In M. Jenkin and L. R. Harris (editors), *Computational and Biological Mechanisms of Visual Coding*, Cambridge University Press, New York NY, 1997.

A simple algorithm that discovers efficient perceptual codes

Brendan J. Frey, Peter Dayan and Geoffrey E. Hinton
Department of Computer Science, University of Toronto
Toronto, Ontario, M5S 1A4, Canada
{frey@cs.toronto.edu, dayan@ai.mit.edu, hinton@cs.toronto.edu}

Abstract

We describe the "wake-sleep" algorithm that allows a multilayer, unsupervised, neural network to build a hierarchy of representations of sensory input. The network has bottom-up "recognition" connections that are used to convert sensory input into underlying representations. Unlike most artificial neural networks, it also has top-down "generative" connections that can be used to reconstruct the sensory input from the representations. In the "wake" phase of the learning algorithm, the network is driven by the bottom-up recognition connections and the top-down generative connections are trained to be better at reconstructing the sensory input from the representation chosen by the recognition process. In the "sleep" phase, the network is driven top-down by the generative connections to produce a fantasized representation and a fantasized sensory input. The recognition connections are then trained to be better at recovering the fantasized representation from the fantasized sensory input. In both phases, the synaptic learning rule is simple and local. The combined effect of the two phases is to create representations of the sensory input that are efficient in the following sense: On average, it takes more bits to describe each sensory input vector directly than to first describe the representation of the sensory input chosen by the recognition process and then describe the difference between the sensory input and its reconstruction from the chosen representation.

Introduction

Artificial neural networks are typically used as bottom-up recognition devices that transform input vectors into output vectors via one or more layers of hidden neurons. The networks are trained by repeatedly adjusting the weights on the connections to minimize the discrepancy between the actual output of the network on a given training case and the target output supplied by a teacher. During training, the hidden neurons are forced to extract informative features from the input vector in order to produce the correct outputs. Although this kind of learning works well for many practical problems, it is unrealistic as a model of real perceptual learning because it requires a teacher who specifies the target output of the network.

When there is no explicit teacher, it is much less obvious what learning should be trying to achieve. In this chapter, we show how it is possible to get target activities for training the hidden neurons of a network without requiring a teacher. This leads to a very simple, local rule for adjusting the weights on connections. We then show that this simple rule can be viewed as a way of optimizing the coding efficiency of the representations extracted by the network.

A simple stochastic neuron and how to train it

There are many different idealized models of neurons. Here we use a particular model in which the neuron has two possible activity states, 1 and 0. The inputs to the neuron only have a probabilistic influence on its state of activity. Big positive inputs tend to turn it on and big negative ones tend to turn it off, but there is always some chance that the neuron will adopt the less probable of its two states. A neuron, i, first computes its total input, x_i

$$x_i = b_i + \sum_j s_j w_{ji} \tag{1}$$

where b_i is the bias of the neuron, s_j is the binary state of another neuron, and w_{ji} is the weight on the connection from j to i. The probability that the neuron turns on then depends only on x_i

$$p_i = prob(s_i = 1) = \frac{1}{1 + e^{-x_i}} \tag{2}$$

If a teacher supplies target states for the neuron it is relatively straightforward to adjust the weights on the incoming connections so as to maximize the probability that the neuron will adopt the correct target states on all the various training cases. This overall probability is just the product over all training cases of the probability that the neuron adopts each target activity. First we note that maximizing the product of these probabilities is equivalent to maximizing the sum of their logs. This can be done by an online procedure that simply adjusts a weight to improve the log-probability of the target value on each training case as it is presented. Provided these improvements are small and in proportion to the derivative of the log-probability with respect to the weight, the combined effect of the weight changes over all of the different training cases will be to improve the overall log-probability. Fortunately, the derivative of the log-probability is very simple and leads to the following learning rule:

$$\Delta w_{ji} = \epsilon s_j (t_i - p_i) \tag{3}$$

where ϵ is the learning rate and t_i is the target value.

The nice thing about this learning rule is that all of the information it requires is local to a synapse (assuming that the synapse can find out p_i or the actual postsynaptic activity s_i which is a stochastic estimate of p_i). The major difficulty in applying this simple rule to a multilayer network is that we do not generally have targets for the hidden neurons.

How bottom-up and top-down models can train each other

One way of solving the problem of hidden targets for a network that performs vision is to use synthetic images that are generated by a realistic graphics program. This program randomly chooses a sensible representation and then produces a synthetic image using its graphics rules. It can therefore provide pairings of images with their underlying representations and these pairings can be used to train a recognition network to recover the underlying representation from the image. If the graphics program is written as a top-down hierarchical network of binary stochastic neurons (see figure 1), each top-down pass will produce an image together with a set of targets for all the hidden neurons in the layers above. So, given a graphics program in this form, we can train a vision program that inverts the generation process. Of course, images will typically be ambiguous in the sense that the graphics program could have generated them using different representations. This means that, given the image, the best we can hope to do is to assign probabilities to the underlying representations. The stochastic neurons we use are therefore entirely appropriate.

The rule for learning the weight, ϕ_{ij}, on the recognition connection from neuron i to neuron j is:

$$\Delta \phi_{ij} = \epsilon s_i (t_j - q_j) \tag{4}$$

where t_j is the actual binary state of neuron j produced by the generation

$$p_i = \frac{1}{1 + \exp\left(-\theta_{0i} - \sum_j \theta_{ji} s_j\right)}$$

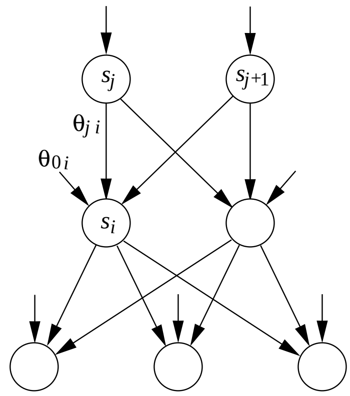


Figure 1: A top-down generative model composed of stochastic neurons. The model can be used to generate data in the bottom layer. Because the neurons are stochastic, many different data vectors can be generated by the same model. A neuron is turned on with probability p_i so its binary state depends on its generative bias and on the binary states already chosen for neurons in the layer above. At the top layer the neurons have uncorrelated states. The correlations between neurons become more and more complicated in lower layers.

process and q_j is the probability that neuron j would be turned on by the recognition connections and the recognition bias, ϕ_{0j}:

$$q_j = \frac{1}{1 + \exp(-\phi_{0j} - \sum_i s_i \phi_{ij})} \tag{5}$$

So far we seem to have merely exchanged one problem for another. In order to get the hidden targets to train a network to do vision, we have appealed to a network that can already do graphics. But where does this network come from?

