The wake-sleep algorithm for unsupervised neural networks

Geoffrey E Hinton* Peter Dayan Brendan J Frey Radford M Neal

Department of Computer Science
University of Toronto
6 King's College Road
Toronto M5S 1A4, Canada

3rd April 1995

Abstract

An unsupervised learning algorithm for a multilayer network of stochastic neurons is described. Bottom-up "recognition" connections convert the input into representations in successive hidden layers and top-down "generative" connections reconstruct the representation in one layer from the representation in the layer above. In the "wake" phase, neurons are driven by recognition connections, and generative connections are adapted to increase the probability that they would reconstruct the correct activity vector in the layer below. In the "sleep" phase, neurons are driven by generative connections and recognition connections are adapted to increase the probability that they would produce the correct activity vector in the layer above.

Supervised learning algorithms for multilayer neural networks face two problems: They require a teacher to specify the desired output of the network and they require some method of communicating error information to all of the connections. The wake-sleep algorithm avoids both these problems. When there is no external teaching signal to be matched, some other goal is required to force the hidden units to extract underlying structure. In the wake-sleep algorithm the goal is to learn representations that are economical to describe but allow the input to be reconstructed accurately. We can quantify this goal by imagining a communication game in which each vector of raw sensory inputs is communicated to a receiver by first sending its hidden representation and then sending the difference between the input vector and its top-down reconstruction from the hidden representation. The aim of learning is to minimize the "description length" which is the total number of bits that would be required to communicate the input vectors in this way [1]. No communication actually takes place, but minimizing the description length that would be required forces the network to learn economical representations that capture the underlying regularities in the data [2].

The neural network has two quite different sets of connections. The bottom-up "recognition" connections are used to convert the input vector into a representation in one or more layers of hidden units. The top-down "generative" connections are then used to reconstruct an approximation to the input vector from its underlying representation. The training algorithm for these two sets of connections can be used with many different types of stochastic neuron, but for simplicity we use only stochastic binary units that have states of 1 or 0. The state of unit v is s_v and the probability that it is on is:

$$\text{Prob}(s_v = 1) \;=\; \frac{1}{1 + \exp(-b_v - \sum_u s_u w_{uv})} \tag{1}$$

where b_v is the bias of the unit and w_{uv} is the weight on a connection from unit u. Sometimes the units are driven by the generative weights and other times by the recognition weights, but the same equation is used in both cases (figure 1).

In the "wake" phase the units are driven bottom-up using the recognition weights, producing a representation of the input vector in the first hidden layer, a representation of this representation in the second hidden layer and so on. All of these layers of representation combined are called the "total representation" of the input, and the binary state of each hidden unit, j, in total representation α is s_j^α. This total representation could be used to communicate the input vector, d, to a receiver. According to Shannon's coding theorem, it requires $-\log r$ bits to communicate an event that has probability r under a distribution agreed by the sender and receiver. We assume that the receiver knows the top-down generative weights [3] so these can be used to create the agreed probability distributions required for communication. First, the activity of each unit, k, in the top hidden layer is communicated using the distribution $(p_k^\alpha, 1 - p_k^\alpha)$ which is obtained by applying Eq. 1 to the single generative bias weight of unit k. Then the activities of the units in each lower layer are communicated using the distribution $(p_j^\alpha, 1 - p_j^\alpha)$ obtained by applying Eq. 1 to the already communicated activities in the layer above, s_k^α, and the generative weights, w_{kj}. The description length of the binary state of unit j is:

$$C(s_j^\alpha) = -s_j^\alpha \log p_j^\alpha - (1 - s_j^\alpha)\log(1 - p_j^\alpha) \tag{2}$$

The description length for input vector d using the total representation α is simply the cost of describing all the hidden states in all the hidden layers plus the cost of describing the input vector given the hidden states

$$C(\alpha, d) = C(\alpha) + C(d|\alpha) = \sum_{\ell \in L} \sum_{j \in \ell} C(s_j^\alpha) + \sum_i C(s_i^d|\alpha) \tag{3}$$

where ℓ is an index over the L layers of hidden units and i is an index over the input units which have states s_i^d.

