Where do features come from?

Geoffrey Hinton
Department of Computer Science, University of Toronto
6 King's College Rd, M5S 3G4, Canada
hinton@cs.toronto.edu

February 18, 2013

Abstract

It is possible to learn multiple layers of non-linear features by backpropagating error derivatives through a feedforward neural network. This is a very effective learning procedure when there is a huge amount of labeled training data, but for many learning tasks, very few labeled examples are available. In an effort to overcome the need for labeled data, several different generative models were developed that learned interesting features by modeling the higher-order statistical structure of a set of input vectors. One of these generative models, the restricted Boltzmann machine (RBM), has no connections between its hidden units and this makes perceptual inference and learning much simpler. More significantly, after a layer of hidden features has been learned, the activities of these features can be used as training data for another RBM. By applying this idea recursively it is possible to learn a deep hierarchy of progressively more complicated features without requiring any labeled data. This deep hierarchy can then be treated as a feedforward neural network which can be discriminatively fine-tuned using backpropagation. Using a stack of RBMs to initialize the weights of a feedforward neural network allows backpropagation to work effectively in much deeper networks and it leads to much better generalization. A stack of RBMs can also be used to initialize a deep Boltzmann machine that has many hidden layers. Combining this initialization method with a new method for fine-tuning the weights finally leads to the first efficient way of training Boltzmann machines with many hidden layers and millions of weights.

1 Introduction

The shape of an object, the layout of a scene, the sense of a word, and the meaning of a sentence must all be represented as spatio-temporal patterns of neural activity. The simplest way to represent things with neurons is to activate a single neuron in a large pool that contains one neuron for each possible thing that might need to be represented. This is obviously hopeless for the meaning of a sentence or the layout of a scene and it is fairly implausible for the shape of an object or the sense of a word. The alternative is to use a distributed representation in which each entity is represented by activity in many neurons and each neuron is involved in the representation of many different entities. If we model a neuron as a binary device that emits 1 or 0 spikes during a short time window, and if we assume that the precise time of a spike within the window is irrelevant, a distributed representation is just a set of binary features[1]. If we model a neuron as a device that can output an approximate real number [2] a distributed representation can be a set of noisy, real-valued features. Either way, a central question for both Psychology and Neuroscience is "where do these features come from?".

First we must dispose of the idea that features are innately specified. There are several reasons why this idea fails:

1. We have about 10^{14} synapses. Even if we treat these as binary and even if we only make use of 1% of their storage capacity to define all of the features we use, we still need to specify 10^{12} bits. There is no hope of packing this much information into our genes.

2. The world changes much too fast for innately specified features to keep up. If I tell you that she scromed him with the frying pan, you immediately have quite a large number of features for the word "scromed". Innately specified detectors for long wriggly things or for a red dot between two almost parallel lines may be a good way to avoid venomous snakes or to get a mother gull to regurgitate food, but for almost all of the perceptual and cognitive tasks for which features are useful, wired-in features cannot adapt nearly fast enough.

3. Evolution is much too slow to discover the millions of features we need.

[1] We shall return to the issue of spike timing at the end of the paper.

[2] This could be implemented by using a gang of similarly tuned real neurons to implement each model "neuron" or by using a rate code over a much longer time period.

> In very high-dimensional spaces, searches that have efficient access to gradient information are millions of times faster than searches that do not[3]. Evolution can optimize hundreds or even thousands of parameters, but it is hopelessly inefficient for optimizing millions of parameters because it cannot compute the gradient of the fitness of the phenotype with respect to heritable parameters. What evolution can do is explore the space of biological devices that *can* make effective use of gradient information. It can also explore the space of objective functions that these devices should optimize and the space of architectures in which this optimization works well.

There are several different ways to approach the question of what objective functions are being optimized by the brain and how it computes the gradients of these objective functions with respect to properties of synapses. We can investigate people's learning abilities without worrying about the hardware [50], we can investigate how real synapses change [31], or we can explore the space of synaptic learning rules that work well in large networks of neuron-like processors. Given enough computational power, we might even use an evolutionary outer loop to explore this space [56]. These approaches are complementary and clearly need to be pursued in parallel. It is impossible to know in advance whether the biologically unrealistic assumptions of a particular type of model neuron will prevent us from learning anything biologically relevant by studying how to get networks of those neurons to learn complex tasks. Similarly, it is impossible to know in advance whether neuroscience experiments to test computationally infeasible theories of learning will tell us anything interesting about how learning really occurs in the brain.

My approach is to try to find learning procedures that work really well for learning things that people are obviously very good at. Provided these procedures can run in neuron-like hardware, they should provide biologists with a much more sensible space of hypotheses. Most intuitively plausible learning procedures do not actually work very well in practice, particularly in large networks, and they can be filtered out without invading any real brains.

[3] It is always possible to use a local random search to estimate the gradient but in spaces with millions of dimensions, this is millions of times slower than methods like backpropagation that compute the gradient efficiently.

2 Learning distributed representations in 1986

In the mid 1980s, there were two exciting new algorithms for learning non-linear distributed representations in multiple layers of hidden units. Back-propagation [45, 55, 27] was a straightforward application of the chain rule for computing gradients in a deterministic feed-forward network (see figure 1). It looked pretty implausible as a model of learning in cortex because it required a lot of labeled training data. Some people thought it was also implausible because the "neurons" needed to send two quite different signals, one during the forward pass to communicate activities and one during the backward pass to communicate error-derivatives. Evolution, however, can produce teeth and eyeballs from the same stem cells, so it is hard to believe that it could fail to find a way to implement back-propagation in a few hundred million years if that was the best thing to do. Getting all of that labeled training data, however, seemed problematic.

[Figure 1 about here.]

The most promising suggestion for getting "labels" was to make the desired ouput of the neural network be a reconstruction of all or part of the input. For static data this amounted to learning a deep auto-encoder [14]. Unfortunately, in the last century, nobody could get deep autoencoders to work significantly better than Principal Components Analysis [7, 12]. For dynamic data, the most natural way to reconstruct the input data was to predict the next frame of data [9], but attempts to apply backpropagation-through-time to learning sequential data failed because the gradients grew or shrank multiplicatively at each time step [3]. We now have good ways of dealing with this problem [32, 23], but back in the 1980's the best we could do was to castrate backpropagation-through-time by throwing away the most interesting part of the gradient.

Given a large enough supply of class labels, back-propagation did learn to solve a number of difficult problems, especially when weight-sharing over time or space was used to implement prior knowledge about invariances [53, 28]. Without weight-sharing, however, it was hard to get backpropagation to make good use of multiple hidden layers and it failed to live up to the extremely high expectations we had for it in 1986. In particular, the hope that backpropagation-through-time could learn to solve complex problems by creating a myriad of small sequential "programs" and dynamically routing their outputs to the right places was never realised.

