the network several times and observing its performance on a separate validation set of 10,000 images that were taken from the remainder of the full training set. For the first 100 epochs of the up-down algorithm, the up-pass was followed by three full iterations of alternating Gibbs sampling in the associative memory before performing the down-pass. For the second 100 epochs, six iterations were performed, and for the last 100 epochs, ten iterations were performed. Each time the number of iterations of Gibbs sampling was raised, the error on the validation set decreased noticeably.

The network that performed best on the validation set was then tested and had an error rate of 1.39%. This network was then trained on all 60,000 training images[6] until its error-rate on the full training set was as low as its final error-rate had been on the initial training set of 44,000 images. This took a further 59 epochs making the total learning time about a week. The final network had an error-rate of 1.25%. The errors made by the network are shown in figure 8. The 49 cases that the network gets correct but for which the second best probability is within 0.3 of the best probability are shown in figure 7.

The error-rate of 1.25% compares very favorably with the error-rates achieved by discriminative, feed-forward neural networks that have one or two hidden layers and are trained by back-propagation. When the detailed connectivity of these networks is not hand-crafted for this particular task, the best reported error-rate for stochastic on-line learning with a separate squared error on each of the 10 output units is 2.95%. These error-rates can be reduced to 1.51% by using small initial weights, "softmax" output units, a cross-entropy error function, and either very gentle learning (John Platt, personal communication) or a regularizer that penalizes the squared weights by an amount that is carefully chosen using a validation set. For comparison, nearest neighbor has a reported error rate (google MNIST) of 3.1% if all 60,000 training cases are used (which is extremely slow) and 4.4% if 20,000 are used. This can be reduced to 2.8% and 4.0% by using an L3 norm.

The only standard machine learning algorithm that outperforms the generative model is support vector machines which give error rates of 1.1% or 1.0%. But it is hard to see how support vector machines can make use of the domain-specific tricks, like weight-sharing and sub-sampling, which LeCun et.al. use to improve the performance of discriminative neural networks from 1.5% to 0.95%. There is no obvious reason why weight-sharing and sub-sampling cannot be used to reduce the error-rate of the generative model. Further improvements can always be achieved by averaging the opinions of multiple networks or by enhancing the training set with distorted data, but these techniques are available to all methods, though data enhancement can seriously slow-down methods that do not scale sub-linearly with the size of the training set.

always set quite conservatively to avoid oscillations. It is highly likely that the learning speed could be considerably improved by a more careful choice of learning parameters, though it is possible that this would lead to worse solutions.

[6]The training set has unequal numbers of each class, so images were assigned randomly to each of the 600 mini-batches.

Figure 9: Each row shows 10 samples from the generative model with a particular label clamped on. The top-level associative memory is run for 1000 iterations of alternating Gibbs sampling between samples.

Figure 10: Each row shows 10 samples from the generative model with a particular label clamped on. The top-level associative memory is initialized by an up-pass from a random binary image in which each pixel is on with a probability of 0.5. The first column shows the results of a down-pass from this initial high-level state. Subsequent columns are produced by 20 iterations of alternating Gibbs sampling in the associative memory.

8.2 Testing the network

One way to test the network is use a stochastic up-pass from the image to fix the binary states of the 500 units in the lower layer of the associative memory. With these states fixed, the label units are given initial real-valued activities of 0.1 and a few iterations of alternating Gibbs sampling are then used to activate the correct label unit. This method of testing gives error rates that are almost 1% higher than the rates reported above.

A better method is to first fix the binary states of the 500 units in the lower layer of the associative memory and to then turn on each of the label units in turn and compute the exact free energy of the resulting 510 component binary vector. Almost all the computation required is independent of which label unit is turned on [Teh and Hinton, 2001] and this method computes the exact conditional equilibrium distribution over labels instead of approximating it by Gibbs sampling which is what the previous method is doing. This method gives error rates that are about 0.5% higher than the ones quoted because of the stochastic decisions made in the up-pass. We can remove this noise in two ways. The simplest is to make the up-pass deterministic by using probabilities of activation in place of stochastic binary states. The second is to repeat the stochastic up-pass twenty times and average either the label probabilities or the label log probabilities over the twenty repetitions before picking the best one. The two types of average give almost identical results and these results are also very similar to using a deterministic up-pass, which was the method used for the reported results.