Suppose we already have a multilayer bottom-up vision network that can recover underlying representations from images. It would then be easy to train a top-down graphics network since the vision network can provide training examples in which images are paired with the higher level representations from which they should be generated.

The rule for learning the weight, θ_{ji}, on the generative connection from neuron j to neuron i is:

$$\Delta\theta_{ji} = \epsilon s_j(t_i - p_i) \tag{6}$$

where t_i is the actual binary state of neuron i produced by the recognition process and p_i is the probability that neuron i would be turned on by the generative connections and generative bias.

So, given a graphics network we can train a vision network and vice versa. Now comes the leap of faith. Given a *poor* graphics network, we use the images it generates to train a vision network. When this vision network is then applied *to real images* we can use it to improve the graphics network. Intuitively, the reason the two networks can improve each other is that the mutual training keeps decreasing the discrepancy between the distribution of the real images and the distribution of the fantasies generated by the graphics network. As we shall see later, this intuitive reasoning is only approximately correct.

A statistical perspective

There are two equivalent but very different approaches to analyzing what the wake-sleep algorithm does. In one approach we view the generative model as primary and the aim is to adjust the top-down generative connections so as to maximize the likelihood that the generative model would produce the observed data. This type of maximum likelihood model fitting is a standard statistical approach. In order to perform the maximization, we need to know how the current generative model explains each data vector. An explanation of a data vector is an assignment of 1's and 0's to all of the hidden neurons in the network. For a given set of generative weights, each possible explanation will have some posterior probability of having generated each data vector and these probabilities are needed in order to adjust the generative weights correctly. This is tricky because the number of possible explanations is exponential in the number of hidden neurons so it is completely intractable to compute all those posterior probabilities.

The recognition connections can be viewed as a way of approximating the posterior probabilities of explanations. Given a data vector, a bottom-up pass through the network will produce a particular explanation. Since the neurons are stochastic, another bottom-up pass may well produce a different explanation for the same data vector. The recognition connections therefore determine a probability distribution over explanations for each data vector. Although this is not the true posterior distribution, the learning in the sleep phase makes it approximate the posterior distribution which is good enough to allow the generative weights to be improved. A more rigorous account from this perspective is given by Dayan, Hinton, Neal and Zemel (1995). In this chapter we focus on a coding perspective (Hinton and Zemel 1994; Hinton, Dayan, Frey and Neal 1995).

The minimum description length perspective

Consider the following communication game: A *sender* must communicate an ensemble of binary data vectors to a *receiver* using as few bits as possible. The simplest method is for the sender to treat all the components of each data vector as independent and to communicate each component separately. The cost of communicating the binary value of a component depends on how often that component is on in the whole ensemble of data vectors. It can be shown that the best possible code for an event that occurs with probability p requires at least $-\log_2 p$ bits. Moreover, by using clever coding techniques it is always possible to approach this limit, so to simplify matters we shall simply assume that an event with probability p can be communicated using $-\log_2 p$ bits. This is only possible if the sender and the receiver both know the value of p, which would require some additional communication. For large ensembles, this additional communication can be amortized across many data vectors so it is negligible and we shall ignore it here. So, taking into account the two possible states of component i of the data vector, the cost of communicating that component is

$$C_i = -s_i \log_2 p_i - (1 - s_i) \log_2(1 - p_i) \tag{7}$$

where p_i is the probability that it is on and s_i is its actual binary state.

If the components of the data vector are not independent it is wasteful to communicate them separately and independently. It is more efficient to transform the data into a different representation in which the components *are* approximately independent. Then this representation is communicated together with any errors that occur when the raw data is reconstructed from the representation. This gives rise to an interesting criterion for what constitutes a good representational scheme. We simply measure the cost of communicating the representations of all the data vectors in the ensemble plus the cost of communicating the reconstruction errors. The smaller this combined description length, the better the representational scheme. This is a simplified version of the minimum description length (MDL) approach introduced by Rissanen (1989). In MDL it is usually important to include the cost of communicating the representational scheme itself since this is needed in order to reconstruct each data vector from its representation. For our current purposes we ignore this additional cost.

A simple idea about how to communicate images may make the MDL perspective clearer. Instead of sending the individual pixel intensities, we could first extract edges from the image and then extract instances of objects from the edges as shown in figure 2. To communicate the image we first send the top-level representation in terms of instantiated objects. If the number of possible object types is limited and if each type of object only has a few degrees of freedom in how it can be instantiated (*eg.* position, size and orientation) it will be much cheaper to send the top-level representation than to send the raw image. Once the receiver knows the instantiated objects he can use a top-down generative model to predict where the edges are. More specifically, his predictions can take the form of a probability distribution across the various possible instantiated edges. If the predictions are good, they will assign fairly high probabilities to the actual edges. Since both the sender and the receiver can construct the predicted probability distributions, these distributions can be used for communicating the edges. This should be much more efficient than communicating the edges under the assumption that all possible edges are equally likely. In effect, an edge is only expensive to communicate if it violates the expectations created from the layer above by the top-down model. Finally, the edges (plus the contrast across them) can be used to create expectations for the pixel intensities so that intensities which meet these expectations can be communicated cheaply.

For an appropriate ensemble of images, this whole scheme is an efficient way to compress the images for communication. But even if we are not interested in communication, the efficiency of the compression can be used as a criterion for whether the representational scheme is any good. The MDL criterion allows us to take an ensemble of images and decide that it really is sensible to code them in terms of edges and objects even if we have no prior bias towards this type of representation[1].