Because the hidden units are stochastic, an input vector will not always be represented in the same way. In the wake phase, the recognition weights determine a conditional probability distribution, $Q(\cdot|d)$ over total representations. Nevertheless, if the recognition weights are fixed, there is a very simple, on-line method of modifying the generative weights to minimize the expected cost $\sum_\alpha Q(\alpha|d)C(\alpha, d)$ of describing the input vector using a stochastically chosen total representation. After using the recognition weights to choose a total representation, each generative weight is adjusted in proportion to the derivative of equation 3 by using the purely local delta rule:

$$\Delta w_{kj} = \epsilon s_k^\alpha (s_j^\alpha - p_j^\alpha) \tag{4}$$

where ϵ is a learning rate. Although the units are driven by the recognition weights, it is only the generative weights that learn in the wake phase. The learning makes each layer of the total representation better at reconstructing the activities in the layer below.

It seems obvious that the recognition weights should be adjusted to maximize the probability of picking the α that minimizes $C(\alpha, d)$. But this is incorrect. When there are many alternative ways of describing an input vector it is possible to design a stochastic coding scheme that takes advantage of the entropy across alternative descriptions [1]. The cost is then:

$$C(d) = \sum_\alpha Q(\alpha|d)C(\alpha, d) - \left(- \sum_\alpha Q(\alpha|d) \log Q(\alpha|d) \right) \tag{5}$$

The second term is the entropy of the distribution that the recognition weights assign to the various alternative representations. If, for example, there are two alternative representations each of which costs 4 bits, the combined cost is only 3 bits provided we use the two alternatives with equal probability[4]. It is precisely analagous to the way in which the energies of the alternative states of a physical system are combined to yield the Helmholtz free energy of the system. As in physics, $C(d)$ is minimized when the probabilities of the alternatives are exponentially related to their costs by the Boltzmann distribution (at a temperature of 1):

$$P(\alpha|d) = \frac{\exp(-C(\alpha, d))}{\sum_\beta \exp(-C(\beta, d))} \tag{6}$$

So rather than adjusting the recognition weights to focus all of the probability on the lowest cost representation, we should try to make the recognition distribution $Q(\cdot|d)$ be as similar as possible to the Boltzmann distribution $P(\cdot|d)$ which is the posterior distribution over representations given the data and given the network's generative model. It is exponentially expensive to compute $P(\cdot|d)$ exactly [5], but there is a simple way of getting approximately correct target states for the hidden units in order to train the distribution $Q(\cdot|d)$ produced by the bottom-up recognition weights.

We turn off the recognition weights and drive all of the units in the network using the generative weights, starting at the topmost hidden layer and working down all the way to the input units. Because the units are stochastic, repeating this process typically produces many different "fantasy" vectors on the input units. These fantasies provide an unbiased sample of the network's generative model of the world. Having produced a fantasy, we then adjust the recognition weights to maximize the log probability of recovering the hidden activities that actually caused the fantasy

$$\Delta w_{jk} = \epsilon s_j^{\gamma}(s_k^{\gamma} - q_k^{\gamma}) \tag{7}$$

where γ specifies the states of both the hidden units and the input units for a particular fantasy and q_k^{γ} is the probability that unit k would be turned on by the recognition weights operating on the binary activities, s_j^{γ}, in the layer below [6]. We call this the "sleep" phase of the algorithm. Like the wake phase, it uses only locally available information. A potential drawback of the sleep phase is that we would like the recognition weights to be good at recovering the true causes for the training data but the sleep phase optimizes the recognition weights for fantasy data. Early in the learning, fantasies will have a quite different distribution than the training data.

