COGNITIVA 85

Paris, 4-7 Juin 1985

LEARNING IN BOLTZMANN MACHINES

APPRENTISSAGE DANS LES MACHINES DE BOLTZMANN

G. E. Hinton and T. J. Sejnowski

Computer Science Department
Carnegie-Mellon University
Pittsburgh PA 15213, USA

Biophysics Department
The Johns Hopkins University
Baltimore MD 21218, USA

RESUME

Un problème central de la perception est la recherche combinatoire. Etant donné un grand nombre d'interprétations possibles d'une partie d'une image, et un ensemble de contraintes, des valeurs de vérité doivent être assignées aux hypothèses afin de minimiser le nombre de contraintes non vérifiées. Ceci peut être réalisé en plaçant un réseau de processeurs élémentaires dans un état stable. Chaque élément représente une hypothèse, et les interactions entre éléments représentent les contraintes.

Il s'est avère difficile d'analyser ce type de calcul coöpératif car les éléments doivent être non-linéaires et couples. Si, toutefois, les éléments ne peuvent prendre que des états discrets déterminés par une founction *stochastique* des entrées, des méthodes statistiques fournissent une description du comportement de l'ensemble du système. Dans ce cas, il existe une relation très simple entre l'état *global* et les valeurs des interactions *local* entre processeurs élémentaires. Cette relation permet à un système coöpératif d'apprendre les contraintes implicites à un domaine en examinant des exemples de ce domaine. La procédure d'apprentissage crée des représentations internes qui reflètent la structure sous-jacente de l'ensemble des exemples presentes au réseau.

SUMMARY

A central computational problem in perception is search. Given a large set of candidate hypotheses about how to interpret parts or aspects of an image, and a set of plausible constraints between them, truth-values must be assigned to the hypotheses so as to minimize the total violation of the plausible constraints. This can be done by allowing a network of computing elements to settle into a stable state. Each element represents a hypothesis, and the interactions between the elements implement the constraints.

Until recently it has been very difficult to analyze this kind of cooperative computation, because the elements must be non-linear and cross-coupled. If, however, the elements adopt discrete states according to a particular *stochastic* function of their inputs, it is possible to use statistical mechanics to describe the behavior of the whole system. This leads to a very simple relationship between the probability of finding the network in a particular *global* state and the strengths of the *local* interactions between the individual computing elements. This relationship allows a cooperative system to learn the implicit constraints in a domain simply by being shown examples from that domain. The learning procedure creates internal representations which express the underlying structure of the ensemble of examples that the network is shown.

Learning in Boltzmann Machines

G. E. Hinton and T. J. Sejnowski

Parallel networks can perform, iterative searches for good solutions to problems. 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 (Ballard, Hinton and Sejnowski, 1983). 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.

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:

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

2. 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.

3. 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.

4. 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 at least 100ms which is about the time allowed for the whole iterative process to settle down.

5. How are the weights that encode the knowledge acquired? For 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 paper is mainly concerned with the last of these questions, but the learning procedure we shall present was an unexpected consequence of our attempts to answer the other questions, so we shall start with them.

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 simply 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.

Some relaxation schemes dispense with separate feedback loops for the constraints and implement weak constraints directly in the 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 non-linear decision rule, and this also speeds convergence. However, the strong non-linearities 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

Learning in Boltzmann Machines

G. E. Hinton and T. J. Sejnowski

the net. This has led many people (Rosenfeld, Hummel and Zucker, 1976; Hopfield, 1982) 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 task as the problem of 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. 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. 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 upclate reduces (or at worst does not increase) the value of a cost function which is analogous to the energy of a physical system. Consequ3ntly, repeated iterations are guaranteed to find an energy minimum. The global energy of the system is defined as

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

where w_{ij} is the strength of connection (synaptic weight) from the j^{th} to the i^{th} unit, s_i is the state of the i^{th} 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 k^{th} hypothesis false and its energy with the k^{th} hypothesis true can be determined locally by the k^{th} unit, and is just

$$\Delta E_k = \sum_i w_{ki} s_i \; - \theta_k \qquad (2)$$

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.

Using probabilistic decisions to escape from local minima

At about the same time as Hopfield showed how parallel networks of this kind could be used to access memories that were stored as local minima, Scott 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". 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 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 more probable than 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 A rather than B, 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.

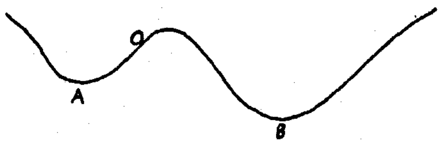

Figure 1: A simple energy landscape

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.

Learning in Boltzmann Machines

G. E. Hinton and T. J. Sejnowski

Simulated annealing in parallel 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 k^{th} unit is ΔE_k then regardless of the previous state set $s_k = 1$ with probability

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

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.

