CHAPTER 7

Learning and Relearning in Boltzmann Machines

G. E. HINTON and T. J. SEJNOWSKI

Many of the chapters in this volume make use of the ability of a parallel network to perform cooperative searches for good solutions to problems. The basic idea is simple: The weights on the connections between processing units encode knowledge about how things normally fit together in some domain and the initial states or external inputs to a subset of the units encode some fragments of a structure within the domain. These fragments constitute a problem: What is the whole structure from which they probably came? The network computes a "good solution" to the problem by repeatedly updating the states of units that represent possible other parts of the structure until the network eventually settles into a stable state of activity that represents the solution.

One field in which this style of computation seems particularly appropriate is vision (Ballard, Hinton, & Sejnowski, 1983). A visual system must be able to solve large constraint-satisfaction problems rapidly in order to interpret a two-dimensional intensity image in terms of the depths and orientations of the three-dimensional surfaces in the world that gave rise to that image. In general, the information in the image is not sufficient to specify the three-dimensional surfaces unless the interpretive process makes use of additional plausible constraints about the kinds of structures that typically appear. Neighboring pieces of an image, for example, usually depict fragments of surface that have similar depths, similar surface orientations, and the same reflectance. The most plausible interpretation of an image is the one that satisfies

constraints of this kind as well as possible, and the human visual system stores enough plausible constraints and is good enough at applying them that it can arrive at the correct interpretation of most normal images.

The computation may be performed by an iterative search which starts with a poor interpretation and progressively improves it by reducing a cost function that measures the extent to which the current interpretation violates the plausible constraints. Suppose, for example, that each unit stands for a small three-dimensional surface fragment, and the state of the unit indicates the current bet about whether that surface fragment is part of the best three-dimensional interpretation. Plausible constraints about the nature of surfaces can then be encoded by the pairwise interactions between processing elements. For example, two units that stand for neighboring surface fragments of similar depth and surface orientation can be mutually excitatory to encode the constraints that each of these hypotheses tends to support the other (because objects tend to have continuous surfaces).

RELAXATION SEARCHES

The general idea of using parallel networks to perform relaxation searches that simultaneously satisfy multiple constraints is appealing. It might even provide a successor to telephone exchanges, holograms, or communities of agents as a metaphor for the style of computation in cerebral cortex. But some tough technical questions have to be answered before this style of computation can be accepted as either efficient or plausible:

- Will the network settle down or will it oscillate or wander aimlessly?

- What does the network compute by settling down? We need some characterization of the computation that the network performs other than the network itself. Ideally we would like to be able to say what *ought* to be computed (Marr, 1982) and then to show that a network can be made to compute it.

- How long does the network take to settle on a solution? If thousands of iterations are required the method becomes implausible as a model of how the cortex solves constraint-satisfaction problems.

- How much information does each unit need to convey to its neighbors? In many relaxation schemes the units communicate accurate real values to one another on each iteration. Again this is implausible if the units are intended to be like cortical neurons which communicate using all-or-none spikes. To send a real-value, accurate to within 5%, using firing rates requires about 100 ms which is about the time allowed for the whole iterative process to settle down.

- How are the weights that encode the knowledge acquired? For models of low-level vision it is possible for a programmer to decide on the weights, and evolution might do the same for the earliest stages of biological visual systems. But if the same kind of constraint-satisfaction searches are to be used for higher level functions like shape recognition or content-addressable memory, there must be some learning procedure that automatically encodes properties of the domain into the weights.

This chapter is mainly concerned with the last of these questions, but the learning procedure we present is an unexpected consequence of our attempt to answer the other questions, so we shall start with them.

Relaxation, Optimization, and Weak Constraints

One way of ensuring that a relaxation search is computing something sensible (and will eventually settle down) is to show that it is solving an optimization problem by progressively reducing the value of a cost function. Each possible state of activity of the network has an associated cost, and the rule used for updating activity levels is chosen so that this cost keeps falling. The cost function must be chosen so that low-cost states represent good solutions to problems in the domain.

Many optimization problems can be cast in a framework known as linear programming. There are some variables which take on real values and there are linear equality and inequality constraints between variables. Each combination of values for the variables has an associated cost which is the sum over all the variables of the current value times a cost-coefficient. The aim is to find a combination of values that satisfies all the constraints and minimizes the cost function. If the variables are further constrained to take on only the values 1 or 0 the problem is called zero-one programming. Hinton (1977) has shown that certain zero-one programming problems can be implemented as relaxation searches in parallel networks. This allows networks to find

good solutions to problems in which there are discrete hypotheses that are true or false. Even though the allowable solutions all assign values of 1 or 0 to the hypotheses, the relaxation process works by passing through intermediate states in which hypothesis units have real-valued activity levels lying between 1 and 0. Each constraint is enforced by a feedback loop that measures the amount by which the current values violate the constraint and tries to alter the values of the variables to reduce this violation.