Quantifying the description length

Figure 3 shows how a multi-layer network could be used to communicate data vectors to a receiver. To communicate an individual data vector, the sender first performs a bottom-up recognition pass which assigns a binary value s_j to each hidden neuron[2]. This value is chosen stochastically using the recognition probability distribution for the neuron $\{q_j, 1-q_j\}$, where q_j

[1]To be fair, we need to also take into account the cost of communicating the generative model itself — how object instantiations predict edges and how edges predict pixel intensities. This raises some tricky issues about what probability distribution to use for communicating the generative models, but these difficulties can be handled.

[2]Notice that the states of neurons within one layer are conditionally independent given the particular binary states chosen for neurons in the layer below, but this still allows the states of neurons in the top layer to be far from independent given the data.

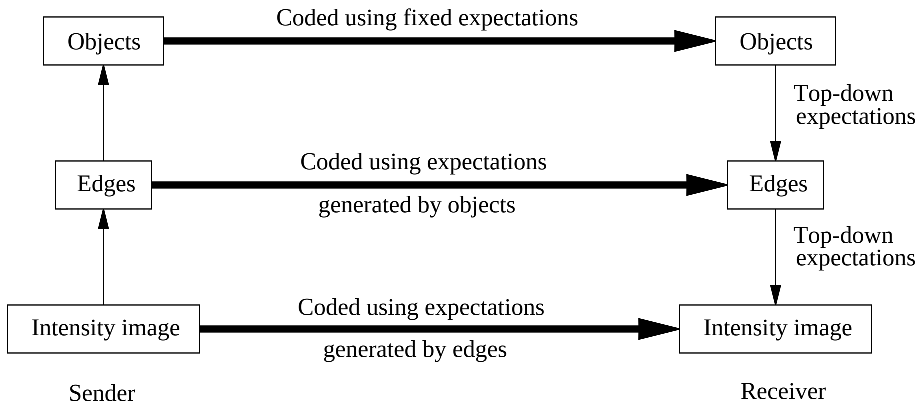


Figure 2: An illustration of the relationship between good representations and economical communication. If the images are well-modelled in terms of objects and edges, then they can be communicated cheaply by using this representation. The top-down expectations produced by the generative model operating on representations at one level will assign high probabilities to the data actually observed at the next level down, so by using the top-down expectations it is possible to communicate the data cheaply.

is determined by the states of the neurons in the layer below and the recognition weights. The binary states of neurons are then sent to the receiver starting with the top layer and working down. Each top-layer neuron s_i has a generative bias, θ_{0i}, that adapts during learning. Applying the logistic function to this bias yields the generative probability p_i that the neuron is on. Assuming that the sender and receiver use the distribution $\{p_i, 1-p_i\}$ as an agreed prior distribution for communicating the binary state s_i, the cost is given by equation 7[3]. For neurons in the middle and bottom layers, the generative probability p_j depends not only on the generative bias of the neuron but also on the states of neurons in the higher layers and the generative weights from those neurons.

Summing over all neurons, the number of bits that would have to be sent across a channel to communicate a data vector is:

[3]To approach this theoretical limit it is necessary to combine the states of many neurons into one message, so what would actually have to be sent across a channel would be much more complicated than just sending the binary value s_i.

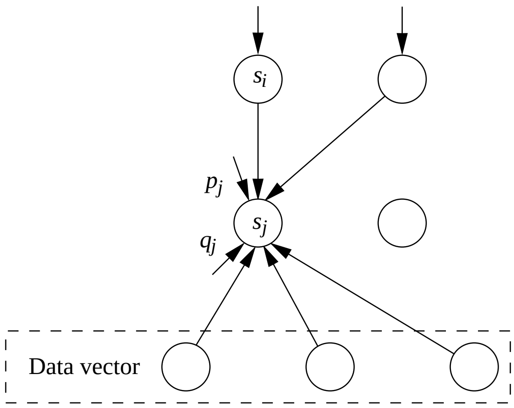


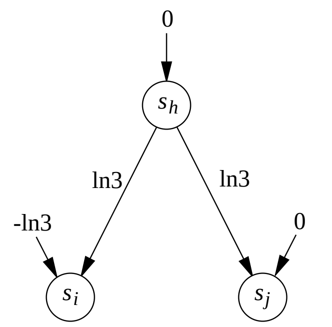
Figure 3: A multi-layer network can be used to communicate data vectors. Binary values for the hidden neurons are obtained by a bottom-up sweep of the recognition network. Assuming that the sender and receiver have identical copies of the generative weights, the generative probability p_j can be used to encode each binary activity.

$$C = \sum_i C_i = -\sum_i \left(s_i \log_2 p_i + (1 - s_i) \log_2 (1 - p_i)\right) \tag{8}$$

C is a stochastic quantity because it depends on the states s_i that are stochastically picked during the bottom-up recognition pass. Apart from this, the only peculiar property of equation 8 is that it is wrong, for reasons explained in the next section.

The bits-back argument

Consider the simple network shown in figure 4a which might be obtained by training on the data set shown in figure 4b. Since there is only one hidden neuron, the network has two alternative ways of representing each data vector. The generative bias of the hidden neuron, h, is 0 so its generative probability is $p_h = 0.5$. It therefore costs 1 bit to communicate the state of the hidden neuron whichever representation is used. The generative probabilities for the two input neurons are $(0.5, 0.75)$ if neuron h is on and $(0.25, 0.5)$ if it is off. Either way, if the data vector is $(1, 0)$ it costs an additional 3 bits to communicate the states of the two data neurons once s_h has



(a)

1 1	1 1
0 1	1 1
0 1	1 1
0 1	1 0
1 0	0 1
0 0	0 1
0 0	0 1
0 0	0 0

(b)

Figure 4: a) A simple example of a generative model. The logistic of $-ln3$ is $1/4$, so it costs 2 bits to communicate $s_i = 1$ when $s_h = 0$ because the top-down distribution $(p_i, 1-p_i)$ under which the state of s_i must be coded is then $(1/4, 3/4)$. Similarly, when s_h is off it only takes 1 bit to communicate the state of s_j. b) A data-set which is well-modelled by the simple network. Note that the frequency of the vector $(1, 0)$ in the dataset is $1/8$ so the optimal code takes 3 bits.

been communicated. The network has two equally good ways of coding the data vector $(1, 0)$ and each way takes a total of 4 bits. Now we show a rather surprising result: Two 4 bit methods are as good as one 3 bit method.