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

where P_α is the probability of being in the α^{th} 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. Geman and 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. Smolensky (1983) has been investigating a similar scheme called "harmony theory".

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 Eq. 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 (Eq 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:

$$\frac{\partial \ln P^-_\alpha}{\partial w_{ij}} = \frac{1}{T}[\, s_i^\alpha s_j^\alpha - \sum_\beta P^-_\beta s_i^\beta s_j^\beta \,] \qquad (5)$$

where s_i^α is the binary state of the i^{th} unit in the α^{th} 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 $^-$). Eq. 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 i^{th} and j^{th} 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 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. 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), (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, 1961) is a learning rule which is designed to capture second-order structure and it therefore fails 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 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 { (1 1 0 1), (1 0 1 0), (0 1 1 0), (0 0 0 0) } and 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 over-rule 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.

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 network and their inter-connectivity 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 EM which stands for "Expectation and Maximization" (Dempster, Laird, and 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, and 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 so as to increase the probability that the generative model would 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 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.

Learning In Boltzmann Machines

G. E. Hinton and T. J. Sejnowski

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:

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

where $P^{+}(V_\alpha)$ is the probability of the α^{th} 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 (if we use base 2 logarithms.) It is possible to improve the network's model of the structure of its environment by changing the weights so as to reduce G. 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. 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, so minimizing G appears to be a difficult computational problem that requires non-local 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 i^{th} and j^{th} 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. Ackley, Hinton and Sejnowski (1985) show that:

$$\frac{\partial G}{\partial w_{ij}} = -\frac{1}{T}\,[p_{ij}^{+} - p_{ij}^{-}] \qquad (7)$$

where p_{ij}^{+} is the probability, averaged over all environmental inputs and measured at equilibrium, that the i^{th} and j^{th} 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 Eq. 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.

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-activation 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. One way to ensure that the network gets close to equilibrium is to keep the weights small. Barak 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.

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 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 2 shows some of the feature detectors that are created by the learning algorithm. 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. The learning algorithm created multiple instances of this feature type, but they focused on different regions 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 (Rumelhart & Zipser, 1985; Fukushima, 1980).

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 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}^-.

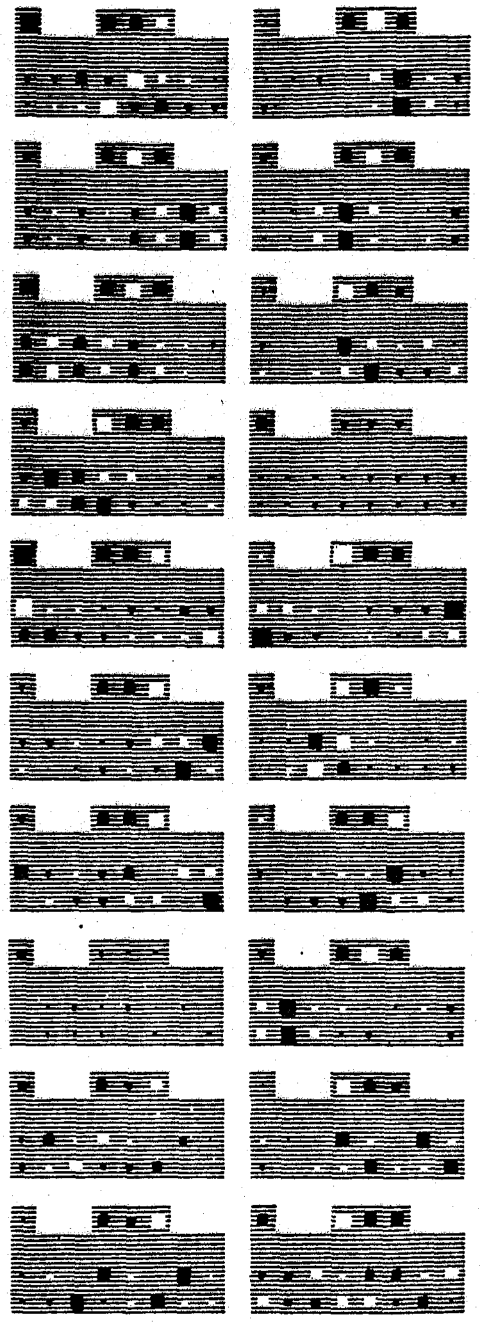

Figure 2: This shows the weights of 20 of the 24 hidden units in the *shifter network*. Each large region corresponds to a unit. Within this region the black rectangles represent negative weights and the white rectangles represent positive ones. The size of a rectangle represents the magnitude of the weight. The two rows of weights at the bottom of each unit are its connections to the two groups of units, V_1 and V_2. The three weights in the middle of the top row of each unit are its connections to the three units of V_3 that represent shift-left, no-shift, and shift-right. The solitary weight at the top right 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.