In 1995, Radford Neal [35] showed that for modest-sized training sets, feed-forward neural nets with one hidden layer generalised much better if the gradient produced by backpropagation was used to wander through the space of possible weights like a heavy particle on a bumpy error surface. The particle tends to head in a downhill direction gathering momentum, but this momentum is occasionally discarded and replaced by a random kick. Every so often, the set of weights corresponding to the current position of the particle is saved and predictions on test data are made by averaging the outputs produced by all of the different networks that use all of these different saved, weight vectors. Neal also showed that as the number of hidden units goes to infinity and the amount of weight-decay[4] on their outgoing connections also increases appropriately, his stochastic method of sampling from the space of good models becomes equivalent to a method known as "Gaussian Processes". The predictions of a Gaussian Process model can be computed in a more direct way [41], so from an engineering perspective, there is not much point using backpropagation with one hidden layer for modest-sized problems [30]. In the machine learning community, backpropagation went out of fashion. Retrospectively, it is fairly clear that this happened because the amount of labeled data and the computational resources available at the time were insufficient to make good use of the enormous modeling potential of multiple layers of non-linear features.

The other exciting new learning algorithm in the mid 1980s [21] was quite different in nature. It didn't work in practice, but theoretically it was much more interesting. From the outset, it was designed to learn binary distributed representations that captured the statistical structure implicit in a set of binary vectors, so it did not need labeled data. A more insightful way to say this is that it treated each training case as a vector of desired *outputs* of a stochastic generative model, so the training data consisted entirely of high-dimensional labels and what was missing was the inputs. The network, called a Boltzmann machine, contained a set of binary stochastic visible units which could be clamped to a training vector and a set of binary stochastic hidden units which learned to represent higher-order features of the data, typically ones that occurred more often than would be expected by chance. Any unit could be connected to any other unit and all of the connections were symmetric. In the vision and statistics literatures this is now known as a partially observed, inhomogeneous, Markov Random Field or an undirected graphical model. Boltzmann machines can also be used to learn the distribution

[4]Weight-decay keeps the weights small by adding an extra penalty that is proportional to the squared value of the weight. The gradient of the penalty pulls the weight towards zero.

of the outputs *given* an input vector. This conditional form of the Boltzmann machine allows it to perform the same tasks as a feedforward neural network trained with backpropagation, but with the added advantage that it can model correlations between the outputs. Given a particular input vector, for example, a conditional Boltzmann machine can assign high probabilities to the output vectors (1,1) and (0,0) and low probabilities to (1,0) and (0,1). A feedforward neural network cannot do this. In the machine learning literature this is known as a conditional random field, though most CRFs used in machine learning do not have hidden units so they cannot learn their own features.

After the weights on the connections have been learned, a Boltzmann machine can be made to perform perceptual inference by clamping a datavector on the visible units and then repeatedly updating the hidden units, one at a time, by turning on each binary hidden unit with a probability that is a logistic function of the total input it receives from all the other visible and hidden units (plus its own bias). After a sufficient length of time, the hidden vectors will be samples from the "stationary distribution" so any particular hidden vector will occur with a fixed probability that depends on how compatible it is with the datavector but does not depend on the initial pattern of hidden activities. Hidden vectors that occur with high probability in the stationary distribution are good representations of that datavector, at least according to the current model.

Another computation that a trained Boltzmann machine can perform is to generate visible vectors with a probability that equals the probability that the model assigns to those vectors. This is done by using exactly the same process as is used for perceptual inference, but with the visible units also being updated. It may, however, take a very long time before the network reaches its stationary distribution because this distribution usually needs to be highly multimodal to represent interesting data distributions well. Many interesting distributions have the property that there are exponentially many modes, each of which has about the same probability, separated by regions of much lower probability. The modes correspond to things that might plausibly occur and the regions between nodes correspond to extremely unlikely things.

The third and most interesting computation that a Boltzmann machine can perform is to update the weights on the connections in such a way that it is probably slightly more likely to generate all of the datavectors in the training set. Although this is a slow process, it is mathematically very simple and only uses information that is locally available. First, the inference process is run on a representative mini-batch of the training data and, for each pair of connected units, the expected

product of their binary activities is sampled. Then, the same computation is performed when the Boltzmann machine is generating visible vectors from its stationary distribution. The weight update is then proportional to the difference of the expected products during inference and generation. This difference is an unbiased estimate of the gradient of the sum of the log probabilities of generating the training data. It is surprising that the learning rule is this simple because the local gradient depends on all the other weights in the network. The most attractive aspect of Boltzmann machines is that everything a connection needs to know about the weights on other connections is contained in the difference of its expected activity products during inference and generation. Instead of requiring a backward pass which explicitly propagates information about gradients, the Boltzmann machine only requires the same computation to be performed twice, once with the visible units clamped to data and once without clamping. It does not require the neurons to communicate two quite different types of information.

Generating data from its model in order to collect the statistics required for learning would interrupt the processing of incoming information, so it is tempting to consider the possibility that this occurs at night during REM sleep[5]. At first sight this seems computationally awkward since it would only allow one weight update per day, but there is a more plausible version of this idea. During the night, generation from the model is used to estimate a baseline for the expected product of two activities. Then during the day, weights are raised when the product exceeds this baseline and lowered when it falls below the baseline. This allows many weight updates per day. though as the day progresses the learning would get less and less accurate.

From a Cognitive Science perspective, Boltzmann machines, if they could be made to work, would be interesting because they would exhibit multi-stability (as in the Necker cube illusion) and top-down effects during perceptual inference [33]. They would also have a tendency towards hallucinations if the input was disrupted, as in Claude Bonnet syndrome[42]. Unfortunately, with a lot of hidden units and unconstrained connectivity, Boltzmann machines trained with the algorithm decribed above learn extremely slowly. They need a very small learning rate to average away all of the noise caused by the stochastic sampling of the pairwise statistics, and they need to be run for an extremely long time in the generative phase to get unbiased samples. In the 1980's, therefore, they could only be used for toy tasks. Terry Sejnowski (personal communication, 1985) believed that the best hope for learning large Boltzmann machines was to find some way of learning smaller modules independently, but we had no idea how to do this. The

solution to this problem eventually presented itself after two decades of meandering through the space of unsupervised learning algorithms that learn distributed representations in networks of neuron-like processing units.

3 Directed versus undirected graphical models

"Graphical Models" is the name of a branch of Statistics and Artificial Intelligence that deals with probabilistic models whose parameters typically have a local structure that can be depicted by using a graph that is not fully connected. Missing interactions are depicted by missing edges in the graph which is a very efficient representation when nearly all of the possible interactions are missing. Graphical models come in two main flavors, directed and undirected.