The distribution $Q(\cdot|d)$ produced by the recognition weights is a factorial distribution in each hidden layer because the recognition weights produce stochastic states of units within a hidden layer that are conditionally independent given the states in the layer below. It is natural to use factorial distributions in a neural net because it allows the probability distribution over the 2^n alternative hidden representations to be specified with n numbers instead of $2^n - 1$. This simplification, however, will typically make it impossible for the distribution $Q(\cdot|d)$ to exactly match the posterior distribution $P(\cdot|d)$ in equation 6. It makes it impossible, for example, to capture "explaining away" effects where the activity vector in one layer can be economically explained by activating either unit a or unit b in the layer above but not by activating both of them.

The restriction of $Q(\cdot|d)$ to a factorial distribution is a potentially very serious limitation. The reason it is not a fatal flaw is that the wake phase of the algorithm adapts the generative weights so as to make $P(\cdot|d)$ close to $Q(\cdot|d)$, thus limiting the loss caused by the inability of $Q(\cdot|d)$ to model non-factorial distributions. To see why this effect occurs it is helpful to rewrite equation 5 in a different form

$$C(d) = \sum_{\alpha} P(\alpha|d)C(\alpha, d) - \left(- \sum_{\alpha} P(\alpha|d) \log P(\alpha|d) \right) + \sum_{\alpha} Q(\alpha|d) \log \frac{Q(\alpha|d)}{P(\alpha|d)} \quad (8)$$

The first two terms in equation 8 are exactly $-\log P(d)$ under the current generative model. The last term, which cannot be negative, is the Kullback-Leibler divergence between $Q(\cdot|d)$ and $P(\cdot|d)$ which is the amount by which the description length using $Q(\cdot|d)$ exceeds $-\log P(d)$. So for two generative models which assign equal probability to d, minimizing equation 8 with respect to the generative weights will tend to favor the model whose posterior distribution is most similar to $Q(\cdot|d)$. Within the available space of generative models, the wake phase seeks out those that give rise to posterior distributions which are approximately factorial.

Because we are making several approximations, the algorithm must be evaluated by its performance. Figure 2 shows that it can learn the correct multilayer generative model for a simple toy problem. Moreover, after learning, the Kullback-Leibler divergence in equation 8 is only 0.08 bits which indicates that this term has forced a solution in which the generative model has an almost perfectly factorial posterior. We also tested the algorithm on two quantitative aspects of its capacity to build models of images of highly variable handwritten digits (figure 3a). Learning ten different models, one for each digit, we were able to recognize new digits accurately by seeing which models gave the most economical descriptions of them. Figure 3b shows that after it has learned a digit model, the fantasies generated by the network are very similar to the real data. We also trained a single large network on all the digits and confirmed that it compressed new digits almost as well as did these ten digit-specific networks, and nearly twice as well as a naive code [17].

Two of the most widely used unsupervised training algorithms for neural networks are principal components analysis and competitive learning (sometimes called vector quantization or clustering). They can both be viewed as special cases of the minimum description length approach in which there is only one hidden layer and it is unnecessary to distinguish between the recognition and generative weights because they are always the same [7]. Other learning schemes have been proposed that use separate feedforward and feedback weights [8, 9, 10, 11]. By contrast with Adaptive Resonance Theory [8], the Counter-Streams model [9], and Kawato *et al*'s algorithm [10], the wake-sleep algorithm treats the problem of unsupervised learning as statistical – one of fitting a generative model which accurately captures the structure in the input examples. Kawato's model is couched in terms of forward and inverse models [12], which are an alternative way to look at our generative and recognition models. The wake-sleep algorithm is closest in spirit to Barlow's ideas about invertible factorial representations [13] and Mumford's proposals [11] for mapping Grenander's generative model approach [14] onto the brain.