9 Looking into the mind of a neural network

To generate samples from the model, we perform alternating Gibbs sampling in the top-level associative memory until the Markov chain converges to the equilibrium distribution. Then we use a sample from this distribution as input to the layers below and generate an image by a single down-pass through the generative connections. If we clamp the label units to a particular class during the Gibbs sampling we can see images from the model's class-conditional distributions. Figure 9 shows a sequence of images for each class that were generated by allowing 1000 iterations of Gibbs sampling between samples.

We can also initialize the state of the top two layers by providing a random binary image as input. Figure 10 shows how the class-conditional state of the associative memory then evolves when it is allowed to run freely, but with the label clamped. This internal state is "observed" by performing a down-pass every 20 iterations to see what the associative memory has in mind. This use of the word "mind" is not metaphorical. A mental state is the state of a hypothetical world in which a high-level internal representation would constitute veridical perception. That hypothetical world is what the figure shows.

10 Conclusion

The network in figure 6 has about as many parameters as 0.002 cubic millimeters of mouse cortex. Several hundred networks of this complexity would fit within a single voxel of a high resolution fMRI scan. Learning algorithms are getting better, but they still have a long way to go.

Acknowledgments
The ideas presented in this paper came from research collaborations with Terry Sejnowski, Yann LeCun, Jay McClelleand, Radford Neal, Rich Zemel, Peter Dayan, Michael Jordan, Brendan Frey, Zoubin Ghahramani, Yee-Whye Teh, Max Welling, Simon Osindero and many other researchers. Andriy Mnih helped to prepare the manuscript. The research was supported by NSERC, the Gatsby Charitable Foundation, CFI and OIT. GEH holds a Canada Research Chair in machine learning.

References

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

[Freund, 1995] Y. Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 12(2):256 – 285, 1995.

[Friedman and Stuetzle, 1981] J.H. Friedman and W. Stuetzle. Projection pursuit regression. *Journal of the American Statistical Association*, 76:817–823, 1981.

[Hinton and Sejnowski, 1986] G. E. Hinton and T. J. Sejnowski. Learning and relearning in Boltzmann machines. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Volume 1: Foundations*, pages 282–317. MIT Press, Cambridge, 1986.

[Hinton *et al.*, 1995] G. E. Hinton, P. Dayan, B. J. Frey, and R. Neal. The wake-sleep algorithm for self-organizing neural networks. *Science*, 268:1158–1161, 1995.

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

[Jordan *et al.*, 1999] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. In M. I. Jordan, editor, *Learning in Graphical Models*. MIT Press, Cambridge, MA, 1999.

[Mayraz and Hinton, 2001] G. Mayraz and G. E. Hinton. Recognizing hand-written digits using hierarchical products of experts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:189–197, 2001.

[Neal and Hinton, 1998] R. M. Neal and G. E. Hinton. A new 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 Publishers, 1998.

[Neal, 1992] R. Neal. Connectionist learning of belief networks. *Artificial Intelligence*, 56:71–113, 1992.

[Pearl, 1988] J. Pearl. *Probabilistic Inference in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988.

[Roth and Black, 2005] S. Roth and M. J. Black. Fields of experts: A framework for learning image priors. In *IEEE Conf. on Computer Vision and Pattern Recognition*, June 2005.

[Rumelhart *et al.*, 1986] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[Teh and Hinton, 2001] Y.W. Teh and G. E. Hinton. Rate-coded restricted Boltzmann machines for face recognition. In *Advances in Neural Information Processing Systems*, volume 13, 2001.

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

[Welling *et al.*, 2003] M. Welling, G. Hinton, and S. Osindero. Learning sparse topographic representations with products of Student-t distributions. In S. Thrun S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 1359–1366. MIT Press, Cambridge, MA, 2003.

[Zemel and Hinton, 1995] R. S. Zemel and G. E. Hinton. Learning population codes by minimizing description length. *Neural Computation*, 7:549–564, 1995.