In an undirected graphical model, like a Boltzmann machine, the parameters (*i. e.* the weights and biases) determine the "energy" of a joint configuration (a set of binary values for all of the observed and unobserved variables). Boltzmann machines use the Hopfield energy which is defined as the negative of the "harmony". The harmony is the sum over all active units of their biases, plus the sum over all pairs of active units of the weight between them. The probability of a joint configuration is then determined by its energy relative to other joint configurations using the Boltzmann distribution:

$$p(\mathbf{v}, \mathbf{h}) = \frac{e^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{v}',\mathbf{h}'} e^{-E(\mathbf{v}',\mathbf{h}')}} \tag{1}$$

The inference phase of the Boltzmann machine learning rule computes the data-dependent statistics needed to lower the energies of joint configurations that contain datavectors on the visible units. The generative phase computes the data-independent statistics needed to raise the energies of all joint configurations in proportion to how often they occur according to the current model. This makes the data more probable by decreasing the divisor in equation 1.

Directed graphical models work in a quite different way. In a directed graphical model, the variables have an ancestral partial ordering. When the model is generating data, the probabilty distribution for each variable only depends on its "parents" – directly connected variables that come earlier in the ordering. So to generate an unbiased sample from the model, we start by sampling the values of the highest ancestors from their prior distributions and then sample each lower

variable in turn using a probability distribution that depends on the sampled values of its parents. This dependency can be in the form of a conditional probability table whose size is exponential in the number of parents, or it can be a parameterized function that outputs a probability distribution for a descendant when given the vector of states of its parents. In a Gaussian mixture model, for example, the discrete choice of Gaussian is the highest level variable and this choice specifies the mean and covariance of the Gaussian distribution from which the lower-level, multi-dimensional variable is to be sampled when generating from the model.

The simplest examples of directed graphical models with hidden variables are Gaussian mixture models which have a single discrete hidden variable (the choice of which Gaussian to use) and factor analysis which has a vector of real-valued hidden variables (the factor values) that are linearly related to the observed data. When generalized to dynamic data these become Hidden Markov Models and Linear Dynamical Systems. All four of these models have a long history in statistics because they allow tractable inference: Given an observed datavector, there is an efficient way to compute the exact posterior distribution over all possible hidden vectors. Efficient inference makes it easy to use models after they have been learned, and it also makes it easy to learn them using variations of the EM algorithm [8].

In the 1980's, researchers in Artificial Intelligence who wanted to handle uncertainty in a principled way developed inference procedures for more complicated directed graphical models which they called "Bayes Nets" or "Belief Nets". Initially, they were not particularly interested in learning because they intended to use domain experts to specify the way in which the probability distribution of each discrete variable depended on the values of its parents. Judea Pearl [39] showed how correct inference could be performed by sending simple messages along the edges of a directed graph, provided there was only one path between any two nodes. His "belief propagation" algorithm can be viewed as a generalisation of the well-known "forward-backward" inference algorithm for Hidden Markov Models [2]. Heckerman [13] showed that expert systems worked better if they used a proper inference procedure instead of ad hoc heuristics and this had a big effect on the more open-minded members of the Artificial Intelligence community. At around the same time, the statistics community developed the "junction tree" algorithm for performing correct inference in sparsely connected, directed graphical models that contained multiple paths between nodes but no directed cycles [26].

The work on directed graphical models initially had little impact on those in the connectionist community who wanted to understand how the brain could

learn non-linear distributed representations. The graphical models community was mainly interested in relatively small models in which the structure of the graph and the way in which each variable depended on its parents were specified by a domain expert. As a result, the individual nodes in the graph could all be interpreted and the directed edges represented meaningful causal effects in the generative model. By contrast, the connectionist community was more interested in getting a large number of units with fairly high connectivity to learn to model the structure implicit in a large set of training examples and they were willing to entertain the possibility that there were many different and equally good solutions and that many of the units would have no simple interpretation.

The two communities became closer when Radford Neal [37] realised that the stochastic binary units used in a Boltzmann machine could be used instead to make a directed graphical model, called a "sigmoid belief net", in which the logistic sigmoid function $\sigma(x) = 1/(1 + exp(-x))$ is used to parameterize the way in which the probability distribution of a unit depends on the values of its parents. This differs from a Boltzmann machine because, when the model is generating data, the children have no effect on the parents so it is possible to generate unbiased samples in a single top-down pass.

Neal implemented a sigmoid belief net with multiple hidden layers and he compared its learning abilities with those of a Boltzmann machine. The inference procedure for a sigmoid belief net uses a similar iterative Monte Carlo process to the Boltzmann machine, but it is significantly more complicated because each hidden unit needs to see two different types of information. The first is the current binary states of all its parents and children and the second is the predicted probability of being on for each child given the current states of all that child's parents. The hidden unit then tends to pick whichever of its two states is the best compromise between fitting in with what its parents predict for it and ensuring that the predicted state of each of its children fits the current sampled state of that child.

Once a binary representation of a datavector has been sampled from the posterior distribution, the learning procedure for a sigmoid belief net is simpler than for a Boltzmann machine because a sigmoid belief net does not have to deal with the normalizing term in equation 1. The learning procedure is simply the delta rule: The sampled binary value of the "post-synaptic" child is compared with that child's probability of being on given the sampled states of its "pre-synaptic" parents. The top-down weights are then updated in proportion to the value of the parent times the difference between the sampled value of the child and the proba-

bility predicted by the parents. This is a generative version of the "delta" rule.

Neal showed that a sigmoid belief net learns faster than a Boltzmann machine, though not by a big factor. Given the extra complexity of the inference procedure, this did not seem like a good reason to abandon Boltzmann machines as a neural model, but it did raise the question of whether the inference procedure for a sigmoid belief net could be simplified.

4 Learning with incorrect inference

Here is an idea that sounds crazy: When given an input vector, instead of sampling the binary states of the hidden units from the true posterior distribution, which contains complicated correlations, sample them from a much simpler distribution that does not contain these complicated correlations and is therefore easy to compute. Then use these sampled states for learning as if they were samples from the correct distribution. On the face of it, this is a hopelessly heuristic approach that has no guarantee that the learning will improve the model. When the hidden states are sampled from the true posterior distribution, we are guaranteed that the learning will increase the probability that the model would generate the training data, provided we make a sufficiently large number of sufficiently small updates to the weights. But if we use incorrect samples of hidden state vectors, it is obvious that this guarantee no longer holds. Indeed, we could make the weight changes go in precisely the wrong direction by choosing the incorrect samples maliciously.

Using arguments from coding theory and from statistical physics, Radford Neal, Richard Zemel and I [36, 22] were able to show that learning using incorrect hidden samples is much more sensible than it appears. It does not necessarily increase the model's log probability of generating the training data[5], but it is guaranteed to improve a different quantity that is a lower bound on this log probability. For each individual training case, c, this bound is the log probability of generating that training case minus the divergence, $KL(Q_c||P_c)$, between the simplified distribution Q_c from which the hidden state vectors are actually sampled, and the true posterior distribution P_c from which they ought to have been sampled. When the weights are adjusted to maximize this bound, one of two things must happen: Either the log probability of the training data improves or the true posterior dis-

[5]Maximizing the product of the probabilities of generating all of the training cases is equivalent to maximizing the sum of the log probabilities.

tribution, P_c, becomes more similar to the simplified distribution Q_c that is being used to approximate it. So even though the log probability can fall, it can only do this by making the posterior distribution much easier to approximate and this itself is a good thing because it means we have a model in which a computationally simple way of doing approximate inference works pretty well.

