Technical Report CMU-CS-84-119

Boltzmann Machines: Constraint Satisfaction Networks that Learn *

Geoffrey E. Hinton
Department of Computer Science
Carnegie-Mellon University, Pittsburgh, PA 15213

Terrence J. Sejnowski
Department of Biophysics
The Johns Hopkins University, Baltimore, MD 21218

David H. Ackley
Department of Computer Science
Carnegie-Mellon University, Pittsburgh, PA 15213

May, 1984

Abstract

The computational power of massively parallel networks of simple processing elements resides in the communication bandwidth provided by the hardware connections between elements. These connections can allow a significant fraction of the knowledge of the system to be applied to an instance of a problem in a very short time. One kind of computation for which massively parallel networks appear to be well suited is large constraint satisfaction searches, but to use the connections efficiently two conditions must be met: First, a search technique that is suitable for parallel networks must be found. Second, there must be some way of choosing internal representations which allow the pre-existing hardware connections to be used efficiently for encoding the constraints in the domain being searched. We describe a general parallel search method, based on statistical mechanics, and we show how it leads to a general learning rule for modifying the connection strengths so as to incorporate knowledge about a task domain in an efficient way. We describe some simple examples in which the learning algorithm creates internal representations that are demonstrably the most efficient way of using the pre-existing connectivity structure.

*The research reported here was supported by grants from the System Development Foundation. We thank Peter Brown, Francis Crick, Mark Derthick, Scott Fahlman, Jerry Feldman, Stuart Geman, Gail Gong, John Hopfield, Jay McClelland, Barak Pearlmutter, Harry Printz, Dave Rumelhart, Tim Shallice, Paul Smolensky, Rick Szeliski, and Venkataraman Venkatasubramanian for helpful discussions.

Table of Contents

The first 5 sections of this technical report together with the conclusion and appendix will appear as a paper in the journal **Cognitive Science.** The reference is:

Ackley, D. H., Hinton, G. E., and Sejnowski, T. J. A learning algorithm for Boltzmann Machines. **Cognitive Science (in press).**

1 Introduction

Evidence about the architecture of the brain and the potential of the new VLSI technology have led to a resurgence of interest in "connectionist" systems (Hinton & Anderson, 1981; Feldman & Ballard, 1982) that store their long term knowledge as the strengths of the connections between simple neuron-like processing elements. These networks are clearly suited to tasks like vision that can be performed efficiently in parallel networks which have physical connections in just the places where processes need to communicate. For problems like surface interpolation from sparse depth data (Grimson, 1981; Terzopoulos, 1984) where the necessary decision units and communication paths can be determined in advance, it is relatively easy to see how to make good use of massive parallelism. The more difficult problem is to discover parallel organizations that do not require so much problem-dependent information to be built into the architecture of the network. Ideally, such a system would adapt a given structure of processors and communication paths to whatever problem it was faced with.

This paper presents a type of parallel constraint satisfaction network which we call a "Boltzmann Machine" that is capable of learning the underlying constraints that characterize a domain simply by being shown examples from the domain. The network modifies the strengths of its connections so as to construct an internal *generative* model that produces examples with the same probability distribution as the examples it is shown. Then, when shown any particular example, the network can "interpret" it by finding values of the variables in the internal model that would generate the example. When shown a partial example, the network can complete it by finding internal variable values that generate the partial example and using them to generate the remainder. At present, we have an interesting mathematical result that guarantees that a certain learning procedure will build internal representations which allow the connection strengths to capture the underlying constraints that are implicit in a large ensemble of examples taken from a domain. We also have simulations that show that the theory works for some simple cases, but the current version of the learning algorithm is very slow.

The search for general principles that allow parallel networks to learn the structure of their environment has often begun with the assumption that networks are randomly wired. This seems to us to be just as wrong as the view that *all* knowledge is innate. If there are connectivity structures that are good for particular tasks that the network will have to perform, it is much more efficient to build these in at the start. However, not all tasks can be foreseen, and even for ones that can, fine-tuning may still be helpful.

Another common belief is that a general connectionist learning rule would make sequential "rule-based" models unnecessary. We believe that this view stems from a misunderstanding of the need for multiple levels of description of large systems, which can be usefully viewed as either parallel or serial depending on the

grain of the analysis. Most of the key issues and questions that have been studied in the context of sequential models do not magically disappear in connectionist models. It is still necessary to perform searches for good solutions to problems or good interpretations of perceptual input, and to create complex internal representations. Ultimately it will be necessary to bridge the gap between hardware-oriented connectionist descriptions and the more abstract symbol manipulation models that have proved to be an extremely powerful and pervasive way of describing human information processing (Newell & Simon, 1972).

2 The Boltzmann Machine

The Boltzmann Machine is a parallel computational organization that is well suited to constraint satisfaction tasks involving large numbers of "weak" constraints. Constraint-satisfaction searches (e.g. Waltz, 1975; Winston, 1984) normally use "strong" constraints that *must* be satisfied by any solution. In problem domains such as games and puzzles, for example, the goal criteria often have this character, so strong constraints are the rule.[1] In some problem domains, such as finding the most plausible interpretation of an image, many of the criteria are not all-or-none, and frequently even the best possible solution violates some constraints (Hinton, 1977). A variation that is more appropriate for such domains uses weak constraints that incur a cost when violated. The quality of a solution is then determined by the total cost of all the constraints that it violates. In a perceptual interpretation task, for example, this total cost should reflect the implausibility of the interpretation.

The machine is composed of primitive computing elements called *units* that are connected to each other by bidirectional *links*. A unit is always in one of two states, *on* or *off*, and it adopts these states as a probabilistic function of the states of its neighboring units and the *weights* on its links to them. The weights can take on real values of either sign. A unit being on or off is taken to mean that the system currently accepts or rejects some elemental hypothesis about the the domain. The weight on a link represents a weak pairwise constraint between two hypotheses. A positive weight indicates that the two hypotheses tend to support one another; if one is currently accepted, accepting the other should be more likely. Conversely, a negative weight suggests, other things being equal, that the two hypotheses should not both be accepted. Link weights are *symmetric*, having the same strength in both directions (Hinton & Sejnowski, 1983).[2]

---

[1] But see (Berliner & Ackley, 1982) for argument that, even in such domains, strong constraints must be used only where absolutely necessary for legal play, and in particular must not propagate into the determination of *good* play.

[2] Requiring the weights to be symmetric may seem to restrict the constraints that can be represented. Although a constraint on boolean variables A and B such as "$A \equiv B$ with a penalty of 2 points for violation" is obviously symmetric in A and B, "$A \Rightarrow B$ with a penalty of 2 points for violation" appears to be fundamentally asymmetric. Nevertheless, this constraint can be represented by the combination of a constraint on A alone and a symmetric pairwise constraint as follows: "Lose 2 points if A is true" and "Win 2 points if both A and B are true."

The resulting structure is related to a system described by Hopfield (1982), and as in his system, each global state of the network can be assigned a single number called the "energy" of that state. With the right assumptions, the individual units can be made to act so as to *minimize the global energy.* If *some* of the units are externally forced or "clamped" into particular states to represent a particular input, the system will then find the minimum energy configuration that is compatible with that input. The energy of a configuration can be interpreted as the extent to which that combination of hypotheses violates the constraints implicit in the problem domain, so in minimizing energy the system evolves towards "interpretations" of that input that increasingly satisfy the constraints of the problem domain.

The energy of a global configuration 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 between units i and j, s_i is 1 if unit i is on and 0 otherwise, and θ_i is a threshold.

2.1 Minimizing energy

A simple algorithm for finding a combination of truth values that is a *local* minimum is to switch each hypothesis into whichever of its two states yields the lower total energy given the current states of the other hypotheses. If hardware units make their decisions asynchronously, and if transmission times are negligible, then the system always settles into a local energy minimum (Hopfield, 1982). Because the connections are symmetric, the difference between the energy of the whole system with the k^{th} hypothesis rejected and its energy with the k^{th} hypothesis accepted can be determined locally by the k^{th} unit, and this "energy gap" 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 *on* state if its total input from the other units and from outside the system exceeds its threshold. This is the familiar rule for binary threshold units.