Linear programming and its variants make a sharp distinction between constraints (which *must* be satisfied) and costs. A solution which achieves a very low cost by violating one or two of the constraints is simply not allowed. In many domains, the distinction between constraints and costs is not so clear-cut. In vision, for example, it is usually helpful to use the constraint that neighboring pieces of surface are at similar depths because surfaces are mostly continuous and are rarely parallel to the line of sight. But this is not an absolute constraint. It doesn't apply at the edge of an object. So a visual system needs to be able to generate interpretations that violate this constraint if it can satisfy many other constraints by doing so. Constraints like these have been called "weak" constraints (Blake, 1983) and it is possible to formulate optimization problems in which all the constraints are weak and there is no distinction between constraints and costs. The optimal solution is then the one which minimizes the total constraint violation where different constraints are given different strengths depending on how reliable they are. Another way of saying this is that all the constraints have associated plausibilities, and the most plausible solution is the one which fits these plausible constraints as well as possible.

Some relaxation schemes dispense with separate feedback loops for the constraints and implement weak constraints directly in the excitatory and inhibitory interactions between units. We would like these networks to settle into states in which a few units are fully active and the rest are inactive. Such states constitute clean "digital" interpretations. To prevent the network from hedging its bets by settling into a state where many units are slightly active, it is usually necessary to use a strongly nonlinear decision rule, and this also speeds convergence. However, the strong nonlinearities that are needed to force the network to make a decision also cause it to converge on different states on different occasions: Even with the same external inputs, the final state depends on the initial state of the net. This has led many people (Hopfield, 1982; Rosenfeld, Hummel, & Zucker, 1976) to assume that the particular problem to be solved should be encoded by the initial state of the network rather than by sustained external input to some of its units.

Hummel and Zucker (1983) and Hopfield (1982) have shown that some relaxation schemes have an associated "potential" or cost function and that the states to which the network converges are local minima of this function. This means that the networks are performing optimization of a well-defined function. Unfortunately, there is no guarantee that the network will find the best minimum. One possibility is to redefine the problem as finding the local minimum which is closest to the initial state. This is useful if the minima are used to represent "items" in a memory, and the initial states are queries to memory which may contain missing or erroneous information. The network simply finds the minimum that best fits the query. This idea was used by Hopfield (1982) who introduced an interesting kind of network in which the units were always in one of two states.[1] Hopfield showed that if the units are symmetrically connected (i.e., the weight from unit i to unit j exactly equals the weight from unit j to unit i) and if they are updated one at a time, each update reduces (or at worst does not increase) the value of a cost function which he called "energy" because of the analogy with physical systems. Consequently, repeated iterations are guaranteed to find an energy minimum. The global energy of the system is defined as

$$E = -\sum_{i<j} w_{ij} s_i s_j + \sum_i \theta_i s_i \tag{1}$$

where w_{ij} is the strength of connection (synaptic weight) from the jth to the ith unit, s_i is the state of the ith unit (0 or 1), and θ_i is a threshold.

The updating rule is to switch each unit into whichever of its two states yields the lower total energy given the current states of the other units. Because the connections are symmetrical, the difference between the energy of the whole system with the kth hypothesis false and its energy with the kth hypothesis true can be determined locally by the kth unit, and is just

$$\Delta E_k = \sum_i w_{ki} s_i - \theta_k. \tag{2}$$

Therefore, the rule for minimizing the energy contributed by a unit is to adopt the true state if its total input from the other units exceeds its threshold. This is the familiar rule for binary threshold units.

---

[1] Hopfield used the states 1 and −1 because his model was derived from physical systems called spin glasses in which spins are either "up" or "down." Provided the units have thresholds, models that use 1 and −1 can be translated into models that use 1 and 0 and have different thresholds.

Using Probabilistic Decisions to Escape From Local Minima

At about the same time that Hopfield showed how parallel networks of this kind could be used to access memories that were stored as local minima, Kirkpatrick, working at IBM, introduced an interesting new search technique for solving hard optimization problems on conventional computers.

One standard technique is to use gradient descent: The values of the variables in the problem are modified in whatever direction reduces the cost function (energy). For hard problems, gradient descent gets stuck at *local* minima that are not globally optimal. This is an inevitable consequence of only allowing downhill moves. If jumps to higher energy states occasionally occur, it is possible to break out of local minima, but it is not obvious how the system will then behave and it is far from clear when uphill steps should be allowed.

Kirkpatrick, Gelatt, and Vecchi (1983) used another physical analogy to guide the use of occasional uphill steps. To find a very low energy state of a metal, the best strategy is to melt it and then to slowly reduce its temperature. This process is called annealing, and so they named their search method "simulated annealing." Chapter 6 contains a discussion of why annealing works. We give a simple intuitive account here.

One way of seeing why thermal noise is helpful is to consider the energy landscape shown in Figure 1. Let us suppose that a ball-bearing starts at a randomly chosen point on the landscape. If it always goes downhill (and has no inertia), it will have an even chance of ending up at A or B because both minima have the same width and so the initial

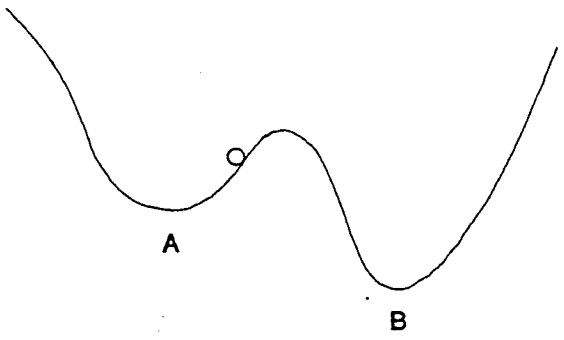

FIGURE 1. A simple energy landscape containing two local minima separated by an energy barrier. Shaking can be used to allow the state of the network (represented here by a ball-bearing) to escape from local minima.