Neal and I wrote a paper about this type of "variational" learning in 1993 and circulated it in the machine learning community but, initially, it had little impact and the Statistics journal we sent it to rejected it. Our 1993 paper eventually appeared as a chapter in an edited book on Graphical Models [36], and by the late 1990's the idea of variational learning had become very popular. It is now very widely used for learning complicated graphical models in which the true posterior is too difficult to compute exactly [25].

[Figure 2 about here.]

The natural way to apply variational learning to a sigmoid belief net leads to fairly complicated inference and learning procedures [47] because an inner loop of iterative optimization is required to find the best approximating distribution within the class of simplified distributions that are easy to compute. However, Peter Dayan noticed that if we are willing to make an additional approximation, both inference and learning become surprisingly simple [17]. Given a datavector, the best factorial distribution over the hidden units, Q, is the one that minimizes $KL(Q||P)$, where P is the true posterior distribution. If, instead, we train a separate feedforward neural net to minimize the highly correlated quantity $KL(P||Q)$, we get a very simple learning procedure called the "wake-sleep" algorithm. Like the Boltzmann machine, this algorithm has two phases, one in which it is driven by data and one in which it generates from its model, but here the resemblance ends. During the wake phase, feedforward "recognition" connections are used to infer an incorrect probability distribution for each hidden unit given the binary states of the units in the layer below (see figure 2). All of the units within a layer are then given binary states that are sampled independently from their inferred distributions. This is done one layer at a time, so only binary states need to be communicated. Given the sampled states of all the units, the top-down "generative" connections that form the sigmoid belief net can then be learned using the delta rule as described earlier. During the "sleep" phase, the network simply generates samples from its model. Since it generated these samples, it knows the correct states of the hidden units and it can use these states as targets for train-

ing the bottom-up recognition connections, again using the delta rule but with the roles of the pre-synaptic and post-synaptic units reversed.

The idea that the cortex learns by minimizing variational free energy has recently been espoused by Karl Friston and his collaborators [11] and is currently one of the many possibilities. As a contribution to machine learning, the wake-sleep algorithm is an interesting form of unsupervised learning but it is rather slow for deep networks that have many hidden layers and it is not used for practical applications. Aesthetically, the use of the wrong KL divergence for learning to approximate variational inference is unsatisfying. It would be much nicer if the recognition connections could perform correct inference for all of the hidden layers in a single bottom-up pass but this seemed hopelessly optimistic.

5 Restricted Boltzmann Machines

One model that does allow simple, correct inference of distributed non-linear representations is a "Restricted Boltzmann Machine" (RBM) in which there are no connections between hidden units and no connections between visible units. When this special case was suggested by Paul Smolensky [48], Terry Sejnowski and I thought it was of no particular interest because we had found the learning algorithm for the general case, and removing the connections between hidden units clearly made the model much less powerful. However, an RBM turned out to be exactly what was needed to divide the task of learning a deep network into a sequence of much simpler tasks.

In an RBM, the hidden units are conditionally independent given a visible vector, so unbiased samples of the expected activity products of a visible and a hidden unit during inference, $\langle v_i h_j \rangle_{\text{data}}$, can be obtained in one parallel step. To sample the expected products during generation, $\langle v_i h_j \rangle_{\text{model}}$, still requires multiple iterations that alternate between updating all the hidden units in parallel and updating all of the visible units in parallel. However, learning still works well if $\langle v_i h_j \rangle_{\text{model}}$ is replaced by $\langle v_i h_j \rangle_{\text{reconstruction}}$ which is obtained as follows: Starting with a data vector, $\mathbf{v}$, on the visible units, update all of the hidden units in parallel:

$$p(h_j = 1|\mathbf{v}) = \sigma(b_j + \sum_{i \in \text{vis}} v_i w_{ij}) \tag{2}$$

where b_j is a bias, w_{ij} is the weight between units i and j and σ is the logistic sigmoid funtion. Then update all of the visible units in parallel to get a "recon-

struction":

$$p(v_i = 1|\mathbf{v}) = \sigma(b_i + \sum_{j \in \text{hid}} h_j w_{ij}) \tag{3}$$

Then update all of the hidden units again. After averaging the pairwise statistics over one or more training cases, update the weights in parallel:

$$\Delta w_{ij} \propto \langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{reconstruction}} \tag{4}$$

This efficient learning procedure approximates gradient descent in a quantity called "contrastive divergence" and usually works well in practice [15].

6 Stacking RBMs to make a deep belief net

Once an RBM has been trained, its weights and biases define a joint distribution $p(\mathbf{v}, \mathbf{h})$ over visible and hidden binary state vectors. They also define $p(\mathbf{v})$, $p(\mathbf{h})$, $p(\mathbf{v}|\mathbf{h})$ and $p(\mathbf{h}|\mathbf{v})$. One slightly odd way to express $p(\mathbf{v})$ is in terms of the prior $p(\mathbf{h})$ that the RBM defines over its hidden states:

$$p(\mathbf{v}) = \sum_{\mathbf{h}} p(\mathbf{h}) p(\mathbf{v}|\mathbf{h}) \tag{5}$$

Now suppose we keep the $p(\mathbf{v}|\mathbf{h})$ defined by the first RBM, but we replace $p(\mathbf{h})$ in equation 5 by the probability distribution that a second RBM defines over its visible units as shown in figure 3. It can be shown that this will improve our model of the original training data if and only if the second RBM models the first RBMs aggregated posterior distribution over $\mathbf{h}$ better than the first RBMs prior, $p(\mathbf{h})$, models this aggregated posterior[6].

[Figure 3 about here.]

It is easy to ensure that the second RBM starts off with a model of the aggregated posterior that is just as good as the $p(\mathbf{h})$ defined by the first RBM: Simply initialise the second RBM to be the same as the first one, but turned upside down so that its visible units are the same as the first RBMs hidden units and vice versa.

[6]The aggregated posterior is the equally weighted mixture of all of the posterior distributions for the individual training cases. Even though each individual posterior is factorial, the aggregated posterior is not.

After training the second RBM, we can apply the same trick again to improve its model of the aggregated posterior of the first RBM. After training a stack of RBMs in this way, we end up with a peculiar kind of composite model called a deep belief net (DBN). The top two layers are the final RBM which acts as an undirected, high-level, associative memory. The remaining layers form a directed belief net because the only thing we kept from the earlier RBMs was the top-down weights that determine $p(\mathbf{v}|\mathbf{h})$. If we perform bottom-up inference in this DBN by using the weights of the RBMs in the bottom-up direction, we do not get samples from the true posterior distribution. Nevertheless, it can be shown [19] that each time we add another RBM to the stack we get a new DBN that has a better variational lower bound on the log probability of the training data than the previous DBN, provided we add the new RBM in the right way.