The threshold terms can be eliminated from Eqs. (1) and (2) by making the following observation: the effect of θ_i on the global energy or on the energy gap of an individual unit is identical to the effect of a link with strength $-\theta_i$ between unit i and a special unit that is by definition always held in the *on* state. This "true unit" need have no physical reality, but it simplifies the computations by allowing the threshold of a unit to be treated in the same manner as the links. The value $-\theta_i$ is called the *bias* of unit i. If a permanently active "true unit" is assumed to be part of every network, then Eqs. (1) and (2) can be written as:

$$E = -\sum_{i<j} w_{ij} s_i s_j \tag{3}$$

$$\Delta E_k = \sum_i w_{ki} s_i \tag{4}$$

2.2 Using noise to escape from local minima

The simple, deterministic algorithm suffers from the standard weakness of gradient descent methods: It gets stuck in *local* minima that are not globally optimal. This is not a problem in Hopfield's system because the local energy minima of his network are used to store "items": If the system is started near some local minimum, the desired behavior is to fall into that minimum, not to find the global minimum. For constraint satisfaction tasks, however, the system must try to escape from local minima in order to find the configuration that is the global minimum given the current input.

A simple way to get out of local minima is to occasionally allow jumps to configurations of higher energy. An algorithm with this property was introduced by Metropolis et. al. (1953) to study average properties of thermodynamic systems (Binder, 1978) and has recently been applied to problems of constraint satisfaction (Kirkpatrick, Gelatt, & Vecchi, 1983). We adopt a form of the Metropolis algorithm that is suitable for parallel computation: If the energy gap between the *on* and *off* states of the k^{th} unit is ΔE_k then regardless of the previous state set $s_k = 1$ with probability

$$p_k = \frac{1}{(1 + e^{-\Delta E_k / T})} \tag{5}$$

where T is a parameter that acts like temperature (see Figure 1).

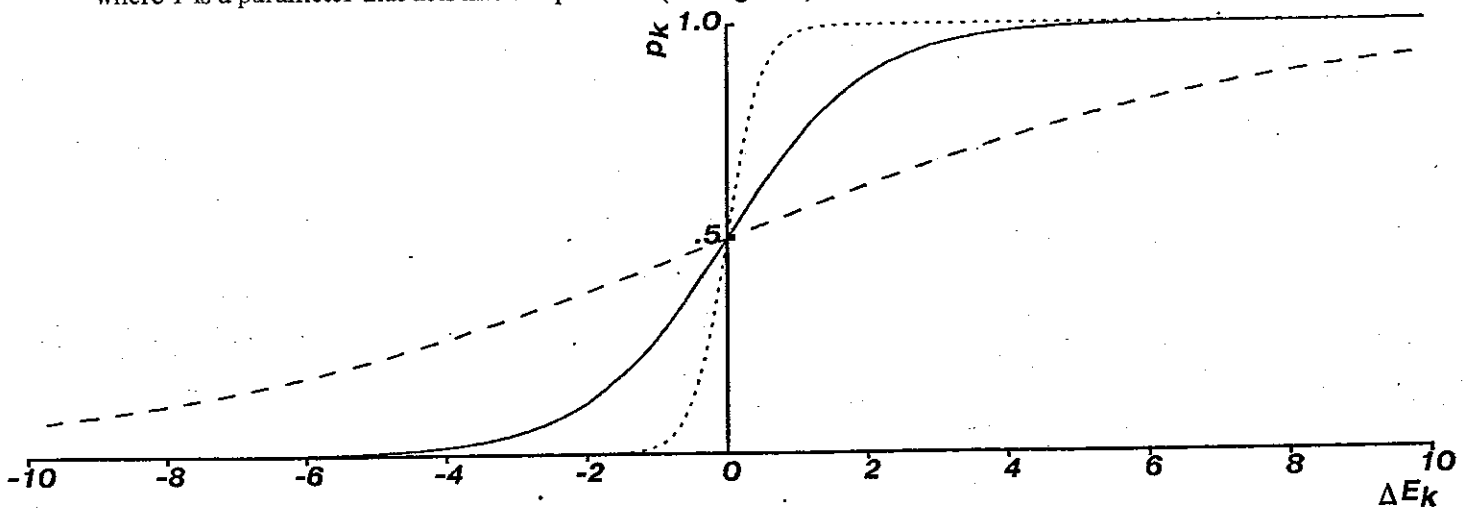


Figure 1: Eq. (5) at $T = 1.0$ *(solid)*, $T = 4.0$ *(dashed)*, and $T = 0.25$ *(dotted)*.

The decision rule in Eq. (5) is the same as that for a particle which has two energy states. A system of such particles in contact with a heat bath at a given temperature will eventually reach thermal equilibrium and the probability of finding the system in any global state will then obey a Boltzmann distribution. Similarly, a network of units obeying this decision rule will eventually reach "thermal equilibrium" and the relative probability of two global states will follow the Boltzmann distribution:

$$\frac{P_\alpha}{P_\beta} = e^{-(E_\alpha - E_\beta)/T} \tag{6}$$

where P_α is the probability of being in the α^{th} global state, and E_α is the energy of that state.

The Boltzmann distribution has some beautiful mathematical properties and it is intimately related to information theory. In particular, the difference in the log probabilities of two global states is just their energy difference (at a temperature of 1). The simplicity of this relationship and the fact that the equilibrium distribution is independent of the path followed in reaching equilibrium are what make Boltzmann machines interesting.

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. A good way to beat this trade-off is to start at a high temperature and gradually reduce it. This corresponds to annealing a physical system (Kirkpatrick et. al. 1983). At high temperatures, the network will ignore small energy differences and will approach equilibrium rapidly. In doing so it will perform a search of the coarse overall structure of the space of global states, and will find a good minimum at that coarse level. As the temperature is lowered, it will begin to respond to smaller energy differences and will find one of the better minima within the coarse-scale minimum it discovered at high temperature. Kirkpatrick et. al. have shown that this way of searching the coarse structure before the fine is very effective for combinatorial problems like graph partitioning, and we believe it will also prove useful when trying to satisfy multiple weak constraints, even though it will clearly fail in cases where the best solution corresponds to a minimum that is deep, narrow and isolated.

3 A learning algorithm

Perhaps the most interesting aspect of the Boltzmann Machine formulation is that it leads to a domain-independent learning algorithm that modifies the connection strengths between units in such a way that the whole network develops an internal model which captures the underlying structure of its environment. There has been a long history of failure in the search for such algorithms (Newell, 1982), and many people (particularly in Artificial Intelligence) now believe that no such algorithms exist. The major technical stumbling block which prevented the generalization of simple learning algorithms to more complex networks

was this: To be capable of interesting computations, a network must contain non-linear elements that are not directly constrained by the input, and when such a network does the wrong thing it appears to be impossible to decide which of the many connection strengths is at fault. This "credit-assignment" problem was what led to the demise of Perceptrons (Rosenblatt, 1961; Minsky & Papert, 1968). The perceptron convergence theorem guarantees that the weights of a single layer of decision units can be trained, but it could not be generalized to networks of such units when the task did not directly specify how to use all the units in the network.

This version of the credit-assignment problem can be solved within the Boltzmann Machine formulation. By using the right stochastic decision rule, and by running the network until it reaches "thermal equilibrium" at some finite temperature, we achieve a mathematically simple relationship between the probability of a global state and its energy. For a network that is running freely without any input from the environment, this relationship is given by Eq. (6). Because the energy is a *linear* function of the weights (Eq. 1) this leads to a remarkably simple relationship between the log probabilities of global states and the individual connection strengths:

$$\frac{\partial \ln P_\alpha}{\partial w_{ij}} = \frac{1}{T}\left[s_i^\alpha s_j^\alpha - p_{ij}'\right] \tag{7}$$

where s_i^α is the state of the i^{th} unit in the α^{th} global state (so $s_i^\alpha s_j^\alpha$ is 1 only if units i and j are both on in state α), and p_{ij}' is just the probability of finding the two units i and j on at the same time when the system is at equilibrium.

Given Eq. (7), it is possible to manipulate the log probabilities of global states. If the environment directly specifies the required probabilities P_α for each global state α, there is a straightforward way of converging on a set of weights that achieve those probabilities, provided any such set exists (See Hinton & Sejnowski, 1983a for details). However, this is not a particularly interesting kind of learning because the system has to be given the required probabilities of *complete* global states. This means that the central question of what internal representation should be used has already been decided by the environment. The interesting problem arises when the environment implicitly contains high-order constraints and the network must choose internal representations that allow these constraints to be expressed efficiently.

3.1 Modeling the underlying structure of an environment

The units of a Boltzmann Machine partition into two functional groups, a non-empty set of *visible* units and a possibly empty set of *hidden* units. The visible units are the interface between the network and the environment; during training all the visible units are clamped into specific states by the environment; when testing for completion ability any subset of the visible units may be clamped. The hidden units, if any, are

never clamped by the environment and can be used to "explain" underlying constraints in the ensemble of input vectors that cannot be represented by pairwise constraints among the visible units. A hidden unit would be needed, for example, if the environment demanded that the states of three visible units should have even parity — a regularity that cannot be enforced by pairwise interactions alone. Using hidden units to represent more complex hypotheses about the states of the visible units, such higher-order constraints among the visible units can be reduced to first and second-order constraints among the whole set of units.