random point is equally likely to lie in either minimum. If we shake the whole system, we are more likely to shake the ball-bearing from A to B than vice versa because the energy barrier is lower from the A side. If the shaking is gentle, a transition from A to B will be many times as probable as a transition from B to A, but both transitions will be very rare. So although gentle shaking will ultimately lead to a very high probability of being in B rather than A, it will take a very long time before this happens. On the other hand, if the shaking is violent, the ball-bearing will cross the barrier frequently and so the ultimate probability ratio will be approached rapidly, but this ratio will not be very good: With violent shaking it is almost as easy to cross the barrier in the wrong direction (from B to A) as in the right direction. A good compromise is to start by shaking hard and gradually shake more and more gently. This ensures that at some stage the noise level passes through the best possible compromise between the absolute probability of a transition and the ratio of the probabilities of good and bad transitions. It also means that at the end, the ball-bearing stays right at the bottom of the chosen minimum.

This view of why annealing helps is not the whole story. Figure 1 is misleading because all the states have been laid out in one dimension. Complex systems have high-dimensional state spaces, and so the barrier between two low-lying states is typically massively degenerate: The number of ways of getting from one low-lying state to another is an exponential function of the height of the barrier one is willing to cross. This means that a rise in the level of thermal noise opens up an enormous variety of paths for escaping from a local minimum and even though each path by itself is unlikely, it is highly probable that the system will cross the barrier. We conjecture that simulated annealing will only work well in domains where the energy barriers are highly degenerate.

Applying Simulated Annealing to Hopfield Nets

There is a simple modification of Hopfield's updating rule that allows parallel networks to implement simulated annealing. If the energy gap between the 1 and 0 states of the kth unit is ΔE_k then, regardless of the previous state set, $s_k = 1$ with probability

$$p_k = \frac{1}{\left(1+e^{-\Delta E_k / T}\right)} \tag{3}$$

where T is a parameter which acts like the temperature of a physical system. This local decision rule ensures that in thermal equilibrium the relative probability of two global states is determined solely by their energy difference, and follows a Boltzmann distribution:

$$\frac{P_\alpha}{P_\beta} = e^{-(E_\alpha - E_\beta)/T} \tag{4}$$

where P_α is the probability of being in the αth global state, and E_α is the energy of that state.

At low temperatures there is a strong bias in favor of states with low energy, but the time required to reach equilibrium may be long. At higher temperatures the bias is not so favorable, but equilibrium is reached faster. The fastest way to reach equilibrium at a given temperature is generally to use simulated annealing: Start with a higher temperature and gradually reduce it.

The idea of implementing constraints as interactions between stochastic processing elements was proposed by Moussouris (1974) who discussed the identity between Boltzmann distributions and Markov random fields. The idea of using simulated annealing to find low energy states in parallel networks has been investigated independently by several different groups. S. Geman and D. Geman (1984) established limits on the allowable speed of the annealing schedule and showed that simulated annealing can be very effective for removing noise from images. Hinton and Sejnowski (1983b) showed how the use of binary stochastic elements could solve some problems that plague other relaxation techniques, in particular the problem of learning the weights. Smolensky (1983) has been investigating a similar scheme which he calls "harmony theory." This scheme is discussed in detail in Chapter 6. Smolensky's harmony is equivalent to our energy (with a sign reversal).

Pattern Completion

One way of using a parallel network is to treat it as a pattern completion device. A subset of the units are "clamped" into their on or off states and the weights in the network then complete the pattern by determining the states of the remaining units. There are strong limitations on the sets of binary vectors that can be learned if the network has one unit for each component of the vector. These limits can be transcended by using extra units whose states do not correspond to components in the vectors to be learned. The weights of connections to these extra units can be used to represent complex interactions that

cannot be expressed as pairwise correlations between the components of the vectors. We call these extra units *hidden units* (by analogy with hidden Markov processes) and we call the units that are used to specify the patterns to be learned the *visible units*. The visible units are the interface between the network and the environment that specifies vectors for it to learn or asks it to complete a partial vector. The hidden units are where the network can build its own internal representations.

Sometimes, we would like to be able to complete a pattern from any sufficiently large part of it without knowing in advance which part will be given and which part must be completed. Other times we know in advance which parts will be given as input and which parts will have to be completed as output. So there are two different completion paradigms. In the first, any of the visible units might be part of the required output. In the second, there is a distinguished subset of the visible units, called the input units, which are always clamped by the environment, so the network never needs to determine the states of these units.