The minimum description length approach to unsupervised learning was developed to improve the pattern recognition abilities of artificial neural networks but the simplicity of the wake-sleep learning algorithm makes it biologically interesting. For example, Hasselmo [15] has suggested that that cholinergic inputs to the cortex may modulate the degree of feedforward control of ongoing activity. By a curious coincidence, the idea that the perceptual system uses generative models was advocated by Helmholtz, so we call any neural network that fits a generative model to data by minimizing the free energy in Eq. 5 a "Helmholtz machine".

References

[1] J Rissanen, *Stochastic Complexity in Statistical Inquiry,* (World Scientific, Singapore, 1989).

[2] The description length can be viewed as an upper bound on the negative log probability of the data given the network's generative model, so this approach is closely related to maximum likelihood methods of fitting models to data.

[3] The number of bits required to specify the generative weights should also be included in the description length [1], but we currently ignore it.

[4] GE Hinton and RS Zemel, in *Advances in Neural Information Processing Systems 6.*, JD Cowan, G Tesauro and J Alspector, editors, (Morgan Kaufmann, San Mateo, 1994), 3-10.

[5] An unbiased estimate of the exact gradient is easy to obtain, but the noise in this estimate increases with the size of the network. Alternatively, a mean-field approximation can be made to the stochastic recognition model and the error derivatives can then be computed by a backpropagation process (P Dayan, GE Hinton, RM Neal and RS Zemel, *Neural Computation*, to appear).

[6] This performs stochastic steepest descent in the Kullback-Leibler divergences

$$\sum_{\ell \in L} \sum_{j \in \ell} p_j^\gamma \log \left(p_j^\gamma / q_j^\gamma \right) + (1 - p_j^\gamma) \log \left((1 - p_j^\gamma)/(1 - q_j^\gamma) \right) \tag{9}$$

The cost function in equation 5 contains the same terms but with p and q interchanged leading to an approximation error equal to the asymmetry of the Kullback-Leibler divergences.

[7] In principal components analysis, the hidden representation vector is a linear function of the input vector and the aim is to minimize the squared reconstruction error. From a description length perspective, the cost of describing the hidden activites is ignored, so only the cost of describing the reconstruction errors needs to be minimized. If these errors are coded using a zero-mean Gaussian, the cost of describing them is proportional to their squared values. In competitive learning, only the hidden unit whose weight vector is most similar to the input vector is activated. The reconstruction is just the weight vector of the winning hidden unit, so minimizing the squared distance between the input vector and the weight vector of the winning hidden unit minimizes the description length of the reconstruction error.

[8] G Carpenter and S Grossberg, *Computer Vision, Graphics and Image Processing,* **37**, 54 (1987).

[9] S Ullman, in *Large-Scale Theories of the Cortex,* C Koch and J Davis, editors, (MIT Press: Cambridge, MA, 1994).

[10] M Kawato, H Hayakama and T Inui, *Network,* **4**, 415, 1993.

[11] D Mumford, in *Large-Scale Theories of the Cortex,* C Koch and J Davis, editors, (MIT Press: Cambridge, MA, 1994).

[12] MI Jordan and DE Rumelhart, *Cognitive Science,* **16**, 307, 1992.

[13] HB Barlow, *Neural Computation,* **1**, 295, 1989.

[14] U Grenander, *Lectures in Pattern Theory I, II and III: Pattern Analysis, Pattern Synthesis and Regular Structures,* (Springer-Verlag, Berlin, 1976-1981).

[15] ME Hasselmo and JM Bower, *Trends in Neurosciences,* 16, 218, 1993.

[16] The learning rates were 0.2 for the generative and recognition weights to and from the input units and 0.001 for the other weights. The generative biases of the first hidden layer started at -3.00 and all other weights started at 0.0. The final asymmetric divergence between the network's generative model and the real data was 0.10 bits. The penalty term in Eq. 8 was 0.08 bits.