We start with a vague intuitive argument. If we can send the data using two different messages each of which costs 4 bits, why can't we save a bit by being vague about which of the two messages we are actually sending? This intuition can be made precise using the scheme illustrated in figure 5. Imagine that in addition to the data vector, the sender and the receiver are also trying to communicate some totally separate information across the same channel. This other information has already been efficiently encoded into a string of binary digits that appears totally random. We show how to communicate both the data vector and the first bit of this random bit-string using only the 4 bits that are required to communicate the data vector. So the net cost of sending the data vector is really only 3 bits which is just what it should be for a vector that occurs one eighth of the time.

During the bottom-up recognition pass, the sender discovers that $q_h = 0.5$ because there are two equally good choices for s_h. Instead of using a random number generator to make the decision, the sender simply uses the first

Figure 5: An illustration of how to make effective use of the freedom of choice available when there are several alternative ways of coding the data. Another source of information is used to decide between alternative codes. This allows the other information to be communicated as well as the data. So the true cost of communicating the data is reduced by the amount of information needed to choose between the alternative codes.

bit in the other message since these bits conveniently have a probability of 0.5 of being on. Using whatever value of s_h gets selected, the sender then communicates s_h and the data vector for a total cost of 4 bits. Assuming the receiver has access to the same recognition model as the sender, the receiver can now recreate the choice that the sender was faced with. He also knows what value was chosen for s_h so he can figure out the first bit of the other message.

In general, the various alternative representations will not give equal description lengths for the data vector, and the recognition probabilities will not be 0.5. In the general case, there are many alternative codes, α, each requiring E_α bits, where E_α includes the cost of communicating the reconstruction error. If we have a probability Q_α of picking each code, the expected number of bits that are used to send the data is $\sum_\alpha Q_\alpha E_\alpha$ and the expected number of bits from the other message used to pick a single code from the Q distribution is the entropy of this distribution, $-\sum_\alpha Q_\alpha log_2 Q_\alpha$. So the net cost of communicating a data vector is:

$$F = \sum_\alpha Q_\alpha E_\alpha - \left(- \sum_\alpha Q_\alpha log_2 Q_\alpha \right) \tag{9}$$

Frey and Hinton (1996) describe an actual implementation of this coding method. Equation 9 is well known in physics. We interpret α as a particular configuration of a physical system, E_α as its energy measured in appropriate units and Q as a probability distribution over configurations. F is then the Helmholtz free energy of the system at a temperature of 1. We call a model that uses separate recognition connections to minimize the Helmholtz free energy in equation 9 a "Helmholtz machine".

The probability distribution that minimizes F is the Boltzmann distribution:

$$Q_\alpha = \frac{e^{-E_\alpha}}{\sum_\gamma e^{-E_\gamma}} \tag{10}$$

The generative weights and biases define the "energy" of each representation and the Boltzmann distribution is then the best possible recognition distribution to use. But F is perfectly well defined for any other recognition distribution over representations and it is generally not worth the effort of computing the full Boltzmann distribution. Our simple bottom-up recognition network builds up the Q distribution as a product of lots of $\{q_i, 1-q_i\}$ distributions within each hidden layer.

When we take into account the savings that occur when the generative model allows many alternative ways of representing the same data, the coding perspective tells us to minimize free energy. If we also decide to restrict the recognition model to using a product distribution in each hidden layer, the free energy can be rewritten as:

$$F = \sum_i \left(q_i \log_2 \frac{q_i}{p_i} + (1-q_i) \log_2 \frac{1-q_i}{1-p_i} \right) \tag{11}$$

where i is an index over all of the neurons, and most of the p's and q's are stochastic quantities that depend on choices of s in higher or lower layers. A pleasing aspect of equation 11 is that each neuron makes a separate additive contribution which is just the asymmetric divergence between the recognition and generative probability distributions for the state of the neuron. A less pleasing aspect of the equation is that changes in the recognition weights in lower layers cause changes in q's and hence p's in higher layers, so the derivatives of the free energy with respect to the recognition weights are complicated. Dayan *et al.* (1995) show how the derivatives can be approximated accurately and efficiently using a backpropagation scheme if the recognition process is modified. However, this is much less biologically plausible than the simple wake-sleep algorithm.

Does the wake-sleep algorithm minimize free energy?

All that remains to be shown is that the simple, local wake-sleep algorithm defined by equations 4 and 6 is actually performing gradient descent in the free energy. For the wake phase, this is easy because the q's are unaffected by changes in the generative weights. When averaged over the stochastic choices of states for the hidden neurons, the right hand side of equation

6 is exactly $-\epsilon$ times the derivative of the free energy with respect to the generative weight θ_{ji}. So wake-phase learning does exactly the right thing.

The sleep phase is more problematic for two reasons. First, the sleep phase uses fantasy data produced by the generative model instead of real data. Early on in the learning the fantasies will be quite different from the real data. Later in the learning, however, the distribution of fantasies comes to resemble the distribution of real data. The second problem is more serious. Instead of performing gradient descent in the free energy, the sleep phase performs descent in a similar expression with the p's and q's interchanged:

$$G = \sum_i \left(p_i \log_2 \frac{p_i}{q_i} + (1 - p_i) \log_2 \frac{1 - p_i}{1 - q_i} \right) \tag{12}$$

Fortunately, the free energy and G have very similar gradients when the p's and q's have soft values that are not close to 1 or 0. Extensive simulations have shown that so as long as we avoid large weights, following the gradient of G almost always reduces the free energy. Naturally, during on-line learning there are stochastic fluctuations in the free energy because the binary states produced by the recognition process are stochastic.