EASY AND HARD LEARNING

Consider a network which is allowed to run freely, using the probabilistic decision rule in Equation 3, without having any of its units clamped by the environment. When the network reaches thermal equilibrium, the probability of finding it in any particular global state depends only on the energy of that state (Equation 4). We can therefore control the probabilities of global states by controlling their energies. If each weight only contributed to the energy of a single global state, this would be straightforward, but changing a weight will actually change the energies of many different states so it is not immediately obvious how a weight-change will affect the probability of a particular global state. Fortunately, if we run the network until it reaches thermal equilibrium, Equations 3 and 4 allow us to derive the way in which the probability of each global state changes as a weight is changed:

$$\frac{\partial \ln P_\alpha^-}{\partial w_{ij}} = \frac{1}{T}\left(s_i^\alpha s_j^\alpha - \sum_\beta P_\beta^- s_i^\beta s_j^\beta\right) \tag{5}$$

where s_i^α is the binary state of the ith unit in the αth global state and P_α^- is the probability, at thermal equilibrium, of global state α of the network when none of the visible units are clamped (the lack of clamping is denoted by the superscript $^-$). Equation 5 shows that clamping is denoted by the superscript $^-$). Equation 5 shows that the effect of a

weight on the log probability of a global state can be computed from purely local information because it only involves the behavior of the two units that the weight connects (the second term is just the probability of finding the ith and jth units on together). This makes it easy to manipulate the probabilities of global states provided the desired probabilities are known (see Hinton & Sejnowski, 1983a, for details).

Unfortunately, it is normally unreasonable to expect the environment or a teacher to specify the required probabilities of entire global states of the network. The task that the network must perform is defined in terms of the states of the visible units, and so the environment or teacher only has direct access to the states of these units. The difficult learning problem is to decide how to use the hidden units to help achieve the required behavior of the visible units. A learning rule which assumes that the network is instructed from outside on how to use *all* of its units is of limited interest because it evades the main problem which is to discover appropriate representations for a given task among the hidden units.

In statistical terms, there are many kinds of statistical structure implicit in a large ensemble of environmental vectors. The separate probability of each visible unit being active is the first-order structure and can be captured by the thresholds of the visible units. The $v^2/2$ pairwise correlations between the v visible units constitute the second-order structure and this can be captured by the weights between pairs of units.[2] All structure higher than second-order cannot be captured by pairwise weights *between the visible units*. A simple example may help to clarify this crucial point.

Suppose that the ensemble consists of the vectors: (1 1 0), (1 0 1), (0 1 1), and (0 0 0), each with a probability of 0.25. There is clearly some structure here because four of the eight possible 3-bit vectors never occur. However, the structure is entirely third-order. The first-order probabilities are all 0.5, and the second-order correlations are all 0, so if we consider only these statistics, this ensemble is indistinguishable from the ensemble in which all eight vectors occur equiprobably.

The Widrow-Hoff rule or perceptron convergence procedure (Rosenblatt, 1962) is a learning rule which is designed to capture second-order structure and it therefore fails miserably on the example just given. If the first two bits are treated as an input and the last bit is treated as the required output, the ensemble corresponds to the function "exclusive-or" which is one of the examples used by Minsky and Papert (1969) to show the strong limitations of one-layer perceptrons. The Widrow-Hoff

---

[2] Factor analysis confines itself to capturing as much of the second-order structure as possible in a few underlying "factors." It ignores all higher order structure which is where much of the interesting information lies for all but the most simple ensembles of vectors.

rule can do easy learning, but it cannot do the kind of hard learning that involves deciding how to use extra units whose behavior is not directly specified by the task.

It is tempting to think that networks with pairwise connections can never capture higher than second-order statistics. There is one sense in which this is true and another in which it is false. By introducing extra units which are not part of the definition of the original ensemble, it is possible to express the third-order structure of the original ensemble in the second-order structure of the larger set of units. In the example given above, we can add a fourth component to get the ensemble {(1101), (1010), (0110), (0000)}. It is now possible to use the thresholds and weights between all four units to express the third-order structure in the first three components. A more familiar way of saying this is that we introduce an extra "feature detector" which in this example detects the case when the first two units are both on. We can then make each of the first two units excite the third unit, and use strong inhibition from the feature detector to overrule this excitation when *both* of the first two units are on. The difficult problem in introducing the extra unit was deciding when it should be on and when it should be off—deciding what feature it should detect.[3]

One way of thinking about the higher order structure of an ensemble of environmental vectors is that it implicitly specifies good sets of underlying features that can be used to model the structure of the environment. In common-sense terms, the weights in the network should be chosen so that the hidden units represent significant underlying features that bear strong, regular relationships to each other and to the states of the visible units. The hard learning problem is to figure out what these features are, i.e., to find a set of weights which turn the hidden units into useful feature detectors that explicitly represent properties of the environment which are only implicitly present as higher order statistics in the ensemble of environmental vectors.

Maximum Likelihood Models

Another view of learning is that the weights in the network constitute a generative model of the environment—we would like to find a set of weights so that when the network is running freely, the patterns of activity that occur over the visible units are the same as they would be if the environment was clamping them. The number of units in the

[3] In this example there are six different ways of using the extra unit to solve the task.

network and their interconnectivity define a space of possible models of the environment, and any particular set of weights defines a particular model within this space. The learning problem is to find a combination of weights that gives a good model given the limitations imposed by the architecture of the network and the way it runs.