We assume that each of the environmental input vectors persists for long enough to allow the network to approach thermal equilibrium, and we ignore any structure that may exist in the *sequence* of environmental vectors. The structure of an environment can then be specified by giving the probability distribution over all 2^v states of the v visible units. The network will be said to have a perfect model of the environment if it achieves exactly the same probability distribution over these 2^v states when it is running freely at thermal equilibrium with all units unclamped so there is no environmental input.

Unless the number of hidden units is exponentially large compared to the number of visible units, it will be impossible to achieve a *perfect* model because even if the network is totally connected the $(v+h-1)(v+h)/2$ weights and $(v+h)$ biases among the v visible and h hidden units will be insufficient to model the 2^v probabilities of the states of the visible units specified by the environment. However, if there are regularities in the environment, and if the network uses its hidden units to capture these regularities, it may achieve a good match to the environmental probabilities.

An information-theoretic measure of the discrepancy between the network's internal model and the environment is

$$G = \sum_{\alpha} P(V_\alpha) \ln \frac{P(V_\alpha)}{P'(V_\alpha)} \tag{8}$$

where $P(V_\alpha)$ is the probability of the α^{th} state of the visible units when their states are determined by the environment, and $P'(V_\alpha)$ is the corresponding probability when the network is running freely with no environmental input. The G metric, sometimes called the asymmetric divergence or information gain (Kullback, 1959; Renyi, 1962), is a measure of the distance from the distribution given by the $P'(V_\alpha)$ to the distribution given by the $P(V_\alpha)$. G is zero if and only if the distributions are identical; otherwise it is positive.

The term $P'(V_\alpha)$ depends on the weights, and so G can be altered by changing them. To perform gradient descent in G, it is necessary to know the partial derivative of G with respect to each individual weight. In most cross-coupled non-linear networks it is very hard to derive this quantity, but because of the simple relationships that hold at thermal equilibrium, the partial derivative of G is straightforward to derive for our

networks. The probabilities of global states are determined by their energies (Eq. 6) and the energies are determined by the weights (Eq. 1). Using these equations the partial derivative of G (see the appendix) is:

$$\frac{\partial G}{\partial w_{ij}} = -\frac{1}{T}(p_{ij} - p'_{ij}) \tag{9}$$

where p_{ij} is the average probability of two units both being in the *on* state when the environment is clamping the states of the visible units, and p'_{ij}, as in Eq. (7), is the corresponding probability when the environmental input is not present and the network is running freely. (Both these probabilities must be measured at equilibrium). Note the similarity between this equation and Eq. (7), which shows how changing a weight affects the log probability of a single state.

To minimize G, it is therefore sufficient to observe p_{ij} and p'_{ij} when the network is at thermal equilibrium, and to change each weight by an amount proportional to the difference between these two probabilities:

$$\Delta w_{ij} = \varepsilon(p_{ij} - p'_{ij}) \tag{10}$$

where ε scales the size of each weight change.

A surprising feature of this rule is that it uses only *locally available* information. The change in a weight depends only on the behavior of the two units it connects, even though the change optimizes a global measure, and the best value for each weight depends on the values of all the other weights. If there are no hidden units, it can be shown that G-space is concave (when viewed from above) so that simple gradient descent will not get trapped at poor local minima. With hidden units, however, there can be local minima that correspond to different ways of using the hidden units to represent the higher-order constraints that are implicit in the probability distribution of environmental vectors. Some techniques for handling these more complex G-spaces are discussed in the next section.

Once G has been minimized the network will have captured as well as possible the regularities in the environment, and these regularities will be enforced when performing completion. An alternative view is that the network, in minimizing G, is finding the set of weights that is most likely to have generated the set of environmental vectors. It can be shown that maximizing this likelihood is mathematically equivalent to minimizing G (Peter Brown, personal communication).

3.2 Controlling the learning

There are a number of free parameters and possible variations in the learning algorithm presented above. As well as the size of ϵ, which determines the size of each step taken for gradient descent, the lengths of time over which p_{ij} and p'_{ij} are estimated have a significant impact on the learning process. The values employed for the simulations presented here were selected primarily on the basis of empirical observations.

A practical system which estimates p_{ij} and p'_{ij} will necessarily have some noise in the estimates, leading to occasional "uphill steps" in the value of G. Since hidden units in a network can create local minima in G, this is not necessarily a liability. The effect of the noise in the estimates can be reduced, if desired, by using a small value for ϵ or by collecting statistics for a longer time, and so it is relatively easy to implement an annealing search for the minimum of G.

The objective function G is a metric that specifies how well two probability distributions match. Problems arise if an environment specifies that only a small subset of the possible patterns over the visible units ever occur. By default, the unmentioned patterns must occur with probability zero, and the only way a Boltzmann Machine running at a non-zero temperature can guarantee that certain configurations *never* occur is to give those configurations infinitely high energy, which requires infinitely large weights.

One way to avoid this implicit demand for infinite weights is to occasionally provide "noisy" input vectors. This can be done by filtering the "correct" input vectors through a process that has a small probability of reversing each of the bits. These noisy vectors are then clamped on the visible units. If the noise is small, the correct vectors will dominate the statistics, but every vector will have some chance of occurring and so infinite energies will not be needed. This "noisy clamping" technique was used for all the examples presented here. It works quite well, but we are not entirely satisfied with it and have been investigating other methods of preventing the weights from growing too large when only a few of the possible input vectors ever occur.

The simulations presented in the next section employed a modification of the obvious steepest descent method implied by Eq. (10). Instead of changing w_{ij} by an amount proportional to $p_{ij} - p'_{ij}$, it is simply incremented by a fixed "weight-step" if $p_{ij} > p'_{ij}$ and decremented by the same amount if $p_{ij} < p'_{ij}$. The advantage of this method over steepest descent is that it can cope with wide variations in the first and second derivatives of G. It can make significant progress on dimensions where G changes gently without taking very large divergent steps on dimensions where G falls rapidly and then rises rapidly again. There is no suitable value for the ϵ in Eq. (10) in such cases. Any value large enough to allow progress along the gently sloping

floor of a ravine will cause divergent oscillations up and down the steep sides of the ravine.[3]

4 The encoder problem

The "encoder problem" (suggested to us by Sanjaya Addanki) is a simple abstraction of the recurring task of communicating information among various components of a parallel network. We have used this problem to test out the learning algorithm because it is clear what the optimal solution is like and it is non-trivial to discover it. Two groups of visible units, designated V_1 and V_2, represent two systems that wish to communicate their states. Each group has v units. In the simple formulation we consider here, each group has only one unit on at a time, so there are only v different states of each group. V_1 and V_2 are not connected directly but both are connected to a group of h hidden units H, with $h<v$ so H may act as a limited capacity bottleneck through which information about the states of V_1 and V_2 must be squeezed. Since all simulations began with all weights set to zero, finding a solution to such a problem requires that the two visible groups come to agree upon the meanings of a set of codes without any *a priori* conventions for communication through H.

To permit perfect communication between the visible groups, it must be the case that $h \geq \log_2 v$. We investigated minimal cases in which $h = \log_2 v$, and cases when h was somewhat larger than $\log_2 v$. In all cases, the environment for the network consisted of v equiprobable vectors of length $2v$ which specified that one unit in V_1 and the corresponding unit in V_2 should be on together with all other units off. Each visible group is completely connected internally and each is completely connected to H, but the units in H are not connected to each other.

Because of the severe speed limitation of simulation on a sequential machine, and because the learning requires many annealings, we have primarily experimented with small versions of the encoder problem. For example, Figure 2 shows a good solution to a "4-2-4" encoder problem in which $v=4$ and $h=2$. The interconnections between the visible groups and H have developed a binary coding — each visible unit causes a different pattern of *on* and *off* states in the units of H, and corresponding units in V_1 and V_2 support identical patterns in H. Note how the bias of the second unit of V_1 and V_2 is positive to compensate for the fact that the code which represents that unit has all the H units turned off.

---

[3]The problem of finding a suitable value for ϵ disappears if one performs a line search for the lowest value of G along the current direction of steepest descent, but line searches are inapplicable in this case. *Only* the local gradient is available. There are bounds on the second derivative that can be used to pick conservative values of ϵ (Mark Derthick, personal communication), and methods of this kind are currently under investigation.