7 Fine-tuning a deep belief net

When a DBN has been created by stacking some RBMs, the whole system can be fine-tuned so that the weights in earlier layers have a chance to adapt to the weights that were subsequently learned in later layers. Either a generative or a discriminative objective function can be used for fine-tuning a DBN. Generative fine-tuning maximizes the probability that the DBN assigns to the training data and can be done using a contrastive version of the wake-sleep algorithm. Each connection that is not part of the top-level RBM is split into a bottom-up recognition connection and a top-down generative connection and the weights on these two connections are untied so that their values can become different.

In the "wake" phase of the learning, the units in all the hidden layers are driven bottom-up by the recognition connections. After a bottom-up pass that selects binary states for all the hidden units, the generative connections are trained to be better at reconstructing the binary activities in one layer from the binary activities in the layer above. This is done using the delta rule, as described in section 3. The bottom-up pass is then followed by a top-down pass that uses the generative connections, but instead of sampling from the top-level hidden states from the model, it just uses the top-level hidden states produced by the bottom-up pass. This is the contrastive version of the "sleep" phase. After the top-down pass, the recognition connections are trained to be better at recovering the true causes in the layer above, again using the delta rule but with the pre-synaptic and post-synaptic roles reversed.

During contrastive wake-sleep learning, the connections in the top-level RBM are kept symmetric and are trained using the usual contrastive divergence learning rule. After fine-tuning, samples generated from the model look more like the real data. A demonstration of a model with three hidden layers that has learned to generate images of handwritten digits can be found at `http://www.cs.toronto.edu/~hinton/digits.html`.

A very different way to fine-tune a DBN is to add a final layer of labels and to use a discriminative objective function that maximizes the log probability that the model assigns to the correct class label. The unsupervised training of the stack of RBMs is regarded as a "pre-training" phase whose role is to discover good features that model the structure in the input domain. Many of these features will be irrelevant to any particular discriminative task, but the ones that are relevant are likely to be much more useful than the raw inputs because they represent strong higher-order correlations in the data that are probably related to the real causes of the data. These relevant features can be given strong weights to the label units and they can also be slightly adjusted to make them more useful for discrimination. This is done by simply treating the DBN, with its extra final layer of labels, as a feedforward neural network and using standard backpropagation. This makes backpropagation work a whole lot better in deep feedforward networks that have many hidden layers [20]. For example, DBNs fine-tuned with backpropagation are now the best speaker-independent method for recognizing phonemes on the benchmark TIMIT test set [6].

In an extensive set of simulations, [10] show that there are two reasons why pretraining a stack of RBMs makes backpropagation work so much better. The first is that when the hidden units are initialised to sensible features by the pre-training, backpropagation can find better local minima on the training data. The optimization is much easier because the weights are started in a good region of the space so backpropagation does not need to design good features from scratch. It merely needs to slightly adjust the features so that the decision boundaries are in exactly the right place.

The second reason for the improvement is that the minima found after unsupervised pre-training give significantly better generalization to the test data. They suffer much less from overfitting, presumably because most of the information in the learned weights comes from modeling the input patterns rather than modeling the function that maps from input to label. The input patterns contain much more information than the labels, so modeling the input can support many more well-determined parameters than modeling the labels given the input. This is especially

important for learning tasks in which there is a large amount of unlabeled data for pre-training and a relatively small amount of labelled data for fine-tuning[7].

It is a curious twist of fate that the search for an efficient, modular way of training large Boltzmann machines ended up with a method for making back-propagation work much better in deep feed-forward neural networks. Some time later [46], Ruslan Salakhutdinov and I discovered a much less obvious way of stacking RBMs to produce a composite model which really is a deep Boltzmann machine (DBM).

8 Stacking RBM's to make a deep Boltzmann machine

Instead of entirely replacing the prior distribution that an RBM defines over its hidden units by a distribution defined by the next RBM in the stack, we could take the geometric mean of these two distributions by using half of the bottom-up weights and half of the top-down weights. For layers in the middle of a deep stack of RBMs this is easy to do: We simply learn an RBM and then divide all its weights and biases by 2 when we compose the individual RBMs in the stack to make a deep Boltzmann machine. For the first RBM in the stack, we need to halve its bottom-up weights, but not its top-down weights and we need to end up with symmetric weights. So we train this RBM with a constraint that bottom-up the weights are twice the top-down weights. This is no longer a proper RBM, but contrastive divergence training still works well. Conversely, for the last RBM in the stack, we can constrain the top-down weights to be twice the bottom-up weights during the pre-training or we can use two sets of hidden units with tied weight matrices and discard one of these sets when we add it to the final deep Boltzmann machine.

9 Fine-tuning a deep Boltzmann machine

After a deep Boltzmann machine has been composed out of RBMs, it is possible to train all of the weights together to improve the generative model. The correct maximum likelihood way to update the weight on the symmetric connection be-

[7] This is clearly the situation for a child learning to name familiar objects.

tween two units is to use the difference between the expected activity products during inference and generation. However, with many hidden layers, it is very difficult to sample the activity products from the true posterior distribution, so instead of performing correct inference, we resort to a variational approximation which uses a much simpler distribution in which the hidden activities are assumed to be independent given the datavector. The fine-tuning is, therefore, only optimizing a variational lower bound on the log probability of generating the training data.

For estimating the expectations of the activity products when generating from the model, it is not permissable to use a variational approximation because these products contribute a negative term to the gradient. As a consequence, if a variational approximation is used for the generative expectations, instead of adjusting the weights to make the variational bound tighter, the learning tries to make the bound as loose as possible, which is very bad news. Ruslan Salakhutdinov realised that this problem can be solved by estimating the generative expectations using a set of persistent Markov chains whose states are updated after each weight update[37]. This means that we need to remember the binary states of all of the units for each persistent chain[8].

If the generative model has many different modes that are widely separated, which is what is required for many applications, one would expect that a very large number of persistent chains would be needed to correctly average the activity products over all the different modes. In practice however, a small number of persistent chains works very well. This is because the activity products contributed by these chains are used for unlearning the model's own beliefs. So the energy landscape is modified to raise the energy of whatever state a persisitent chain is currently in [51, 52]. This causes the chain to rapidly move to another part of the energy landscape. If any chain is stuck in a deep energy minimum that does not contain any training data, the learning will quickly rasie the energy of this minimum until the chain escapes. The learning, therefore, causes the states of the chains to move around much faster than they would with the learning turned off – a very fortuitous phenomenon that makes it possible to fine-tune deep Boltzmann machines with many hidden layers and millions of weights [46].