More formally, we would like a way of finding the combination of weights that is most likely to have produced the observed ensemble of environmental vectors. This is called a *maximum likelihood* model and there is a large literature within statistics on maximum likelihood estimation. The learning procedure we describe actually has a close relationship to a method called Expectation and Maximization (EM) (Dempster, Laird, & Rubin, 1976). EM is used by statisticians for estimating missing parameters. It represents probability distributions by using parameters like our weights that are exponentially related to probabilities, rather than using probabilities themselves. The EM algorithm is closely related to an earlier algorithm invented by Baum that manipulates probabilities directly. Baum's algorithm has been used successfully for speech recognition (Bahl, Jelinek, & Mercer, 1983). It estimates the parameters of a hidden Markov chain—a transition network which has a fixed structure but variable probabilities on the arcs and variable probabilities of emitting a particular output symbol as it arrives at each internal node. Given an ensemble of strings of symbols and a fixed-topology transition network, the algorithm finds the combination of transition probabilities and output probabilities that is most likely to have produced these strings (actually it only finds a local maximum).

Maximum likelihood methods work by adjusting the parameters to increase the probability that the generative model will produce the observed data. Baum's algorithm and EM are able to estimate new values for the probabilities (or weights) that are guaranteed to be better than the previous values. Our algorithm simply estimates the gradient of the log likelihood with respect to a weight, and so the magnitude of the weight change must be decided using additional criteria. Our algorithm, however, has the advantage that it is easy to implement in a parallel network of neuron-like units.

The idea of a stochastic generative model is attractive because it provides a clean quantitative way of comparing alternative representational schemes. The problem of saying which of two representational schemes is best appears to be intractable. Many sensible rules of thumb are available, but these are generally pulled out of thin air and justified by commonsense and practical experience. They lack a firm mathematical foundation. If we confine ourselves to a space of allowable stochastic models, we can then get a simple Bayesian measure of the quality of a representational scheme: How likely is the observed ensemble of

environmental vectors given the representational scheme? In our networks, representations are patterns of activity in the units, and the representational scheme therefore corresponds to the set of weights that determines when those patterns are active.

THE BOLTZMANN MACHINE LEARNING ALGORITHM

If we make certain assumptions it is possible to derive a measure of how effectively the weights in the network are being used for modeling the structure of the environment, and it is also possible to show how the weights should be changed to progressively improve this measure. We assume that the environment clamps a particular vector over the visible units and it keeps it there long enough for the network to reach thermal equilibrium with this vector as a boundary condition (i.e., to "interpret" it). We also assume (unrealistically) that the there is no structure in the sequential order of the environmentally clamped vectors. This means that the complete structure of the ensemble of environmental vectors can be specified by giving the probability, $P^+(V_\alpha)$, of each of the 2^v vectors over the v visible units. Notice that the $P^+(V_\alpha)$ do not depend on the weights in the network because the environment clamps the visible units.

A particular set of weights can be said to constitute a perfect model of the structure of the environment if it leads to exactly the same probability distribution of visible vectors when the network is running freely *with no units being clamped by the environment.* Because of the stochastic behavior of the units, the network will wander through a variety of states even with no environmental input and it will therefore generate a probability distribution, $P^-(V_\alpha)$, over all 2^v visible vectors. This distribution can be compared with the environmental distribution, $P^+(V_\alpha)$. In general, it will not be possible to exactly match the 2^v environmental probabilities using the weights among the v visible and h hidden units because there are at most $(v+h-1)(v+h)/2$ symmetrical weights and $(v+h)$ thresholds. However, it may be possible to do very well if the environment contains regularities that can be expressed in the weights. An information theoretic measure (Kullback, 1959) of the distance between the environmental and free-running probability distributions is given by:

$$G = \sum_\alpha P^+(V_\alpha) \ln \frac{P^+(V_\alpha)}{P^-(V_\alpha)} \tag{6}$$

where $P^+(V_\alpha)$ is the probability of the αth state of the visible units in

phase$^+$ when their states are determined by the environment, and $P^-(V_\alpha)$ is the corresponding probability in *phase*$^-$ when the network is running freely with no environmental input.

G is never negative and is only zero if the distributions are identical. G is actually the distance in bits *from* the free running distribution *to* the environmental distribution.[4] It is sometimes called the asymmetric divergence or information gain. The measure is not symmetric with respect to the two distributions. This seems odd but is actually very reasonable. When trying to approximate a probability distribution, it is more important to get the probabilities correct for events that happen frequently than for rare events. So the match between the actual and predicted probabilities of an event should be weighted by the actual probability as in Equation 6.

It is possible to improve the network's model of the structure of its environment by changing the weights so as to reduce G.[5] To perform gradient descent in G, we need to know how G will change when a weight is changed. But changing a single weight changes the energies of one quarter of all the global states of the network, and it changes the probabilities of all the states in ways that depend on *all* the other weights in the network. Consider, for example, the very simple network shown in Figure 2. If we want the two units at the ends of the chain to be either both on or both off, how should we change the weight $w_{3,4}$? It clearly depends on the signs of remote weights like $w_{1,2}$ because we need to have an even number of inhibitory weights in the chain.[6] So the partial derivative of G with respect to one weight depends on all the other weights and minimizing G appears to be a

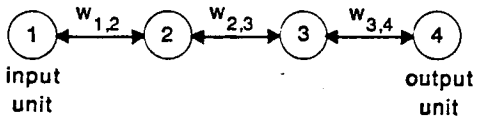

FIGURE 2. A very simple network with one input unit, one output unit, and two hidden units. The task is to make the output unit adopt the same state as the input unit. The difficulty is that the correct value for weight $w_{3,4}$ depends on remote information like the value of weight $w_{1,2}$.