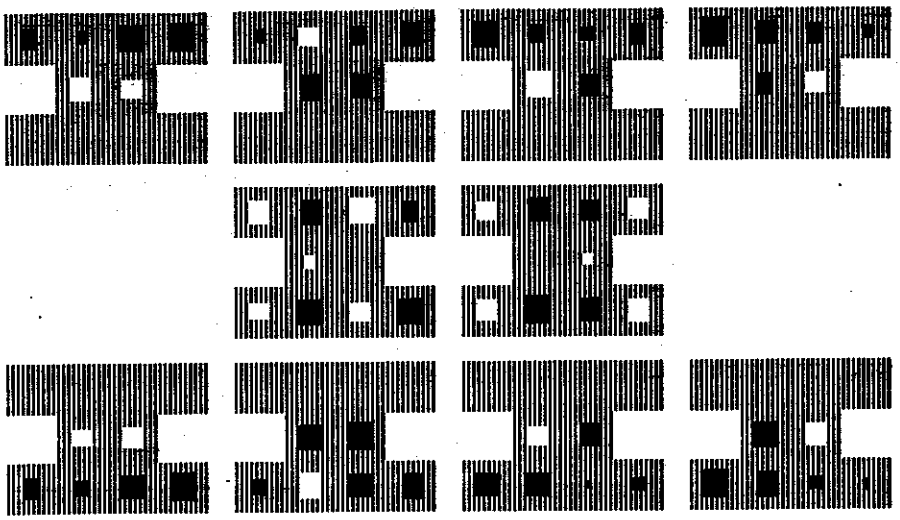


Figure 2: A solution to an encoder problem. The link weights are displayed using a recursive notation. Each unit is represented by a shaded I-shaped box; from top to bottom the rows of boxes represent groups V_1, H, and V_2. Within each box, the black or white rectangles show the strengths of that unit's connections to the other units. The *size* of a rectangle indicates the magnitude of the weight; white rectangles represent positive weights and black rectangles represent negative weights. Within the box representing a unit, the *relative position* of a rectangle indicates which other unit is involved in the connection. For example, the large white rectangle in the third unit along in the top row represents a positive connection to the first unit in the second row. All connections between units appear twice in the diagram, once in the box for each of the two units being connected. For example, the connection described above also appears in the first box of the second row of units, but it occupies a different relative position within this box. To give an idea of the scale of the weights, this connection has a weight of 26. In the position that would correspond to a unit connecting to itself (the second position in the top row of the second unit in the top row, for example), the bias is displayed.

4.1 The 4-2-4 encoder

The experiments on networks with $v = 4$ and $h = 2$ were performed using the following learning cycle:

1. *Estimation of p_{ij}:* Each environmental vector in turn was clamped over the visible units. For each environmental vector, the network was allowed to reach equilibrium twice. Statistics about how often pairs of units were both on together were gathered at equilibrium. To prevent the weights from growing too large we used the "noisy" clamping technique described in Section 3.2. Each *on* bit of a clamped vector was set to *off* with a probability of 0.15 and each *off* bit was set to *on* with a probability of 0.05.

2. *Estimation of p'_{ij}:* The network was completely unclamped and allowed to reach equilibrium at a temperature of 10. Statistics about co-occurrences were then gathered for as many annealings as were used to estimate p_{ij}.

3. *Updating the weights:* All weights in the network were incremented or decremented by a fixed weight-step of 2, with the sign of the increment being determined by the sign of $p_{ij} - p'_{ij}$.

When a settling to equilibrium was required, all the unclamped units were randomized with equal probability on or off (corresponding to raising the temperature to infinity), and then the network was allowed to run for the following times at the following temperatures: [2@20, 2@15, 2@12, 4@10].[4] After this annealing schedule it was assumed that the network had reached equilibrium, and statistics were collected at a temperature of 10 for 10 units of time.

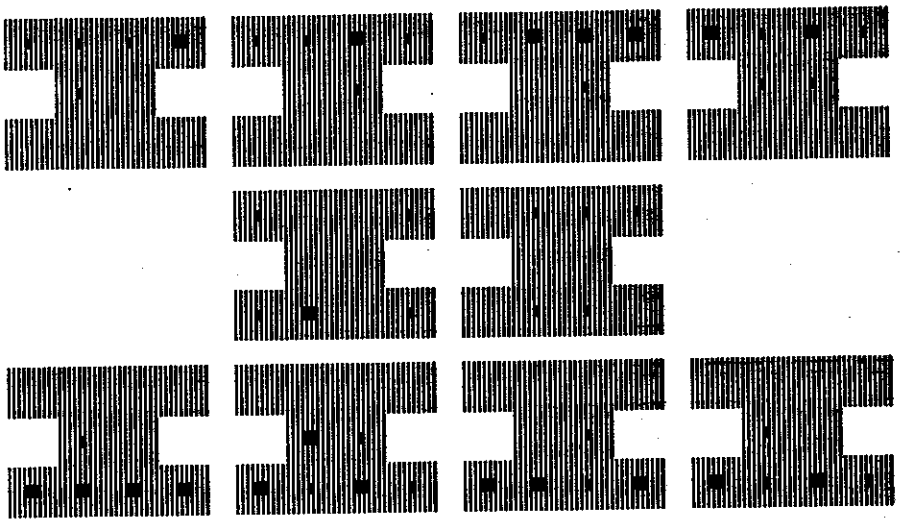
We observed three main phases in the search for the global minimum of G, and found that the occurrence of these phases was relatively insensitive to the precise parameters used. The first phase begins with all the weights set to zero, and is characterized by the development of negative weights throughout most of the network, implementing two winner-take-all networks that model the simplest aspect of the environmental structure — only one unit in each visible group is normally active at a time. In a 4-2-4 encoder, for example, the number of possible patterns over the visible units is 2^8. By implementing a winner-take-all network among each group of four this can be reduced to 4 x 4 low energy patterns. Only the final reduction from 2^4 to 2^2 low energy patterns requires the hidden units to be used for communicating between the two visible groups. Figure 3a shows a 4-2-4 encoder network after 4 learning cycles.

Although the hidden units are exploited for inhibition in the first phase, the lateral inhibition task can be handled by the connections within the visible groups alone. In the second phase, the hidden units begin to develop positive weights to some of the units in the visible groups, and they tend to maintain symmetry between the sign and approximate magnitude of a connection to a unit in V_1 and the corresponding unit in V_2. The second phase finishes when every hidden unit has significant connection weights to each unit in V_1 and analogous weights to each unit in V_2, and most of the different codes are being used, but there are some codes that are used more than once and some not at all. Figure 3b shows the same network after 60 learning cycles.

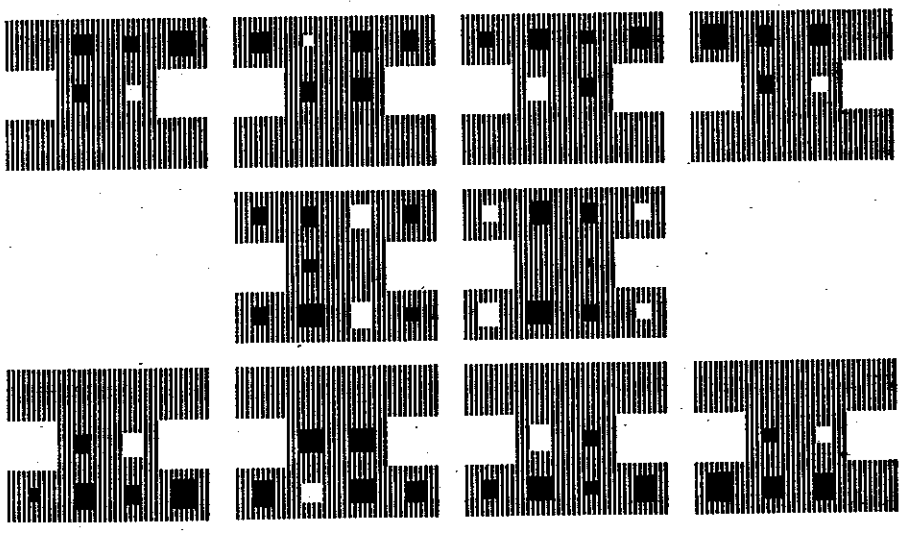
Occasionally all the codes are being used at the end of the second phase in which case the problem is solved. Usually, however, there is a third and longest phase during which the learning algorithm sorts out the remaining conflicts and finds a global minimum. There are two basic mechanisms involved in the sorting out process. Consider the conflict between the 1st and 4th units in Figure 3b, which are both employing the code $\langle -, + \rangle$. When the system is running without environmental input, the two units will be on together quite frequently. Consequently, $p'_{1,4}$ will be higher than $p_{1,4}$ because the environmental input tends to prevent the two units from being on together. Hence the learning algorithm keeps decreasing the weight of the

[4] One unit of time is defined as the time required for each unit to be given, on average, one chance to change its state. This means that if there are n unclamped units, a time period of 1 involves n random probes in which some unit is given a chance to change its state.

connection between the first and fourth units in each group, and they come to inhibit each other strongly. (This effect explains the variations in inhibitory weights in Figure 2. Visible units with similar codes are the ones that inhibit each other strongly.) Visible units thus compete for "territory" in the space of possible codes, and this repulsion effect causes codes to migrate away from similar neighbors. In addition to the repulsion effect, we observed another process that tends to eventually bring the unused codes adjacent (in terms of hamming distance) to codes that are involved in a conflict. The mechanics of this process are somewhat subtle and we do not take the time to expand on them here.