[17] The training data was 700 examples of each digit from the CEDAR CDROM 1 made available by the US Postal Service Office of Advanced Technology. Starting with the input layer, each of the ten digit specific networks had a 64-16-16-4 architecture. All weights were started at 0 and the learning rate on all connections was 0.01. Training involved 500 sweeps through the 700 examples. For testing, each net was run 10 times to estimate the expected description length of the image. The single network trained on all the digits had a 64-32-32-16 architecture and was also trained for 500 sweeps through the 7000 examples. Using just the base rates of the individual pixels in the input, the average code length per test digit was 59.0 bits; using the ten digit specific networks, the average length was 33.8 bits, including the 3.3 bits required to specify which model was being used; using the single network, the average length was 37.3 bits.

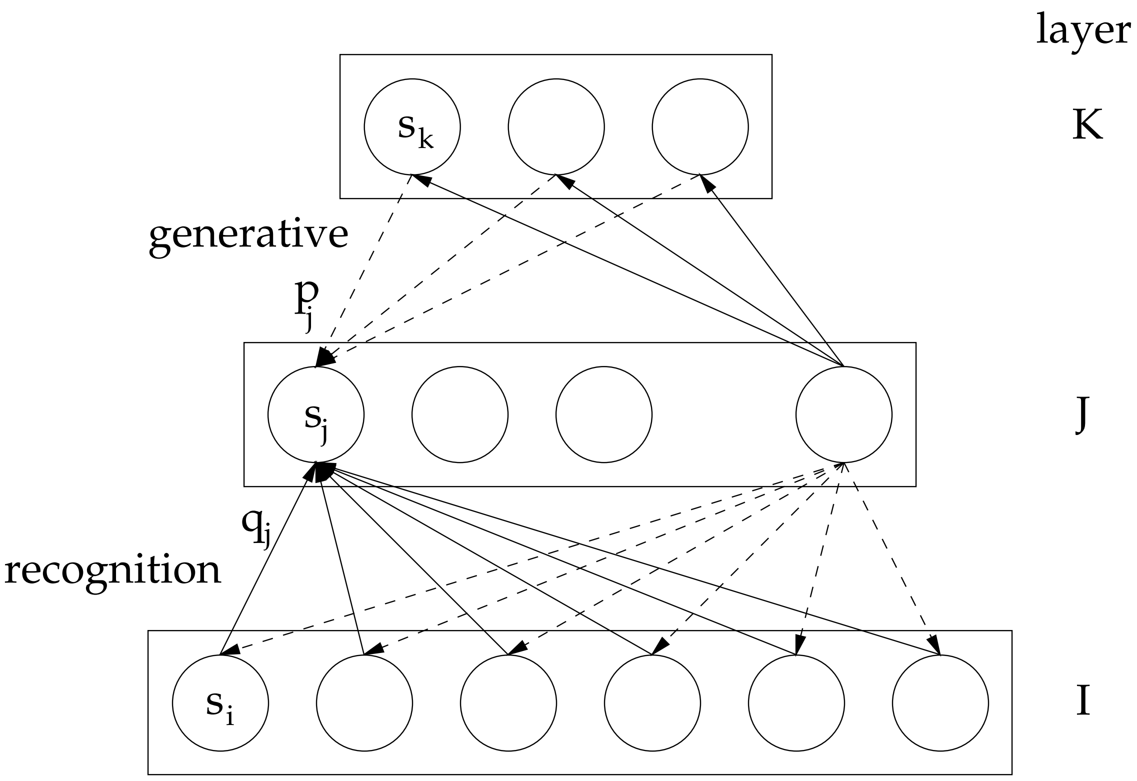
[18] This research was supported by Canadian federal grants from NSERC and IRIS and an Ontario grant from ITRC. GEH is the Noranda fellow of CIAR.