[4] If we use base 2 logarithms.

[5] Peter Brown (personal communication) has pointed out that minimizing G is equivalent to maximizing the log of the likelihood of generating the environmental probability distribution when the network is running freely at equilibrium.

[6] The thresholds must also be adjusted appropriately.

difficult computational problem that requires nonlocal information.

Fortunately, all the information that is required about the other weights in order to change w_{ij} appropriately shows up in the behavior of the ith and jth units at thermal equilibrium. In addition to performing a search for low energy states of the network, the process of reaching thermal equilibrium ensures that the joint activity of any two units contains all the information required for changing the weight between them in order to give the network a better model of its environment. The joint activity implicitly encodes information about all the other weights in the network. The Appendix shows that

$$\frac{\partial G}{\partial w_{ij}} = -\frac{1}{T}[p_{ij}^{+} - p_{ij}^{-}] \tag{7}$$

where p_{ij}^{+} is the probability, averaged over all environmental inputs and measured at equilibrium, that the ith and jth units are both on when the network is being driven by the environment, and p_{ij}^{-} is the corresponding probability when the network is free running. One surprising feature of Equation 7 is that it does not matter whether the weight is between two visible units, two hidden units, or one of each. The same rule applies for the gradient of G.

Unlearning

Crick and Mitchison (1983) have suggested that a form of reverse learning might occur during REM sleep in mammals. Their proposal was based on the assumption that parasitic modes develop in large networks that hinder the distributed storage and retrieval of information. The mechanism that Crick and Mitchison propose is based on

> More or less random stimulation of the forebrain by the brain stem that will tend to stimulate the inappropriate modes of brain activity . . . and especially those which are too prone to be set off by random noise rather than by highly structured specific signals. (p. 112)

During this state of random excitation and free running they postulate that changes occur at synapses to decrease the probability of the spurious states.

A simulation of reverse learning was performed by Hopfield, Feinstein, and Palmer (1983) who independently had been studying ways to improve the associative storage capacity of simple networks of binary processors (Hopfield, 1982). In their algorithm an input is presented to the network as an initial condition, and the system evolves by falling

into a nearby local energy minimum. However, not all local energy minima represent stored information. In creating the desired minima, they accidentally create other spurious minima, and to eliminate these they use "unlearning": The learning procedure is applied with reverse sign to the states found after starting from random initial conditions. Following this procedure, the performance of the system in accessing stored states was found to be improved.

There is an interesting relationship between the reverse learning proposed by Crick and Mitchison and Hopfield et al. and the form of the learning algorithm which we derived by considering how to minimize an information theory measure of the discrepancy between the environmental structure and the network's internal model (Hinton & Sejnowski, 1983b). The two phases of our learning algorithm resemble the learning and unlearning procedures: Positive Hebbian learning occurs in *phase*$^+$ during which information in the environment is captured by the weights; during *phase*$^-$ the system randomly samples states according to their Boltzmann distribution and Hebbian learning occurs with a negative coefficient.

However, these two phases need not be implemented in the manner suggested by Crick and Mitchison. For example, during *phase*$^-$ the average co-occurrences could be computed without making any changes to the weights. These averages could then be used as a baseline for making changes during *phase*$^+$; that is, the co-occurrences during *phase*$^+$ could be computed and the baseline subtracted before each permanent weight change. Thus, an alternative but equivalent proposal for the function of dream sleep is to recalibrate the baseline for plasticity—the break-even point which determines whether a synaptic weight is incremented or decremented. This would be safer than making permanent weight decrements to synaptic weights during sleep and solves the problem of deciding how much "unlearning" to do.

Our learning algorithm refines Crick and Mitchison's interpretation of why two phases are needed. Consider a hidden unit deep within the network: How should its connections with other units be changed to best capture regularity present in the environment? If it does not receive direct input from the environment, the hidden unit has no way to determine whether the information it receives from neighboring units is ultimately caused by structure in the environment or is entirely a result of the other weights. This can lead to a "folie a deux" where two parts of the network each construct a model of the other and ignore the external environment. The contribution of internal and external sources can be separated by comparing the co-occurrences in *phase*$^+$ with similar information that is collected in the absence of environmental input. *phase*$^-$ thus acts as a control condition. Because of the special properties of equilibrium it is possible to subtract off this

purely internal contribution and use the difference to update the weights. Thus, the role of the two phases is to make the system maximally responsive to regularities present in the environment and to prevent the system from using its capacity to model internally-generated regularities.

Ways in Which the Learning Algorithm Can Fail

The ability to discover the partial derivative of G by observing p_{ij}^{+} and p_{ij}^{-} does not completely determine the learning algorithm. It is still necessary to decide how much to change each weight, how long to collect co-occurrence statistics before changing the weight, how many weights to change at a time, and what temperature schedule to use during the annealing searches. For very simple networks in very simple environments, it is possible to discover reasonable values for these parameters by trial and error. For more complex and interesting cases, serious difficulties arise because it is very easy to violate the assumptions on which the mathematical results are based (Derthick, 1984).