(A)



(B)

Figure 3: Two phases in the development of the perfect binary encoding shown in Figure 2. The weights are shown (A) after 4 learning trials and (B) after 60 learning trials.

The third phase finishes when all the codes are being used, and the weights then tend to increase so that the solution locks in and remains stable against the fluctuations caused by random variations in the cooccurrence statistics. (Figure 2 is the same network shown in Figure 3, after 120 learning cycles.)

In 250 different tests of the 4-2-4 encoder, it always found one of the global minima, and once there it remained there. The median time required to discover four different codes was 110 learning cycles. The longest time was 1810 learning cycles.

4.2 The 4-3-4 encoder

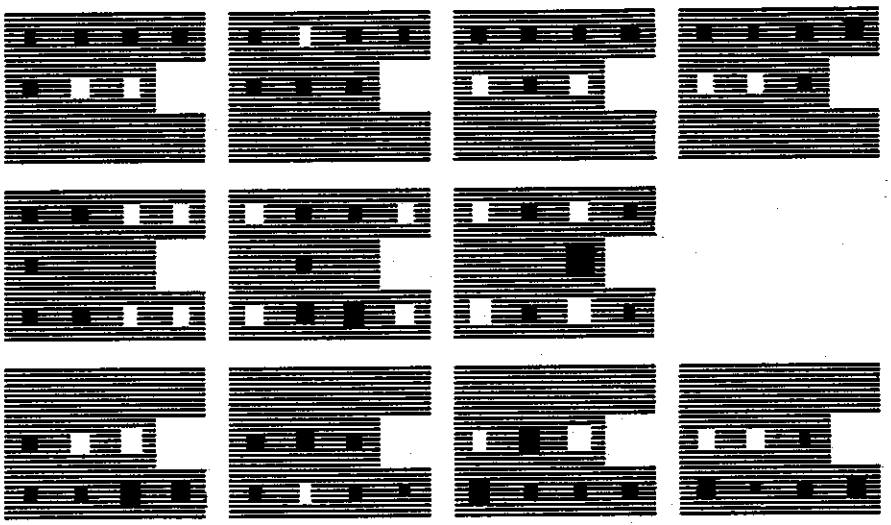


Figure 4: A 4-3-4 encoder that has developed optimally spaced codes.

A variation on the binary encoder problem is to give H more units than are absolutely necessary for encoding the patterns in V_1 and V_2. A simple example is the 4-3-4 encoder which was run with the same parameters as the 4-2-4 encoder. In this case the learning algorithm quickly finds four different codes. Then it always goes on to modify the codes so that they are optimally spaced out and no pair differ by only a single bit, as shown in Figure 4. The median time to find four well-spaced codes was 270 learning cycles and the maximum time in 200 trials was 1090.

4.3 The 8-3-8 encoder

With $v=8$ and $h=3$ it took many more learning cycles to find all 8 three-bit codes. We did 20 simulations, running each for 4000 learning cycles using the same parameters as for the 4-2-4 case (but with a probability of 0.02 of reversing each *off* unit during noisy clamping). The algorithm found all 8 codes in 16 out of 20 simulations and found 7 codes in the rest. The median time to find 7 codes was 210 learning cycles and the median time to find all 8 was 1570 cycles.

The difficulty of finding all 8 codes is not surprising since the fraction of the weight space that counts as a solution is much smaller than in the 4-2-4 case. Sets of weights that use 7 of the 8 different codes are found fairly rapidly and they constitute local minima which are far more numerous than the global minima and have almost as good a value of G. In this type of G-space, the learning algorithm must be carefully tuned to achieve a global minimum, and even then it is very slow. We believe that the G-spaces for which the algorithm is well-suited are ones where there are a great many possible solutions and it is not essential to get the very best one. For large networks to learn in a reasonable time, it may be necessary to have enough units and weights and a liberal enough specification of the task so that no single unit or weight is essential. The next example illustrates the advantages of having some spare capacity.

4.4 The 40-10-40 encoder

A somewhat larger example is the 40-10-40 encoder. The 10 units in H is almost twice the theoretical minimum, but H still acts as a limited bandwidth bottleneck. The learning algorithm works well on this problem. Figure 5 displays the resulting network. Figure 6 shows its performance when given a pattern in V_1 and required to settle to the corresponding pattern in V_2. Each learning cycle involved annealing once with each of the 40 environmental vectors clamped, and the same number of times without clamping. The codes that the network selected to represent the patterns in V_1 and V_2 were all separated by a hamming distance of at least 2, which is very unlikely to happen by chance. As a test, we compared the weights of the connections between visible and hidden units. Each visible unit has 10 weights connecting it to the hidden units, and to avoid errors, the 10 dimensional weight vectors for two different visible units should not be too similar. The cosine of the angle between two vectors was used as a measure of similarity, and no two codes had a similarity greater than 0.73, whereas many pairs had similarities of 0.8 or higher when the same weights were randomly rearranged to provide a control group for comparison.

To achieve good performance on the completion tests, it was necessary to use a very gentle annealing schedule during testing. The schedule spent twice as long at each temperature and went down to half the final temperature of the schedule used during learning. As the annealing was made faster, the error rate increased, thus giving a very natural speed/accuracy trade-off. We have not pursued this issue any further, but it may prove fruitful because some of the better current models of the speed/accuracy trade-off in human reaction time experiments involve the idea of a biased random walk (Ratcliff 1978) and the annealing search gives rise to similar underlying mathematics.

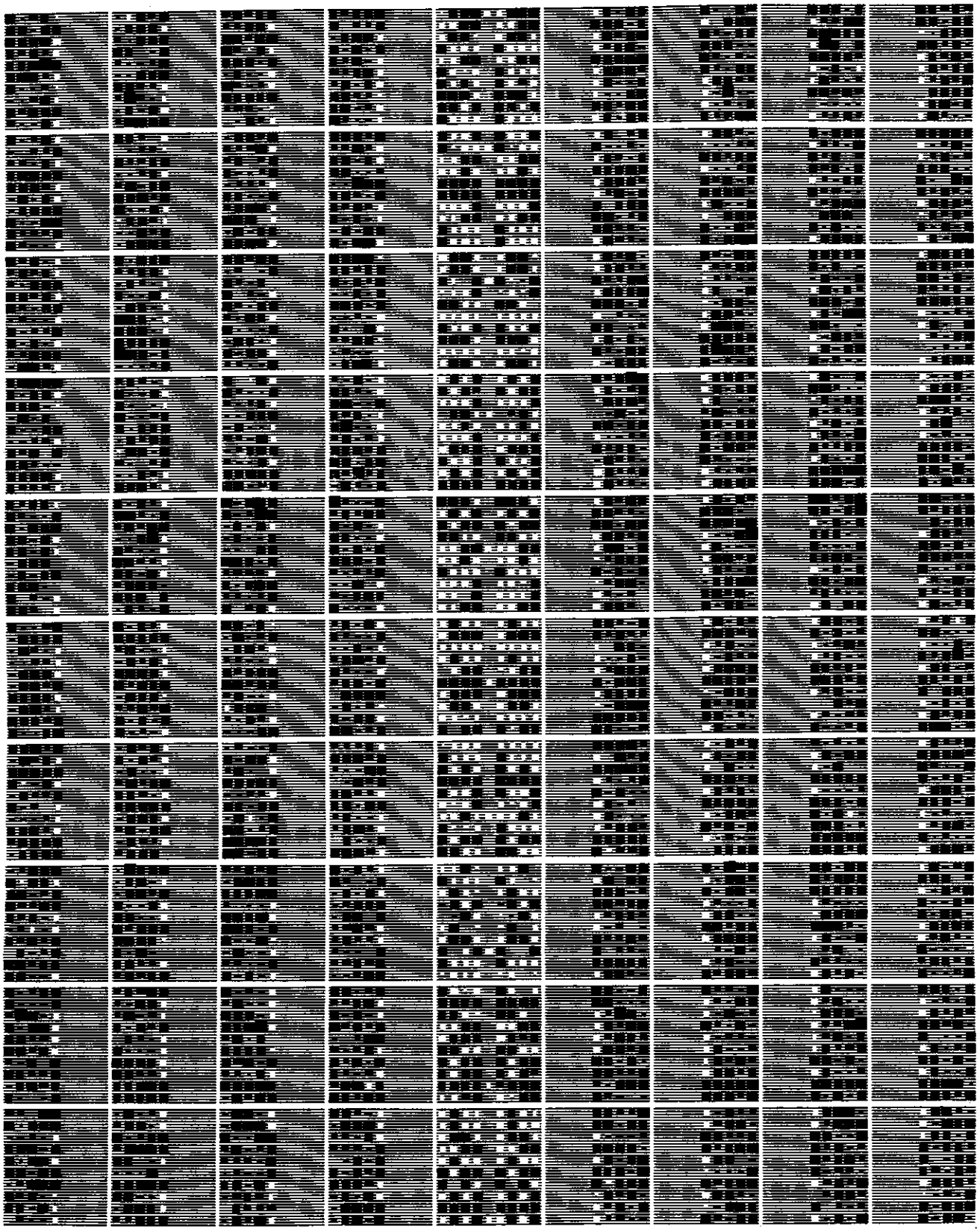


Figure 5: A 40-10-40 encoder network. The center column is group H, and the left and right four columns are groups V_1 and V_2.