An example: Extracting structure from noisy images

An interesting problem relevant to vision is that of extracting independent horizontal and vertical bars from an image (Foldiak 1990; Saund 1995; Zemel 1993; Dayan and Zemel 1995; Hinton *et al.* 1995). Figure 6 shows 48 examples of the binary images we are interested in. Each image is produced by randomly choosing between horizontal and vertical orientations with equal probability. Then, each of the 16 possible bars of the chosen orientation is independently instantiated with probability 0.25. Finally, additive noise is introduced by randomly turning on with a probability of 0.25 each pixel that was previously off. So, the graphics program used to produce the training data has three levels of hierarchy: the first and lowest level represents pixel noise, the second represents bars that consist of groups of 16 pixels each, and the third represents the overall orientation of the bars in the image.

Using the wake-sleep algorithm, we trained a Helmholtz machine that has 4 top-layer neurons, 36 middle-layer neurons, and 256 bottom-layer image neurons. Learning is performed through a series of iterations, where each iteration consists of one bottom-up wake phase sweep used to adjust the generative connections and one top-down sleep phase sweep used to adjust the recognition connections. Every 5000 iterations, an estimate of the free

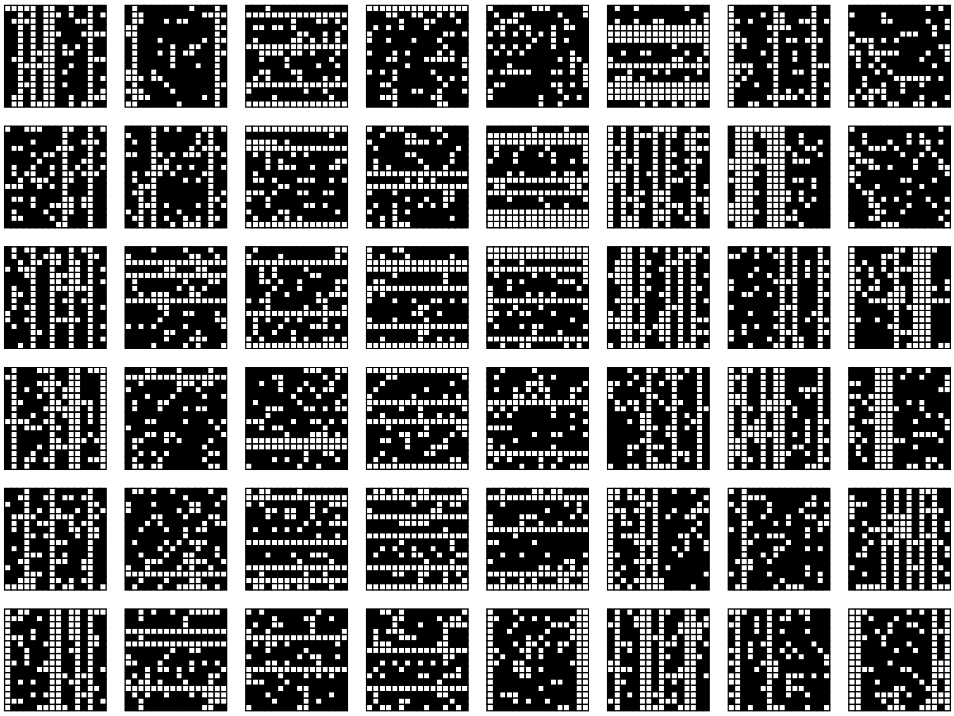


Figure 6: Examples of training images produced by a graphics program with three levels of hierarchy. First, an orientation (*ie.*, horizontal or vertical) is randomly chosen with fair odds. Second, each bar of the chosen orientation is randomly instantiated with probability 0.25. Third, additive noise is introduced by randomly turning on with a probability of 0.25 each pixel that was previously off.

energy and the variance of this estimate are computed. To do this, 1000 recognition sweeps are performed without learning. During each recognition sweep, binary values for the hidden neurons are obtained for the given training image. The negative log-likelihood of these values under the recognition model gives an unbiased estimate of the second (entropy) term in the free energy of equation 9. The negative log-likelihood of the values of *all* the neurons under the generative model gives an unbiased estimate of the first (energy) term in the free energy of equation 9. In this way we obtain 1000 independent, identically distributed, noisy unbiased estimates of the free energy. The average of these values gives a less noisy unbiased estimate of the free energy. Also, the variance of this estimate is estimated by dividing the sample variance by 999.

Since we are interested in solutions where the generative model can construct the image by adding features, but cannot remove previously instantiated features, we constrain the middle-to-bottom connections to be positive by setting to zero any negative weights every 20th learning iteration. In order to encourage a solution where each image can be succinctly described by the minimum possible number of causes in the middle layer, we initialize the middle-layer generative biases to -4.0 which favors most middle-layer neurons being off on average. All other weights and biases are initialized to zero. For the first 100,000 iterations, we use a learning rate of 0.1 for the generative connections feeding into the bottom layer and for the recognition connections feeding into the middle layer; the remaining learning rates are set to 0.001. After this, learning is accelerated by setting all learning rates to 0.01.

Figure 7 shows the learning curve for the first 300,000 iterations of a simulation consisting of a total of 1,000,000 iterations. Aside from several minor fluctuations, the wake-sleep algorithm minimizes free energy in this case. Eventually, the free energy converges to the optimum value (170 bits) shown by the solid line. This value is computed by estimating the free energy for the graphics program that was used to produce the training images (*ie.*, in this case, how many random bits it uses when generating an image).