The first difficulty is that there is nothing to prevent the learning algorithm from generating very large weights which create such high energy barriers that the network cannot reach equilibrium in the allotted time. Once this happens, the statistics that are collected will not be the equilibrium statistics required for Equation 7 to hold and so all bets are off. We have observed this happening for a number of different networks. They start off learning quite well and then the weights become too large and the network "goes sour"—its performance deteriorates dramatically.

One way to ensure that the network gets close to equilibrium is to keep the weights small. Pearlmutter (personal communication) has shown that the learning works much better if, in addition to the weight changes caused by the learning, every weight continually decays towards a value of zero, with the speed of the decay being proportional to the absolute magnitude of the weight. This keeps the weights small and eventually leads to a relatively stable situation in which the decay rate of a weight is balanced by the partial derivative of G with respect to the weight. This has the satisfactory property that the absolute magnitude of a weight shows how important it is for modeling the environmental structure.

The use of weight-decay has several other consequences which are not so desirable. Because the weights stay small, the network cannot construct very deep minima in the energy landscape and so it cannot make the probability ratios for similar global states be very different.

This means that it is bound to give a significant number of errors in modeling environments where very similar vectors have very different probabilities. Better *performance* can be achieved by annealing the network to a lower final temperature (which is equivalent to making all the weights larger), but this will make the *learning* worse for two separate reasons. First, with less errors there is less to drive the learning because it relies on the difference between the $phase^+$ and $phase^-$ statistics. Second, it will be harder to reach thermal equilibrium at this lower temperature and so the co-occurrence statistics will be unreliable. One way of getting good statistics to drive the learning and also getting very few overt errors is to measure the co-occurrence statistics at a temperature higher than the final one.

Another way of ensuring that the network approaches equilibrium is to eliminate deep, narrow minima that are often not found by the annealing process. Derthick (1984) has shown that this can be done using a longer gentler annealing schedule in $phase^-$. This means that the network is more likely to occupy the hard-to-find minima in $phase^-$ than in $phase^+$., and so these minima will get filled in because the learning rule raises the energies of states that are occupied more in $phase^-$ than in $phase^+$.

AN EXAMPLE OF HARD LEARNING

A simple example which can only be solved by capturing the higher order statistical structure in the ensemble of input vectors is the "shifter" problem. The visible units are divided into three groups. Group V_1 is a one-dimensional array of 8 units, each of which is clamped on or off at random with a probability of 0.3 of being on. Group V_2 also contains 8 units and their states are determined by shifting and copying the states of the units in group V_1. The only shifts allowed are one to the left, one to the right, or no shift. Wrap-around is used so that when there is a right shift, the state of the right-most unit in V_1 determines the state of the left-most unit in V_2. The three possible shifts are chosen at random with equal probabilities. Group V_3 contains three units to represent the three possible shifts, so at any one time one of them is clamped on and the others are clamped off.

The problem is to learn the structure that relates the states of the three groups. One facet of this problem is to "recognize" the shift—i.e., to complete a partial input vector in which the states of V_1 and V_2 are clamped but the units in V_3 are left free. It is fairly easy to see why this problem cannot possibly be solved by just adding together a lot of pairwise interactions between units in V_1, V_2, and V_3. If you know

that a particular unit in V_1 is on, it tells you nothing whatsoever about what the shift is. It is only by finding *combinations* of active units in V_1 and V_2 that it is possible to predict the shift, so the information required is of at least third-order. This means that extra hidden units are required to perform the task.

The obvious way to recognize the shift is to have extra units which detect informative features such as an active unit in V_1 and an active unit one place to the right in V_2 and then support the unit V_3 that represents a right shift. The empirical question is whether the learning algorithm is capable of turning some hidden units into feature detectors of this kind, and whether it will generate a set of detectors that work well together rather than duplicating the same detector. The set of weights that minimizes G defines the *optimal* set of detectors but it is not at all obvious what these detectors are, nor is it obvious that the learning algorithm is capable of finding a good set.

Figure 3 shows the result of running a version of the Boltzmann machine learning procedure. Of the 24 hidden units, 5 seem to be doing very little but the remainder are sensible looking detectors and most of them have become spatially localized. One type of detector which occurs several times consists of two large negative weights, one above the other, flanked by smaller excitatory weights on each side. This is a more discriminating detector of no-shift than simply having two positive weights, one above the other. It interesting to note that the various instances of this feature type all have different locations in V_1 and V_2, even though the hidden units are not connected to each other. The pressure for the feature detectors to be different from each other comes from the gradient of G, rather than from the kind of lateral inhibition among the feature detectors that is used in "competitive learning" paradigms (Fukushima, 1980; Rumelhart & Zipser, 1985).