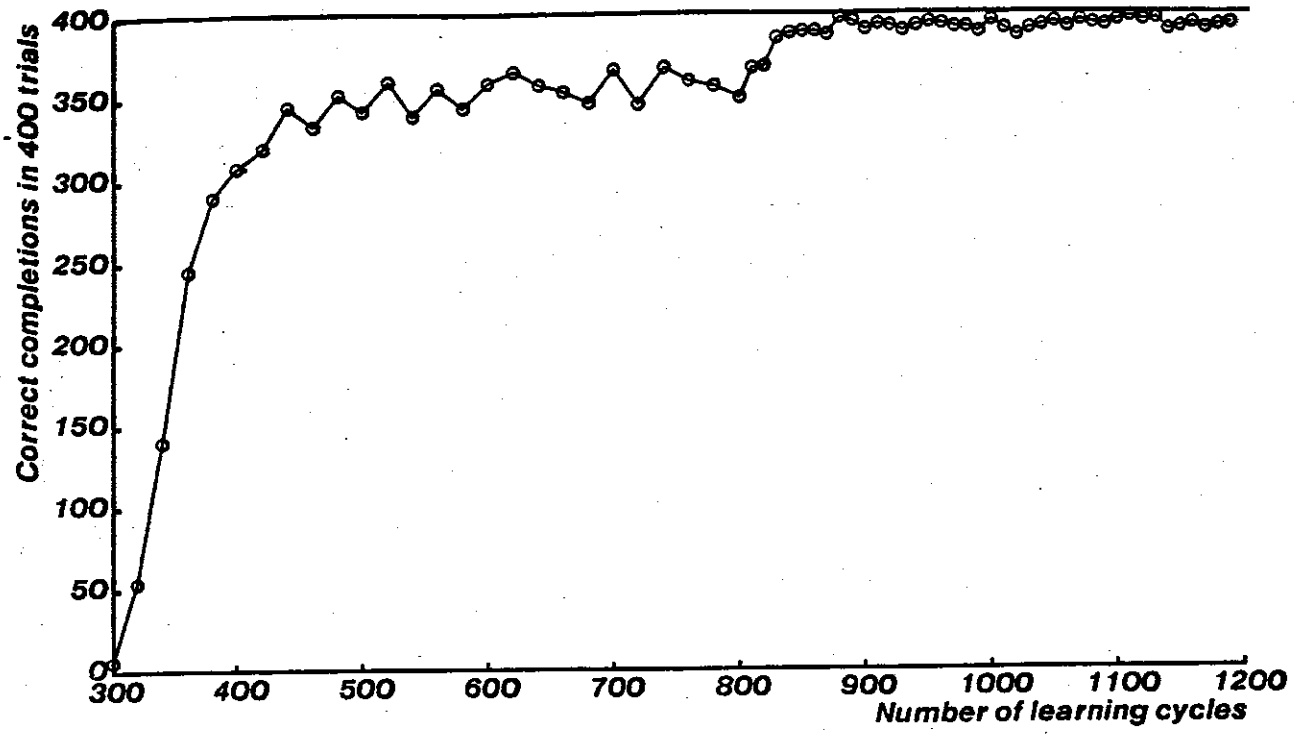


Figure 6: Completion accuracy of a 40-10-40 encoder during learning. The network was tested by clamping the states of the units in V_1 and letting the remainder of the network reach equilibrium. If just the correct unit was on in V_2, the test was successful. This was repeated 10 times for each of the 40 units in V_1. For the first 300 learning cycles the network was run without connecting up the hidden units. This ensured that each group of 40 visible units developed enough lateral inhibition to implement an effective winner-take-all network. The hidden units were then connected up and for the next 500 learning cycles we used "noisy" clamping, switching *on* bits to *off* with a probability of 0.1 and *off* bits to *on* with a probability of 0.0025. After this we removed the noise and this explains the sharp rise in performance after 800 cycles. The final performance asymptotes at 98.6% correct.

5 Representation in parallel networks

So far, we have avoided the issue of how complex concepts would be represented in a Boltzmann machine. The individual units stand for "hypotheses", but what is the relationship between these hypotheses and the kinds of concepts for which we have words? Some workers suggest that a concept should be represented in an essentially "local" fashion: the activation of one or a few computing units is the representation for a concept (Feldman & Ballard, 1982), while others view concepts as "distributed" entities: a particular pattern of activity over a large group of units represents a concept, and different concepts correspond to *alternative* patterns of activity over the same group of units (Hinton, 1981a).

One of the better arguments in favor of local representations is their inherent modularity. Knowledge about relationships between concepts is localized in specific connections and is therefore easy to add, remove, and modify, if some reasonable scheme for forming hardware connections can be found (Feldman, 1981; Fahlman, 1980). With distributed representations, however, the knowledge is diffuse. This is good for tolerance to local hardware damage, but it appears to make the design of modules to perform specific functions much harder. It is particularly difficult to see how new distributed representations of concepts could originate spontaneously.

In a Boltzmann machine, a distributed representation corresponds to an energy minimum, and so the problem of creating a good collection of distributed representations is equivalent to the problem of creating a good "energy landscape." The learning algorithm we have presented is capable of solving this problem, and it therefore makes distributed representations considerably more plausible. The diffuseness of any one piece of knowledge is no longer a serious objection, because the mathematical simplicity of the Boltzmann distribution makes it possible to manipulate all the diffuse local weights in a coherent way on the basis of purely local information. The formation of a simple set of distributed representations is illustrated by the encoder problems.

5.1 Communicating information between modules

The encoder problem examples also suggest a method for communicating symbols between various components of a parallel computational network. Feldman & Ballard (1982) present sketches of two implementations for this task, using the example of the transmission of the concept "wormy apple" from where it is recognized in the perceptual system to where the phrase "wormy apple" can be generated by the speech system. They argue that there appear to be only two ways that this could be accomplished. In the first method, the perceptual information is encoded into a set of symbols that are then transmitted as messages to the speech system, where they are decoded into a form suitable for utterance. In this case, there would be a set of general-purpose communication lines, analogous to a bus in a conventional computer, that would be used as the medium for all such messages from the visual system to the speech system. Feldman & Ballard describe the problems with such a system as:

- Complex messages would presumably have to be transmitted sequentially over the communications lines.
- Both sender and receiver would have to learn the common code for each new concept.
- The method seems biologically implausible as a mechanism for the brain.

The alternative implementation they suggest requires an individual, dedicated hardware pathway for each concept that is communicated from the perceptual system to the speech system. The idea is that the

simultaneous activation of "apple" and "worm" in the perceptual system can be transmitted over private links to their counterparts in the speech system. The critical issues for such an implementation are having the necessary connections available between concepts, and being able to establish new connection pathways as new concepts are learned in the two systems. The main point of this approach is that the links between the computing units carry simple, non-symbolic information such as a single activation level.

The behavior of the Boltzmann machine when presented with an encoder problem demonstrates a way of communicating concepts that largely combines the best of the two implementations mentioned above. Like the second approach, the computing units are small, the links carry a simple numeric value, and the computational and connection requirements are within the range of biological plausibility. Like the first approach, the architecture is such that many different concepts can be transmitted over the same communication lines, allowing for effective use of limited connections. The learning of new codes to represent new concepts emerges automatically as a cooperative process from the G-minimization learning algorithm.