By examining the generative weights after learning, we see that it has extracted the correct 3-level hierarchical structure. Figure 8 shows the generative incoming weights, incoming biases, and outgoing weights for the middle-layer neurons. A black blob indicates a negative weight and a white blob indicates a positive weight; the area of each blob is proportional to the magnitude of the weight (the largest weight shown has a value of 7.77 and the smallest a value of -7.21). There are 36 blocks arranged in a 6x6 grid and each block corresponds to a middle-layer neuron. The 4 blobs at the upper-left of a block show the weights from each of the top-layer neurons to the corresponding middle-layer neuron. The single blob at the upper-right of a block shows the bias for the corresponding middle-layer neuron. The 16x16 matrix that forms the bulk of a particular block shows the weights from the corresponding middle-layer neuron to the bottom-layer image. The outgoing weights clearly indicate that 32 of the 36 middle-layer neurons are used by the network as "bar neurons" to represent the 32 possible bars. These bar neurons are controlled mainly by the right-most top-layer "orientation" neuron – the weights from all the other top-layer neurons are nearly zero. If the orientation neuron is off, the probability of each bar neuron is determined mainly by its bias. Vertical bar neurons have significantly negative biases,

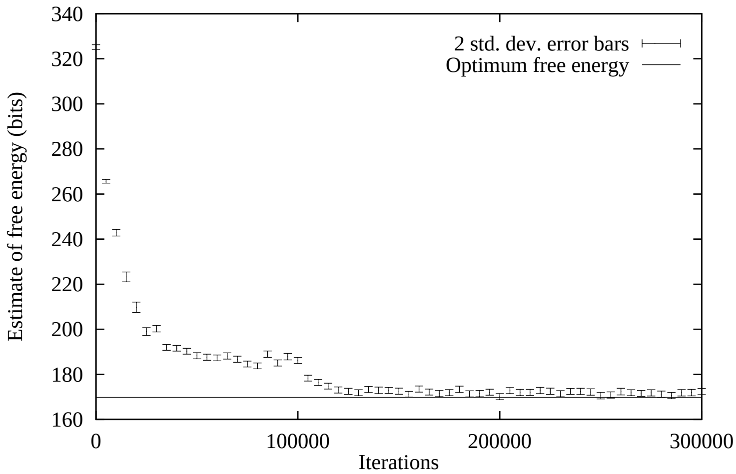


Figure 7: Variation of free energy with the number of wake-sleep learning iterations.

causing them to remain off if the orientation neuron is off. Horizontal bar neurons have only slightly negative biases, causing them to fire roughly 25% of the time if the orientation neuron is off. The vertical bar neurons have significantly positive incoming weights from the orientation neuron, so that when the orientation neuron is on the net inputs to the vertical bar neurons are slightly negative, causing them to fire roughly 25% of the time. The horizontal bar neurons have significantly negative incoming weights from the orientation neuron, so that when the orientation neuron is on the net inputs to the horizontal bar neurons are significantly negative, causing them to remain off. The 4 middle-layer neurons that are not used to represent bars are usually inactive, since they have large negative biases and all incoming weights are negative. Because the bottom-layer biases (not shown) are only slightly negative, a pixel that is not turned on by a bar neuron still has a probability of 0.25 of being turned on. This accounts for the additive noise.

Once learned, the recognition model can nonlinearly filter the noise from a given image, detect the underlying bars, and determine the orientation of these bars. To clean up each of the training images shown in figure 6, we apply the learned recognition model to the image and obtain middle-layer activities which reveal an estimate of which bars are on. The results of this procedure are shown in figure 9 and clearly show that the recognition model is capable of filtering out the noise. Usually, the recognition model correctly

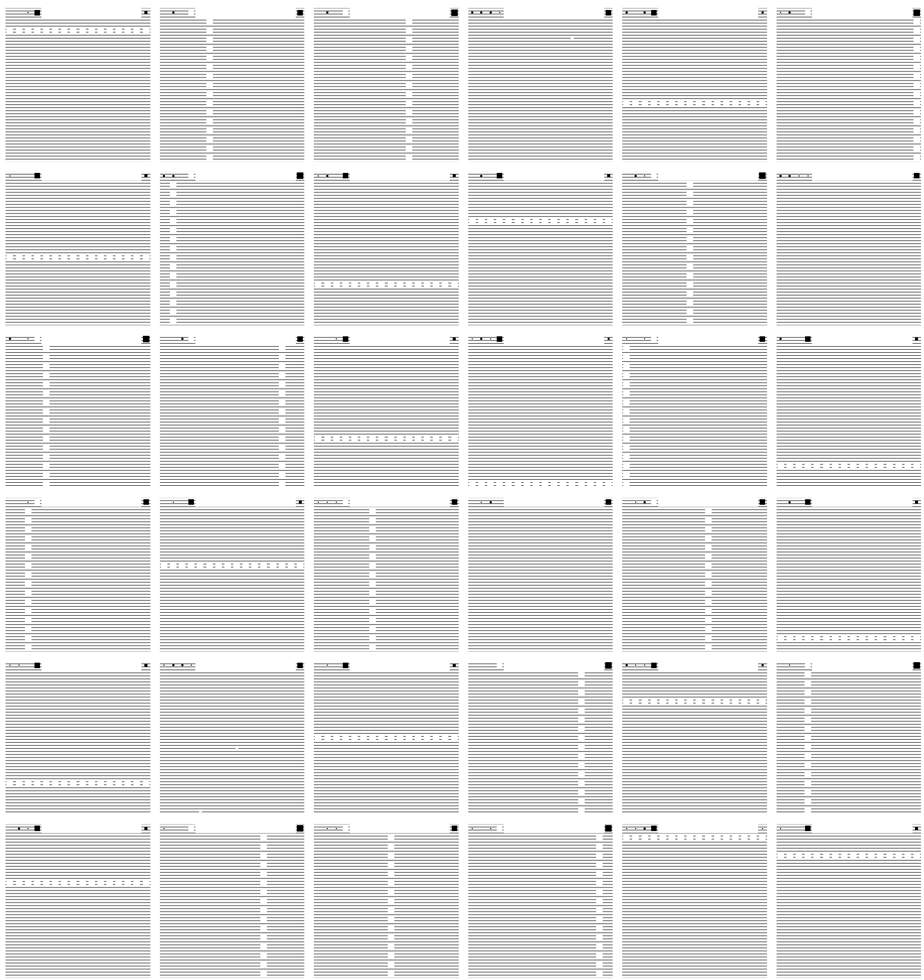


Figure 8: Weights for connections that feed into and out of the middle-layer neurons. A black blob indicates a negative weight and a white blob indicates a positive weight; the area of each blob is proportional to the weight's magnitude (the largest weight shown has a value of 7.77 and the smallest a value of -7.21).

identifies which bars were instantiated in the original image. Occasionally, a bar is not successfully detected. In two cases a bar is detected that has an orientation that is the opposite of the dominant orientation; however, usually the recognition model preserves a single orientation. Inspection of