It is tempting to also use persistent chains for estimating the data-dependent statistics, and this works well for small datasets [37]. For large datasets, however, it is much more efficient to update the weights after a small minibatch of training

[8] Actually, it is sufficient to remember the states of alternate layers.

cases and this means that the weights have typically changed by a lot before we revisit the same training case. Consequently, the persistent chain for that training case is no longer anywhere near its stationary distribution given the current weights.

10 Summary of the main story

In about 1986, backpropagation replaced the Boltzmann machine learning algorithm as the method of choice for learning distributed representations. This paper has described three developments, enumerated below, in methods for learning stochastic generative models. These three methods resulted in a very good way to initialize the weights of deterministic feedforward neural networks. With this initialization, backpropagation works much better.

The weights of a deep Boltzmann machine can also be initialised in a similar way, and a fourth development then allows deep Boltzmann machines to be fine-tuned as a generative model. After 25 years, this finally makes it possible to learn large, deep Boltzmann machines.

1. **Variational learning**: With the advent of graphical models, it became obvious that the stochastic binary variables used in the Boltzmann machine could be used in directed generative models – sigmoid belief nets – and this revived interest in stochastic neural nets. For these directed nets, learning is easy if the hidden states can be sampled from their posterior distribution but sampling from this distribution is infeasible in large and densely connected networks. Surprisingly, learning still works pretty well if we sample the hidden states from a much simpler distribution, because the learning optimizes a variational bound on the log probability of generating the data. Optimizing this bound changes the weights to achieve a compromise between two goals: maximize the probability of generating the training data and make the true posterior be as similar as possible to the type of simple distribution that is being used to approximate it.

2. **Contrastive divergence**: There is a very simple form of the Boltzmann machine, first described by Paul Smolensky, in which inference is very easy because the hidden units really are independent given the data. Learning is still a problem because it appears to require samples from the model and these are hard to get for undirected models. Again, the solution is to use

the wrong statistics. In this case the activity products during generation are replaced by the activity products after reconstructing the data from the hidden activities. This finally made it possible to learn large Boltzmann machines, albeit ones with very restricted connectivity.

3. **Forming deep models by stacking RBMs**: After learning one RBM, the states of its hidden units can be used as data to train another RBM. A stack of RBMs learned in this way is a good way to initialize the weights of a feedforward neural net that is then fine-tuned with backpropagation. However, the composite generative model formed by a stack of RBMs is not a multilayer Boltzmann machine. It is a hybrid that has an undirected RBM in its top two layers and a directed belief net in its lower layers. To compose a multilayer Boltzmann machine out of RBMs, we need to average the top-down and bottom-up input that a hidden layer receives from the two RBM's in the stack that contain that layer. This is different from simply replacing the bottom-up input with the top-down input which is what happens in the generative model when RBMs are composed to form a deep belief net.

4. **Combining variational learning with persistent Markov chains**: Early attempts to find an effective learning procedure for Boltzmann machines just assumed that the same method would be used for estimating both the data-dependent and the data-independent statistics. Variational methods are no good for the data-independent statistics and persistent chains are no good for the data-dependent statistics when using small minibatches and large datasets because the saved state of a persistent chain for a given mini-batch is completely out of date by the time that mini-batch is revisited. However, combining variational learning for the data-dependent statistics with persisitent chains for the data-independent statistics works well due to an unexpected interaction: In addition to trying to make the variational approximation tight, the learning makes the persistent chains move around the space rapidly. For deep Boltzmann machines that have already been pre-trained, this combination is very effective.

In this paper I have described how two learning procedures from the 1980's evolved over the next 25 years. I focussed on the main ideas that were required to get these learning procedures to work really well. Other important developments could not be covered. These include related developments in the types of units that can be used [54, 34], the ways they can interact [16], the ways they can

share weights [29], and the modifications that allow these ideas to be applied to sequential data [49]. Another important thread has been the development of alternative unsupervised modules that can be used to replace RBMs in the pre-training phase. These include denoising and contractive autoencoder modules developed by Yoshua Bengio's group [1, 43] and sparse energy-based modules developed by Yann LeCun's group [40].

11 A speculation on the future of neural network models

I currently believe that the highly idealized "neurons" used in this paper may suffer from a serious flaw as models of real neurons. They assume that a real cortical neuron cannot communicate an approximate real value efficiently. Experiments by Markram and others[31] have shown that synaptic learning rules can be exquisitely sensitive to the precise time of a spike and this casts doubt on the idea that the precise time of a spike is unreliable and therefore conveys little information. When performing signal processing, communicating a 0 or a 1 is not nearly as useful as communicating either a 0 or the combination of a 1 with a real number that is accurate to within about 10%. It would be very surprising if hundreds of millions of years of evolution had failed to notice that the precise time of a spike can be used to convey this additional analog value. For tasks such as sound localization, spike times can be made accurate to within less than 1 ms, so the only thing that could prevent evolution from exploiting this free additional bandwidth would be if there was some other very important reason for making cortical neurons extremely noisy[4].

Of course, for precise spike times to be useful, neurons must be able to compute with them. I shall therefore sketch out very briefly how this could be done. The method I propose here is almost certainly wrong in its details, but once you have seen how convenient it is to use spike times for signal processing you are forced to choose between two possibilities both of which are problematic: Either the brain uses spike times to communicate analog values or there is some good reason why it does not need to communicate analog values.

The first operation I shall consider is comparing some real values to see if many of them are approximately equal. This is very hard to do using standard binary, sigmoid or linear threshold neurons. Even for only two values it is equivalent

to solving the famous XOR problem and requires an extra layer of processing. Using spike times it is trivial. We simply use feedforward excitation followed, a few milleseconds later, by feedforward inhibition via one or more inhibitory interneurons. To exceed its threshold, the receiving neuron must receive several excitatory spikes in the same narrow temporal window before the inhibition arrives. If it does, it has detected that some numbers agree and it reports the binary existence of this agreement by spiking and the agreed value by the time of the spike.

The second operation I shall consider is computing a scalar product between a vector of spike times and a vector of synaptic weights. For simplicity, let us make the gross assumption that an excitatory post-synaptic potential has a very fast rise time followed by a rate of injection of charge that is constant over the next 20ms or so. Let us also assume that there is a global oscillation and that a particular phase of this oscillation is called the "deadline". A spike arriving at a time t_i before the deadline will initiate the injection of charge at a rate of w_i. The temporal integral of the injected charge at the deadline will therefore be the scalar product $\sum_i t_i w_i$. The multiplies have been computed by temporal integration and the adds by the addition of charge. We then need to convert the amount of injected charge into the time advance of an outgoing spike. This can be done by injecting additional charge at a rate of $1 - \sum_i w_i$ starting at the deadline. The total rate of injection of charge will then be 1 and the time after the deadline at which the neuron crosses its threshold will be advanced by exactly the amount of charge that was already injected by the deadline. So the scalar product has been computed and converted back into the advance of a spike time in one cycle of the global oscillation.