FIGURE CAPTIONS

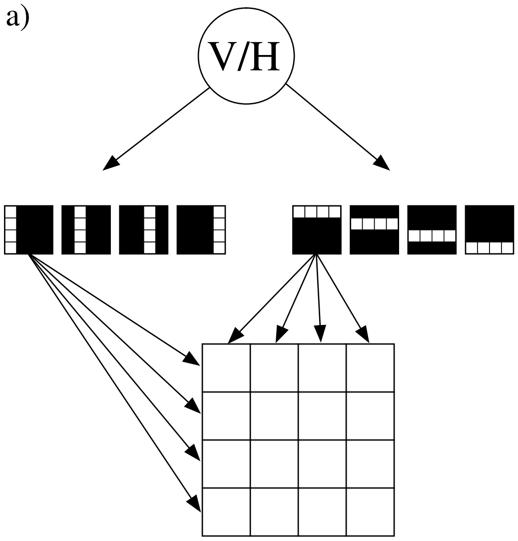
FIGURE 1: A three-layer Helmholtz machine. The bottom layer represents the raw sensory inputs. Units in layers I, J and K are completely interconnected with recognition (solid lines) and generative (dotted lines) connections. The binary activity of unit j in layer J is s_j. q_j is determined by the recognition weights and p_j is determined by the generative weights. When the units are driven bottom-up, the probability that $s_j = 1$ is q_j; when they are driven top-down, the probability is p_j.

FIGURE 2: a) A generative model for 4×4 images. The top level decides whether to use vertical or horizontal bars. The next level decides whether or not each possible bar of the chosen orientation should be present in the image. b) A sample of the images produced by the model in (a) with the ambiguous all-white images removed. A neural net with 16 input units, 8 units in the first hidden layer, and 1 hidden unit in the second hidden layer was trained on 2×10^6 random examples produced by the generative model. After training [16], the probability distribution produced in the sleep phase was almost exactly correct. c) The generative weights to and from the 8 units in the first hidden layer. Positive weights are white, negative weights are black, and the area is proportional to the magnitude. The largest weight shown is 14.1. The generative bias of the unit is shown on the top right of each block and its generative weight from the single unit in the layer above is shown on the top left. The rightmost block shows the generative biases of the input units. To encourage an easily interpretable solution, the generative weights to the input units were constrained to be positive. If they are allowed to go negative, the algorithm finds solutions that produce the correct distribution but in a much more complicated way, and it requires more units in the second hidden layer.

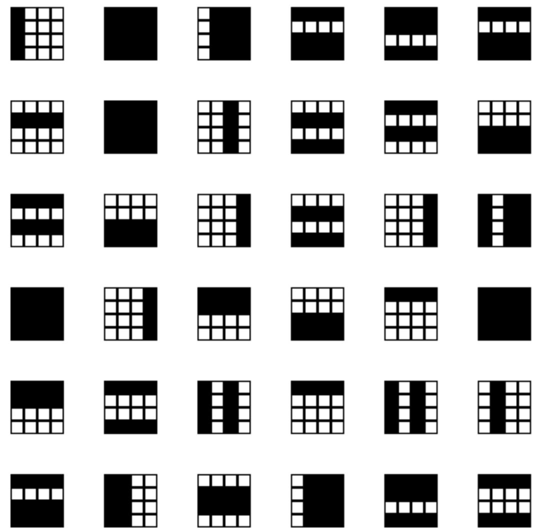
FIGURE 3: Handwritten digits were normalized and quantized to produce 8×8 binary images. 24 examples of each digit are shown on the left. A separate network was trained on each digit class and 24 fantasies from each network are shown on the right. The variations within each digit class are modelled quite well. The error rate was 4.8% when new test images were classified by choosing the network that minimized the description length of the image. On the same data, nearest neighbor classification gave 6.7% errors and backpropagation training of a single supervised net with 10 output units and one hidden layer gave a minimum of 5.6% errors even when we used the test data to optimize the number of hidden units, the training time, and the amount of weight decay [17].


layer
s_k
K
generative
p_j
s_j
J
q_j
recognition
s_i
I

a)



b)



c)