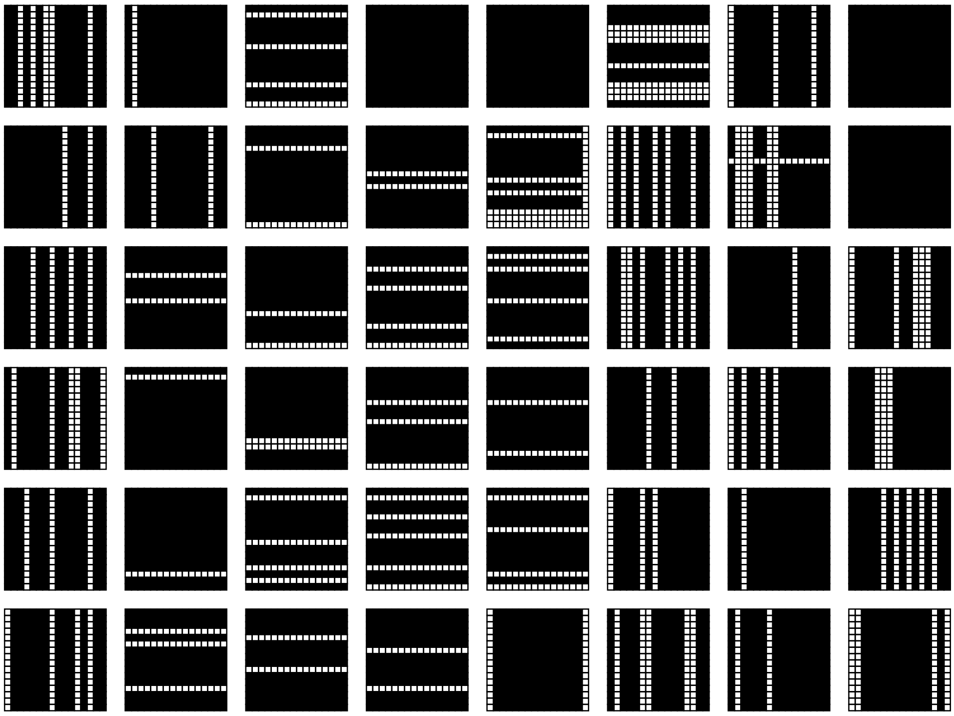


Figure 9: Filtered versions of the training examples from figure 6 extracted using the the learned recognition model.

the original noisy training images for these two cases shows that aside from the single-orientation constraint, there is significant evidence that the mistakenly detected bars *should* be on. Further training reduces the chance of misdetection.

If all the weights and biases are initialized to zero, the middle-to-bottom connections are not constrained to be positive, and all the learning rates are set to 0.01, the trained network is not able to separate the horizontal bars from the vertical bars. Figure 10 shows the middle-to-bottom weights after 5,000,000 learning iterations. The black bars indicate that some middle-layer neurons are capable of uninstantiating bars that may be instantiated by other neurons. Although it is imaginable that such a complex scheme is still capable of modelling the training images, the free energy for this trained network is 190 bits – significantly higher than the optimum value of 170 bits.

Although this bar extraction problem may seem simple, it must be kept

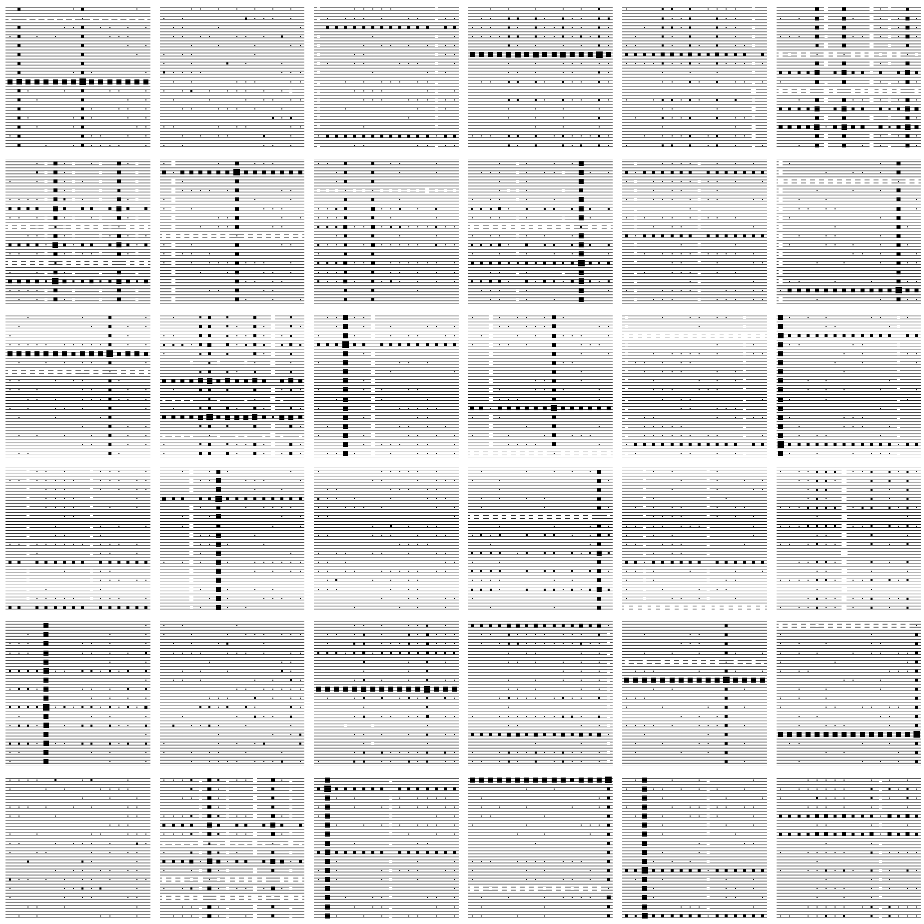


Figure 10: Weights for connections that feed out of the middle-layer neurons after learning without special initialization of the weights, without different learning rates between layers, and without positive weight constraints.

in mind that the network is not given *a priori* topology information – a fixed random rearrangement of the pixels in the training images would not change the learning performance of the network. So, insofar as the network is concerned, the actual training examples look like those shown in figure 11 which were produced by applying a fixed random rearrangement to the pixels in the images from figure 6.