The Training Procedure

The training procedure alternated between two phases. In *phase*$^+$, all the units in V_1, V_2, and V_3 were clamped into states representing a pair of 8-bit vectors and their relative shift. The hidden units were then allowed to change their states until the system approached thermal equilibrium at a temperature of 10. The annealing schedule is described below. After annealing, the network was assumed to be close to thermal equilibrium and it was then run for a further 10 iterations during which time the frequency with which each pair of connected units were both on was measured. This was repeated 20 times with

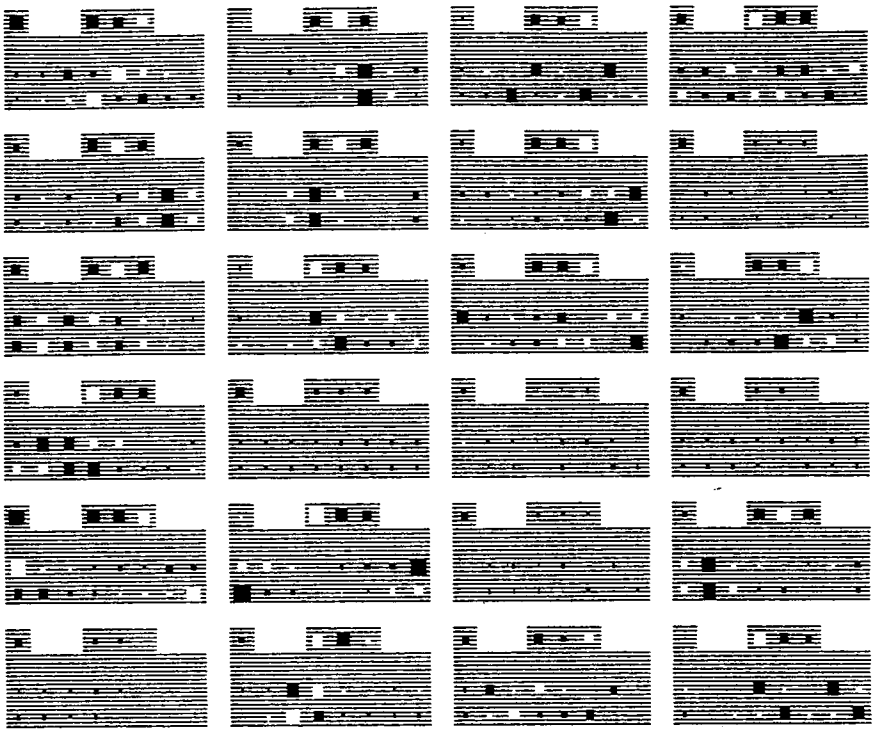

FIGURE 3. The weights of the 24 hidden units in the shifter network. Each large region corresponds to a unit. Within this region the black rectangles represent negative weights and the white rectangles represent positive ones. The size of a rectangle represents the magnitude of the weight. The two rows of weights at the bottom of each unit are its connections to the two groups of input units, V_1 and V_2. These weights therefore represent the "receptive field" of the hidden unit. The three weights in the middle of the top row of each unit are its connections to the three output units that represent shift-left, no-shift, and shift-right. The solitary weight at the top left of each unit is its threshold. Each hidden unit is directly connected to all 16 input units and all 3 output units. In this example, the hidden units are not connected to each other. The top-left unit has weights that are easy to understand: Its optimal stimulus is activity in the fourth unit of V_1 and the fifth unit of V_2, and it votes for shift-right. It has negative weights to make it less likely to come on when there is an alternative explanation for why its two favorite input units are active.

different clamped vectors and the co-occurrence statistics were averaged over all 20 runs to yield an estimate, for each connection, of p_{ij}^{+} in Equation 7. In *phase*$^{-}$, none of the units were clamped and the network was annealed in the same way. The network was then run for a further 10 iterations and the co-occurrence statistics were collected for all connected pairs of units. This was repeated 20 times and the co-occurrence statistics were averaged to yield an estimate of p_{ij}^{-}.

The entire set of 40 annealings that were used to estimate p_{ij}^{+} and p_{ij}^{-} was called a sweep. After each sweep, every weight was incremented by $5(p_{ij}^{+} - p_{ij}^{-})$. In addition, every weight had its absolute magnitude decreased by 0.0005 times its absolute magnitude. This weight decay prevented the weights from becoming too large and it also helped to resuscitate hidden units which had predominantly negative or predominantly positive weights. Such units spend all their time in the same state and therefore convey no information. The $phase^{+}$ and $phase^{-}$ statistics are identical for these units, and so the weight decay gradually erodes their weights until they come back to life (units with all zero weights come on half the time).

The Annealing Schedule

The annealing schedule spent the following number of iterations at the following temperatures: 2 at 40, 2 at 35, 2 at 30, 2 at 25, 2 at 20, 2 at 15, 2 at 12, 2 at 10. One iteration is defined as the number of random probes required so that each unit is probed one time on average. When it is probed, a unit uses its energy gap to decide which of its two states to adopt using the stochastic decision rule in Equation 3. Since each unit gets to see the most recent states of all the other units, an iteration cannot be regarded as a single parallel step. An truly parallel asynchronous system must tolerate time delays. Units must decide on their new states without being aware of very recent changes in the states of other units. It can be shown (Sejnowski, Hinton, Kienker, & Schumacher, 1985) that first-order time delays act like added temperature and can therefore be tolerated by networks of this kind.

The Performance of the Shifter Network

The shifter network is encouraging because it is a clear example of the kind of learning of higher order structure that was beyond the capability of perceptrons, but it also illustrates several weaknesses in the current approach.

- The learning was very slow. It required 9000 learning sweeps, each of which involved reaching equilibrium 20 times in $phase^{+}$ with vectors clamped on V_1, V_2, and V_3, and 20 times in $phase^{-}$ with no units clamped. Even for low-level perceptual learning, this seems excessively slow.