There are numerous problems with this over-simplified model: EPSPs decay with time, the rate of injection of charge depends on the membrane potential, incoming spikes after the deadline need to be blocked, membranes leak, not all numbers are positive, and so on. Nevertheless, the combination of temporal integration for computing multiplies, charge accumulation for computing adds, and an additional clocked input for converting accumulated charge into the time advance of an outgoing spike seems like a very efficient way to use a membrane to compute a scalar product.

If precise spike times are being used by cortex, it is rather surprising that there is not more experimental evidence in their favor. One possible reason is that experimentalists have been trying to correlate precise spike times with the wrong kind of information. In inferotemporal cortex, for example, the *existence* of a spike could be used to represent the presence of an entity of a particular kind, and the precise times of spikes could represent the pose parameters (*i.e.* the position,

orientation and scale of the entity relative to the viewer). A scalar product can then be used to predict one of the pose parameters of a whole from all the pose parameters of a part[9] and if many parts agree a neuron could use this coincidence to decide that the whole is present and also to report the value of the pose parameter. The precise time of a spike would not convey any additional information about the presence or absence of a visual entity but it would convey information about the pose of the entity. This seems worth looking for in inferotemporal cortex[10].

The idea that neurons can communicate approximate real numbers also undermines one of the main motivations behind coarse coding [44]. In coarse coding, the six pose parameters of a 3-D object (three orientation and three position) are coded by using a large number of binary neurons that each have a large, receptive field in the six-dimensional pose space. The intersection of the receptive fields of the active neurons can then code the six pose parameters fairly accurately. Indeed, as the receptive fields get larger, the accuracy of the encoding gets better, so large receptive fields cannot be interpreted as evidence against accurate representations of pose. This is an ingenious way of using binary neurons, but six numbers is a lot more economical and is also a lot more useful for the computations required to recognize an object by recognizing that its parts all predict the same pose for the whole and therefore have the appropriate spatial relationships to each other, as described in the previous paragraph. Scalar products of vectors of pose parameters with vectors of weights that describe spatial relationships are the way that computer graphics deals with viewpoint so effortlessly and it makes a lot of sense for the cortex to use the same method. The idea that vision is inverse graphics[24] may be more than just a guiding principle: It may be true right down to the level of the matrix multiplies used to relate the poses of wholes to the poses of their parts.

References

[1] P. Vincent H..and Larochelle.and I. Lajoie and.Y. Bengio and P.-A. Manzagol. Stacked denoising autoencoders: learning useful representations in a

[9]This only works if the pose parameters of the part are represented in a way that allows spatial relationships to be modeled as matrix multiplies, as is done in computer graphics. A neural network that learns to extract the correct representation of pose parameters from images is described in [18].

[10]There is evidence that in hippocampal place cells, the phase of a spike relative to a global oscillation is used to represent the position of the rat within a place field[38] *i. e.* a pose parameter.

deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11:3371–3408, 2010.

[2] L. E. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, 3:1–8, 1972.

[3] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5:157–166, 1994.

[4] L. Bsing, J. Bill, B. Nessler, and W. Maass. Neural dynamics as sampling: A model for stochastic computation in recurrent networks of spiking neurons. *PLoS Computational Biology*, doi:10.1371/journal.pcbi.1002211., 2011.

[5] F. Crick and G. Mitchison. The function of dream sleep. *Nature*, 304:111114, 1986.

[6] G. E. Dahl, M. Ranzato, A. Mohamed, and G. E. Hinton. Phone recognition with the mean-covariance restricted boltzmann machine. In *Advances in Neural Information Processing Systems 24*, 2010.

[7] D. DeMers and G.W. Cottrell. Nonlinear dimensionality reduction. In Cowan Hanson and Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 580–587, San Mateo, CA, 1993. Morgan Kaufmann.

[8] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, B*, 39:1–38, 1977.

[9] J.L. Elman. Finding Structure in Time. *Cognitive Science*, 14(2):179–211, 1990.

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

[11] K. Friston, J. Kilner, and L. Harrison. A free energy principle for the brain. *Journal of Physiology*, 100:70–87, 2006.

[12] R. Hecht-Nielsen. Replicator neural networks for universal optimal source coding. *Science*, 269:1860–1863, 1995.

[13] D. Heckerman. Probabilistic interpretations for mycin's certainty factors. In L. Kanal. and J. Lemmer, editors, *Uncertainty in Artificial Intelligence*, pages 167–196. North-Holland, New York, 1986.

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

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

[16] G. E. Hinton. Learning to represent visual input. *Philosophical Transactions of the Royal Society, B.*, 365:177–184, 2010.

[17] 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.

[18] G. E. Hinton, A. Krizhevsky, and S. Wang. Transforming auto-encoders. In *ICANN-11: International Conference on Artificial Neural Networks*, Helsinki, 2011.

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

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

[21] 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.

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

[23] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

[24] B. K. P. Horn. Understanding image intensities. *Artificial Intelligence*, 8:201–231, 1977.

[25] 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.

[26] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *J. Roy. Statist. Soc. B*, 50:157224., 1988.

[27] Y. LeCun. Une procédure d'apprentissage pour réseau a seuil asymmetrique (a learning scheme for asymmetric threshold networks). In *Proceedings of Cognitiva 85*, pages 599–604, Paris, France, 1985.

[28] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[29] H. Lee, R. Grosse, R. Ranganath, and A. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th International Conference on Machine Learning*, pages 609–616, Montreal, 2009.

[30] D. J. C. MacKay. *Information theory, inference, and learning algorithms*. Cambridge University Press, 2003.

[31] H. Markram, L. Joachim, M. Frotscher, and B. Sakmann. Regulation of synaptic efficacy by coincidence of postsynaptic aps and epsps. *Science*, 275:213–215, 1997.

[32] J. Martens. Deep learning via hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, 2010.

[33] J. L. McClelland and D. E. Rumelhart. An interactive activation model of context effects in letter perception: Part 1. an account of basic findings. *Psychological Review*, 88:375–407, 1981.

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

[35] R. M. Neal. *Bayesian Learning for Neural Networks*. PhD thesis, Department of Computer Science, University of Toronto, 1994.

[36] 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.

[37] R.M. Neal. Connectionist learning of belief networks. *Artificial Intelligence*, 56(1):71–113, 1992.

[38] J. O'Keefe and M. L. Recce. Phase relationship between hippocampal place units and the eeg theta rhythm. *Hippocampus 3, 317 (1993)*, 3:317–330, 1993.

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

[40] M. Ranzato, Y. Boureau, and Y. LeCun. Sparse feature learning for deep belief networks. In *Advances in Neural Information Processing Systems 20*, pages 1185–1192, San Mateo, CA, 2007. Morgan Kaufmann.

[41] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[42] D. P. Reichert, P. Series, and A. J. Storkey. Hallucinations in charles bonnet syndrome induced by homeostasis: a deep boltzmann machine model. In *Advances in Neural Information Processing Systems 23*, pages 2020–2028, San Mateo, CA, 2010. Morgan Kaufmann.