The entire set of 40 annealings that were used to estimate p_{ij}^+ and p_{ij}^- was called a sweep, and a total of 9000 sweeps were performed. 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 spent the following number of iterations at the following temperatures: 2@40, 2@35, 2@30, 2@25, 2@20, 2@15, 2@12, 2@10. One iteration is defined as the number of random probes required so that each unit is probed once 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 Eq. 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. A 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 and Schumacher, 1985) that to first order time delays act like added temperature and can therefore be tolerated by networks of this kind.

Conclusion

We have described a new kind of relaxation search: Networks of symmetrically connected binary threshold units can escape from local minima by using a *stochastic* decision rule. The search is not guaranteed to produce the optimal solution, but it is guaranteed to produce solutions with a probability that depends on how good they are (provided it is run for long enough to approach thermal equilibrium at a finite temperature).

At thermal equilibrium, the probability that two units are both active encode information about how to change the weight between them. The joint probability must be measured in two different conditions: when the environment is clamping the states of some units, and when the network is free-running. The difference in the joint probabilities in these two phases specifies how to change the weight so as to make the free-running network into a generative model of the environmental structure.

Acknowledgements

This research was supported by grants from the System Development Foundation. We thank David Ackley, Peter Brown, Francis Crick, Mark Derthick, Scott Fahlman, Stuart Geman, Martial Hebert, John Hopfield, Paul Kienker, Jay McClelland, Barak Pearlmutter, David Rumelhart, Tim Shallice, and Paul Smolensky for helpful discussions. An expanded version of this paper will appear in: D. E. Rumelhart & J. L. McClelland (Eds.) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations.* Cambridge, MA: Bradford Books, (in press), 1985.

References

Ackley, D. H., Hinton, G. E., Sejnowski, T. J. A learning algorithm for Boltzmann machines. *Cognitive Science*, 1985, 9, 147-169.

Bahl, L. R., Jelinek, F., & Mercer, R. L. A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1983, PAMI-5, 179-190.

Ballard, D. H., Hinton G. E., Sejnowski, T. J. Parallel visual computation. *Nature*, 1983, 306, 21-26.

Blake, A. The least disturbance principle and weak constraints. *Pattern Recognition Letters*, 1983, 1, 393-399.

Dempster, A.P., Laird, N. M., & Rubin, D. B. Maximum likelihood from incomplete data via the EM algorithm. *Proceedings of the Royal Statistical Society*, 1976, 1-38.

Derthick, M. A. Variations on the Boltzmann Machine learning algorithm. Technical Report CMU-CS-84-120. Carnegie-Mellon University, Pittsburgh PA. Aug 1984.

Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 1980, 36, 193-202.

Geman, S., & Geman D. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1984, PAMI-6, 721-741.

Hinton, G. E. Relaxation and its role in vision. PhD Thesis, University of Edinburgh, 1977.

Hinton, G.E., & Sejnowski, T.J. Analyzing cooperative computation. *Proceedings of the Fifth Annual Conference of the Cognitive Science Society*. Rochester, NY, May 1983a.

Hinton, G. E. & Sejnowski, T. J. Optimal perceptual inference. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, Washington DC, June 1983b.

Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences USA*, 1982, 79 pp 2554-2558.

Hummel, R. A., & Zucker, S. W. On the foundations of relaxation labeling processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1983, PAMI-5, 267-287.

Kirkpatrick, S. Gelatt, C. D. & Vecchi, M. P. Optimization by simulated annealing. *Science*, 1983, 220, 671-680.

Kullback, S. *Information Theory and Statistics.* New York: Wiley, 1959.

Marr, D. *Vision*. San Francisco: Freeman, 1982.

Minsky, M. & Papert, S. *Perceptrons*. Cambridge, Mass. MIT Press, 1969.

Moussoris, J. Gibbs and Markov random systems with constraints. *Journal of statistical physics*, 1974, 10, 11-33.

Rosenblatt, F. *Principles of neurodynamics.* Washington D.C.: Spartan, 1961.

Rosenfeld, A., Hummel, R. A., & Zucker, S. W. Scene labeling by relaxation operations. *IEEE Transactions on Systems, Man, and Cybernetics*, 1976, SMC-6, 420-433.

Rumelhart, D. E. & Zipser, D. Competitive Learning. *Cognitive Science*, 1985, 9.

Sejnowski, T. J., Hinton, G. E., Kienker, P. & Schumacher, L. (in preparation).

Smolensky, P., Schema selection and stochastic inference in modular environments. *Proceedings of the National Conference on Artificial Intelligence AAAI-83*, Washington, DC, August 1983