6 The Shifter Problem

The encoder example is a good test case for the learning algorithm because it is clear how the hidden units must be used. However, the encoder example is very untypical in an important way: the only reason the hidden units are needed at all is that the visible units do not have the appropriate direct connections. For most interesting tasks, hidden units are needed even if each visible unit is connected to every other because the sum of a lot of pairwise interactions is incapable of capturing any higher-order statistical structure that exists in the ensemble of patterns which the environment clamps on the visible units.

A simple example which can only be solved by capturing the higher-order structure 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 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 "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

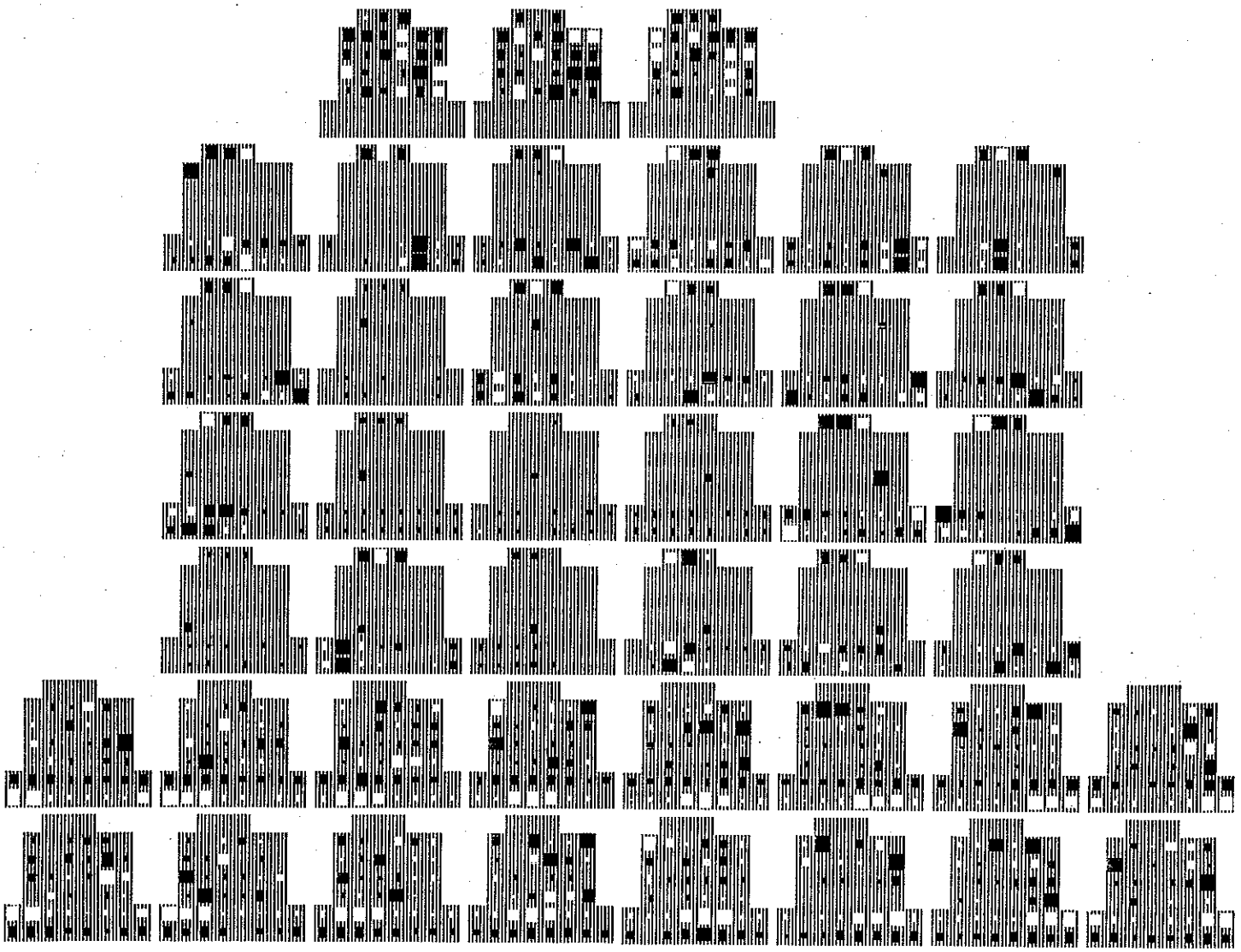


Figure 7: A shifter network. The bottom two rows of eight units are groups V_1 and V_2. The three units of group V_3 appear in the top row. The activation of the first unit of V_3 indicates that a left shift applied to V_1 produces V_2, the second unit indicates no shift, and the third unit indicates a right shift. The central four rows are the 24 hidden units. The connection strengths appearing at the bottom of a hidden unit show its "receptive field" over V_1 and V_2, and those at the top show how it affects the units in V_3.

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.

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 in 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 them.

Figure 7 shows the result of running a variation of the standard learning algorithm. 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 localised. 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 is 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 (Rumelhart & Zipser, 1984; Fukushima, 1980).

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

1. The weights are fairly clearly not optimal because of the 5 hidden units that appear to do nothing useful, and further learning did not fix this. Also, the performance is not as good as it could be. When the states of the units in V_1 and V_2 are clamped and the network is annealed gently to half the final temperature used during learning, the shift units quite frequently adopt the wrong states. If the number of *on* units in V_1 is 1, 2, 3, 4, 5, 6, 7 the percentage of correctly recognized shifts is 50% 71% 81% 86% 89% 82% 66% respectively. The wide variation in the number of active units in V_1 naturally makes the task harder to learn than if a constant proportion of the units were active.

2. When the "standard" version of the learning algorithm described above was applied to the shifter problem there was a pronounced tendency for some of the hidden units to develop weights that caused them to be permanently off. This "suicide" effect is hard to recover from because if a unit is always off, its co-occurence statistics will always be the same (zero) in both the clamped and the free-running phases, and so the estimated derivative of G with respect to any of its weights will be zero. Some of the reasons for the suicide effect are discussed by Derthick (1984). To avoid it we used a variation of the learning algorithm that was suggested and tested by Barak Pearlmutter. This variation is described in Section 6.1.

3. The learning was very slow. It required 9000 learning cycles each of which involved reaching equilibrium 20 times with clamped input vectors and the same number of times without any clamping. The speed of learning is discussed further in Section 7.

6.1 Keeping the weights small

The learning algorithm presupposes that the network reaches thermal equilibrium, and it uses the co-occurence statistics measured at equilibrium to create an energy landscape that models the structure of the ensemble of vectors produced by the environment. Unfortunately there is nothing in the learning algorithm to prevent it from creating an energy landscape that contains large energy barriers which prevent the network from reaching equilibrium. If this happens, the network may perform badly, and more importantly, the statistics that are collected will not be equilibrium statistics so there is no guarantee that the changes in the weights will improve G.

One way to avoid the large weights that tend to make it hard to achieve equilibrium is to keep all the weights small. A simple way of doing this is to redefine the quantity to be minimized as:

$$G + h \sum_{ij} (w_{ij})^2$$

where h is a coefficient that determines the relative importance of minimizing G and keeping the weights small. The effect of the extra term is to make all the weights decay towards zero by an amount proportional to their current magnitude. This ensures that large weights which are not important for achieving a low value of G tend to shrink. For the shifter example shown in Figure 7 the value of h was 0.0005.

7 The speed of learning

The examples presented here took a long time to learn even though they are very small-scale. The time taken is not just a consequence of using serial machines to simulate parallel networks. Even in a truly parallel machine[5] the learning would be slow because the gradient descent requires a great many annealings with different input vectors. This slowness raises several questions:

1. How does the learning time scale with the size of the problem?

2. Can the learning algorithm be generalized to exhibit the kind of "one-shot" learning in which a person is told a fact once and then remembers it for a long time?

3. How much faster is the learning when the connectivity of the network and the initial values of the weights are approximately correct for the task at hand (as they might be for parts of the visual or motor system that have had time to evolve an appropriate architecture)?

4. Do good solutions generally have a particular statistical structure? If so, it may be possible to impose strong *a priori* domain-independent constraints on the values of the weights or the connectivity that will constrain the search for a good set of weights to a subspace. For example, it

[5] A truly parallel machine requires the annealing process to be tolerant of time delays. When one unit makes its decision it will be "unaware" of the states of other units that changed very recently, and this must not prevent the network from reaching equilibrium. See Section 11 for a discussion of time delays.

may be that for many problems a layered network with lateral inhibition within a layer and excitation between layers is a good solution. If so, the search can be confined to this subspace of the weight space.

We do not yet have good answers to any of these questions, but some comments on "one-shot" learning and on the way learning-time scales with the size of the problem may be helpful.