[43] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. Contracting auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on Machine Learning*, 2011.

[44] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Distributed representations. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Volume 1: Foundations*, pages 77–109. MIT Press, Cambridge, 1986.

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

[46] R. R. Salakhutdinov and G. E. Hinton. An efficient learning procedure for deep boltzmann machines. *Neural Computation*, in press, 2012.

[47] L. K. Saul, T. Jaakkola, and M. Jordan. Mean field theory for sigmoid belief networks. *Journal of Artificial Intelligence Research*, 4:61–76., 1996.

[48] 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.

[49] G. W. Taylor, G. E. Hinton, and S. Roweis. Two distributed-state models for generating high-dimensional time series. *Journal of Machine Learning Research*, 12:1025–1068, 2011.

[50] J. B. Tenenbaum, T. L. Griffiths, and C. Kemp. Theory-based bayesian models of inductive learning and reasoning. *Trends in Cognitive Sciences*, 10:309–318, 2006.

[51] T. Tieleman. Training restricted Boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th International Conference on Machine learning*, pages 1064–1071. ACM New York, NY, USA, 2008.

[52] 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.

[53] A. Waibel, T. Hanazawa, G. E. Hinton, K. Shikano, and K. J. Lang. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 31:328–339, 1989.

[54] 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*, 2005.

[55] P.J. Werbos. *Beyond regression: new tools for prediction and analysis in the behavioral sciences*. PhD thesis, Harvard University, 1974.

[56] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87:1423 – 1447, 1999.

List of Figures

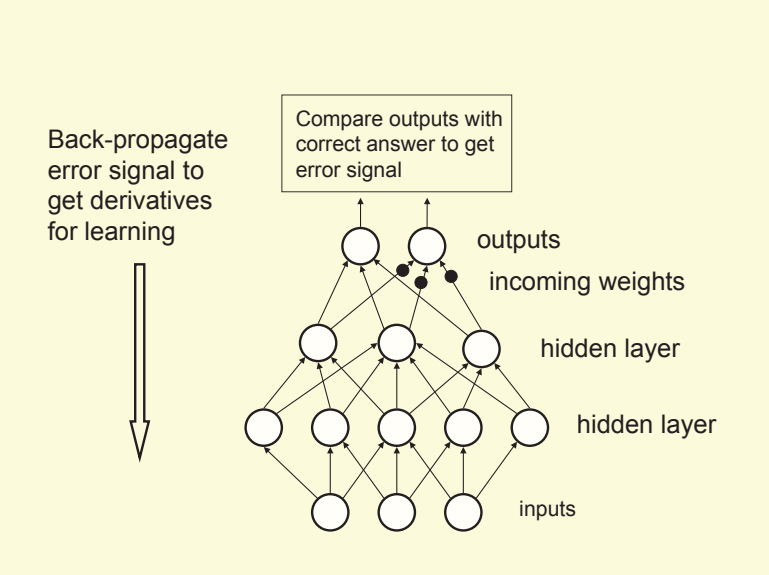

Figure 1: A feedforward neural network containing two hidden layers. The network maps input vectors to predicted output vectors in a forward pass. The incoming weights to each hidden or output unit are learned gradually by changing them in the direction that reduces the discrepancy between the predicted output and the correct output, averaged over a set of training cases. For each training case, the effect of changing a weight on the discrepancy is computed by using the chain rule to backpropagate error derivatives from one layer to the previous layer. The incoming weights of each hidden unit determine how it responds to patterns of activity in the layer below and different hidden units tend to discover different features that are useful for predicting the correct output.

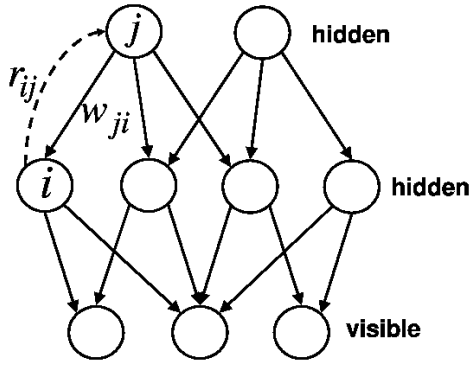

Figure 2: A multi-layer belief net composed of logistic binary units. To generate fantasies from the model during the sleep phase, we start by picking a random binary state of 1 or 0 for each top-level unit. Then we perform a stochastic downwards pass in which the probability, $\hat{h}_i$, of turning on each unit, i, is determined by applying the logistic function, $\sigma(x) = 1/(1 + \exp(-x))$, to the total input $\sum_j h_j w_{ji}$ that i receives from the units, j, in the layer above, where h_j is the binary state that has already been chosen for unit j. It is easy to give each unit an additional bias, but this has been omitted for simplicity. r_{ij} is a recognition weight that is used for inferring the activity in one layer from the activities in the layer below during the wake phase using exactly the same inference procedure as the sleep phase but in the reverse direction.

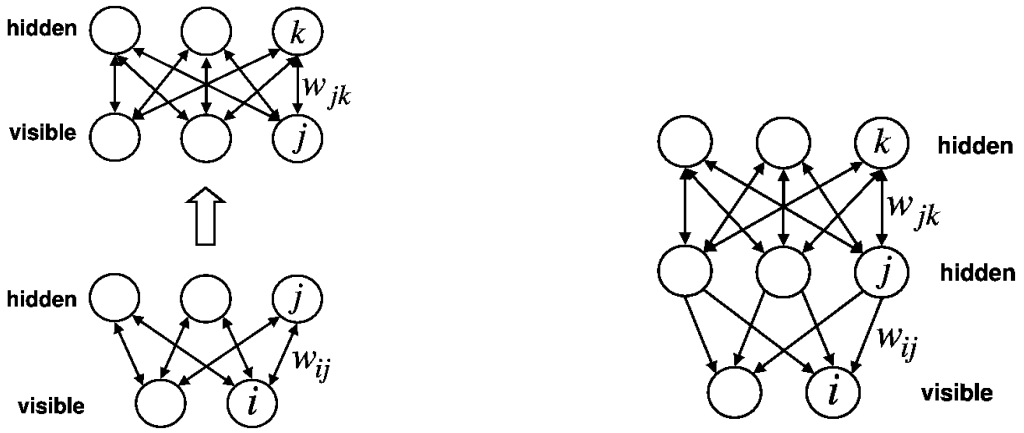

Figure 3: a) Two separate restricted Boltzmann machines (RBM's). The stochastic, binary variables in the hidden layer of each RBM are symmetrically connected to the stochastic, binary variables in the visible layer. There are no connections within a layer. The higher-level RBM is trained by using "data" that consists of the inferred hidden activities of the lower RBM, when it is presented with real data. b) The composite generative model produced by composing the two RBM's. Note that the connections in the lower layer of the composite generative model are directed. The hidden states are still inferred by using bottom-up recognition connections, but these are no longer part of the generative model.