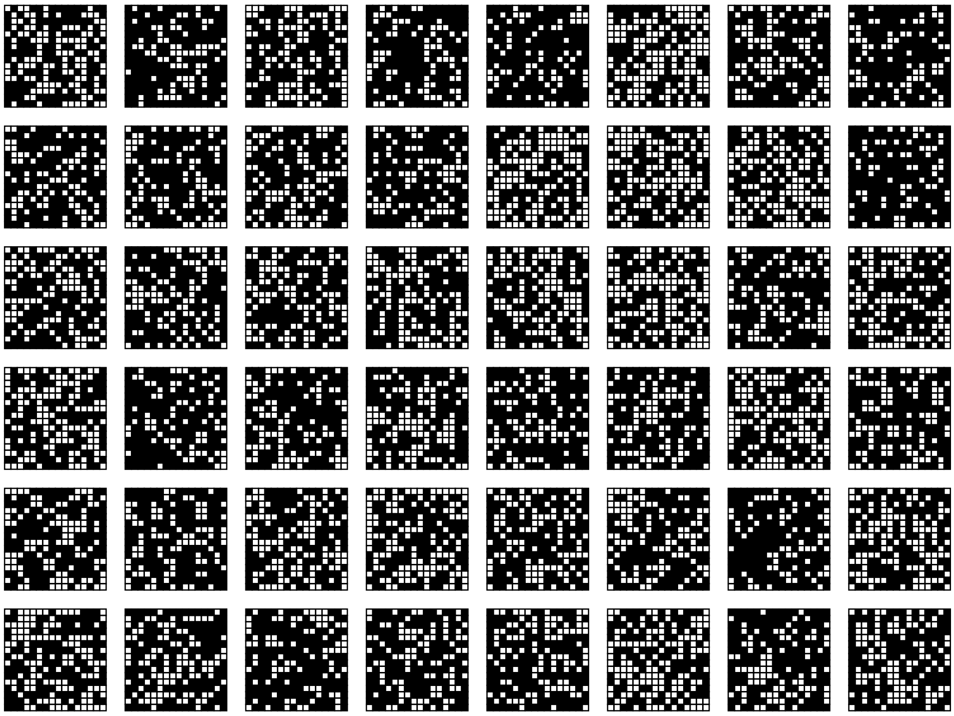


Figure 11: Training examples from figure 6 after a fixed random rearrangement of the pixels has been applied. These are indicative of the difficulty of the bars problem in the absence of a topological prior that favors local intensity coherence.

Summary

We have shown that a very simple local algorithm for adjusting synapse strengths is capable of constructing sensible hierarchical representations from the sensory input alone. The algorithm performs approximate gradient descent in a measure of the coding efficiency of the representations. The top-down generative connections are required both for defining the measure that is to be optimized and for generating fantasies to train the recognition weights.

Although the wake-sleep algorithm performs moderately well, we believe that there is room for considerable further development in order to make it more useful in practice (Frey, Hinton and Dayan 1996) and more realistic as

a neural model (Dayan and Hinton 1996). The simple form of the algorithm presented here is unable to account for top-down influences *during* recognition (Palmer 1975) because it lacks top-down recognition weights. Nor does it explain the role of lateral connections within a cortical area which seem to be important in modelling a number of psychophysical phenomena (Somers, Nelson and Sur 1995). The whole approach would be more plausible if the learning could be driven by the discrepancy between recognition probabilities and generative expectations *during recognition*. How to achieve this without making recognition tediously slow is an open issue.

Acknowledgements

This research was funded by the Information Technology Research Center and Natural Sciences and Engineering Research Council. Hinton is a fellow of the Canadian Institute for Advanced Research. We thank Radford Neal, Rich Zemel and Carl Rasmussen for helpful discussions.

References

Dayan, P., and Hinton, G. E. 1996. Varieties of Helmholtz Machine. *Neural Networks*, to appear.

Dayan, P., Hinton, G. E., Neal, R. M., and Zemel, R. S. 1995. The Helmholtz Machine. *Neural Computation* **7**, 889-904.

Dayan, P., and Zemel, R. S. 1995. Competition and multiple cause models. *Neural Computation* **7**, 565-579.

Foldiak, P. 1990. Forming sparse representations by local anti-Hebbian learning. *Biological Cybernetics* **64**, 165-170.

Frey, B. J., and Hinton, G. E. 1996. Free energy coding. To appear in *Proceedings of the Data Compression Conference 1996*, IEEE Computer Society Press.

Frey, B. J., Hinton, G. E., and Dayan, P. 1996. Does the wake-sleep algorithm produce good density estimators? In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo (editors), *Advances in Neural Information Processing Systems 8*, MIT Press, Cambridge, MA.

Hinton, G. E., Dayan, P., Frey, B. J., and Neal, R. M. 1995. The wake-sleep algorithm for unsupervised neural networks. *Science* **268**, 1158-1161.

Hinton, G. E., and Zemel, R. S. 1994. Autoencoders, Minimum Description Length, and Helmholtz Free Energy. In J. D. Cowan, G. Tesauro, and J. Alspector (editors), *Advances in Neural Information Processing Systems 6*, Morgan Kaufmann, San Mateo, CA.

Palmer, S. E. 1975. Visual perception and world knowledge: Notes on a model of sensory-cognitive interaction. In D. A. Norman and D. E. Rumelhart (editors), *Explorations in Cognition*, W. H Freeman and Co., San Francisco, CA.

Rissanen, J. 1989. *Stochastic complexity in Statistical Inquiry*, World Scientific, Singapore.

Saund, E. 1995. A multiple cause mixture model for unsupervised learning. *Neural Computation* **7**, 51-71.

Somers, D. C., Nelson, S. B., and Sur, M. 1995. An emergent model of orientation selectivity in cat visual cortical simple cells. *Journal of Neuroscience* **15**, 5448-5465.