7.1 One-shot learning versus regularity detection

In the shifter problem there are 19 visible units and hence 2^{19} possible vectors that can be clamped on the visible units. Of these, only 3×2^8 actually occur. This is far too many for each pattern to be learned as an independent fact. The only way to learn the task is to capture the relatively low-order underlying regularities, and this is what the hidden units do. It is not surprising that the learning is slow. To reach the weights shown in Figure 7 required 20 x 9000 samples, but this does not seem all that excessive when the problem is expressed as learning a particular subset of 768 out of half a million patterns. A lot of samples must be taken before this subset is revealed.

It is hard for people to realise the difficulty of the task because Figure 7 is laid out in such a way that our existing notions of spatial locality make it easy to express the task. If the visible units were randomly reordered, the learning algorithm would do just as well because it starts from scratch, but people would find the task much harder. When it creates localised feature detectors among the hidden units, the learning algorithm is actually *constructing* spatial locality.

Learning a single new fact that can be expressed in terms of familiar concepts is very different from learning from scratch, particularly if the fact is one which is plausible given the existing knowledge that the system has. In the Boltzmann machine, the existing knowledge is an energy landscape, and to learn a new fact is to create a new energy minimum. For a plausible fact this means taking a state which had an energy somewhat higher than the energy of the known facts and lowering its energy. For successful one-shot learning it is important that creating this new minimum does not disturb the existing minima too much. If we use local representations in which one unit is dedicated to each fact, only the connections to this unit need to be modified, and so it is obvious that a new fact can be incorporated without interfering with existing facts. If we use distributed representations it is less obvious, because each weight will be involved in many minima, and so changing the weights to lower one minimum will interfere with many others.

The following reasoning shows that the interference effect can be made very small by using patterns of activity in which the fraction, f, of the units which are on at any one time is small. To keep the mathematics simple we assume a version of the learning algorithm that performs gradient descent by changing each weight

by $\epsilon(p_{ij} - p'_{ij})$. We also assume that all the global states which occur have the same fraction of *on* bits. This means that the total number of pairs of *on* units remains constant. Hence, the mean values of p_{ij} and p'_{ij} averaged over all connections must be the same and so the total of all the weights must remain constant.

Suppose that the units in the network are totally connected.[6] Suppose also that a set of global states are used to estimate p_{ij} before changing the weights. Now consider the effect of adding a single additional global state, α, to this set. This will have two effects:

1. The weights between units which are both *on* in α will get a bigger increment than before. Define δ to be the size of this increase in the increment.

2. To keep the sum of the weights constant, the remaining weights must have a reduction of $f^2\delta/(1-f^2)$ in their expected increment.

So the global state α will have its energy lowered by $N\delta$ where N is the number of pairs of *on* units in a global state. A global state, β, that is randomly related to α will have about Nf^2 of its weights incremented by δ more than before and the remainder incremented by $f^2\delta/(1-f^2)$ less than before. The net effect of the occurence of α on the energy of β will therefore be about 0, and this effect will be composed of two conflicting effects each of which is only about f^2 of the change in the energy of α. So provided $f \ll 1$ there will be very little unwanted transfer of learning. Only patterns which are significantly similar to α will have their energies significantly effected, and this is just what is need to achieve generalization. Hence, one-shot learning is possible without significantly disrupting the existing knowledge.

7.2 How the learning-time scales

The question of how the learning time depends on the size of the problem is very important, but it is also very complex because many other factors have to be scaled at the same time and it is not at all obvious how they should be scaled. The factors include:

1. The ratio of hidden to visible units.

2. The number of connections per unit.

3. The number of constraints in which each visible unit is involved.

4. The order of the underlying constraints: If the constraints among the visible units become higher order as the problem gets bigger, the problem may get much harder.

5. The compatibility of the constraints: If different constraints typically conflict with one another, the problem may become much harder as its size increases because many constraints must be

[6] The use of sparse connectivity does not affect the argument.

traded off against each other and this means that the system will take a long time to reach equilibrium and will only perform well if the various weak constraints have just the right relative strengths. If, on the other hand, the constraints typically agree with each other the global minimum will be over-determined.[7]. It will be possible to reach equilibrium rapidly, and the learning algorithm will only need to find a subset of the constraints in order to achieve good performance.

The encoder problem illustrates some of these points about scaling. The 8-3-8 encoder takes much longer to achieve 8 different codes than the 4-2-4 takes to achieve 4 different codes. However, the 40-10-40 encoder achieves almost perfect performance in *fewer* annealings than the 8-3-8 encoder.

There are an exponential number of valid constraints that the hidden units can capture in the encoder problem. These constraints have the form: If the active unit in V_1 is in subset S, then the active unit in V_2 must be in the equivalent subset within V_2. If the subset, S, is small the hidden unit will rarely be active and so it will convey little information on average. The best solution is for each hidden unit to dedicate itself to representing a different subset of about half the alternatives. The subsets should be chosen so that when a unit in V_1 is activated, the represented subsets that contain the unit are sufficient to encode which unit it is. For the 4-2-4 and 8-3-8 cases this can only be done by choosing orthogonal subsets each of which contains exactly half the alternatives. For the 40-10-40 case there is much more freedom in choosing the particular constraints that the hidden units should represent, and that is why the learning is relatively easier.

7.3 Degeneracy and the speed of learning

The small examples presented here require each and every visible unit to behave correctly. This may be unreasonable for larger problems. It may well be sufficient to have rather broad, degenerate energy minima in which many of the visible units are not strongly constrained to be on or off. In large networks it will probably be much easier to construct and modify these degenerate minima than to construct deep narrow minima in which the state of every unit is crucial. Narrow deep minima require large weights because changing the states of a few units must make a big change in the energy. Broad minima can be made deep without any of the weights being large.

A further interesting property of broad minima is that they suggest a way in which one concept can be differentiated into several more refined ones. By varying the weights between units that are not firmly on or off within a minimum it is possible to modify the shape of the floor of the minimum and thus to differentiate one large minimum into several closely related minima which are only separated by small energy barriers.

---

[7]The problem of labeling a line-drawing in computer vision has this property. Simple drawings without shadows are often highly ambiguous. More complex drawing with shadows can be *easier* to label because more constraints conspire together to rule out all but the correct labeling (Waltz, 1975)

This kind of differentiation could be used to model what happens when a concept like "dog" gets refined into two more specific concepts like "big nasty dog" and "little nice dog." The small barriers between concepts within the same general class should make it easy for the network to "side-track" to a neighboring, similar concept when the current concept does not quite match the data, as suggested by Minsky (1975).

When a neural model does not work as well as was hoped, it is not uncommon for its creators to claim that if only they had a *really big* network everything would be fine. Despite this unfortunate tendency, we feel that the degeneracy of large broad minima may be very important for the speed of learning and the ease of search, and so large networks may be essential because degeneracy effects disappear in very small networks.

8 Reducing higher-order constraints

An important way of distinguishing between constraints is by their "order." A first-order constraint involves only a single variable; a system consisting of only first-order constraints is trivial, since the constraints can be considered independently. Second-order constraints, involving two variables, are the simplest constraints that can require a combinatoric search. Constraints which *necessarily* involve more than two variables will be called "higher-order." When no strong constraints are involved, weak higher-order constraints among three or more given variables can always be approximated by introducing further variables and using only weak first and second-order constraints among the larger set of variables. This means that a general architecture for solving problems involving weak constraints need only implement first and second-order constraints directly, provided it can automatically reduce higher-order constraints to lower-order ones among a larger set of variables.

Some weak constraints on a set of variables that appear to be higher-order can be expressed as collections of first and second-order constraints without any extra variables. Consider, for example, the constraint on n boolean variables "If exactly one variable is true, win one point, otherwise lose at least one point". This appears to an n^{th}-order constraint, but it can actually be expressed by the following collection of first- and second-order constraints: For each variable "Win one point if the variable is true", and for each pair of variables "Lose two points if the variables are both true".

On the other hand, some third-order constraints cannot be reduced to first- and second-order ones without introducing extra variables. The prototypical example of such a constraint is the exclusive-OR function: C should be true if A is true or B is true, but not both. The difficulty is that only the *combination* of the states of A and B constrains the state of C; the state of A or B alone says nothing about the state of C. Section 8.1 discusses this key example at length and demonstrates how the addition of an extra variable makes the reduction to second-order possible.

In general, there may be an exponentially large number of higher-order constraints among a given set of variables. A practical system will have only a limited number of extra variables at its disposal for doing the reduction to second order, so it must make compromises in selecting the best reduction. Determining which higher-order constraints to reduce, and what to reduce them to, is a central problem for efficient constraint-based knowledge representation. The general purpose learning algorithm presented in Section 3 can be viewed as an automatic method of reducing higher-order constraints among a set of variables to second-order constraints among a larger set. One can view the set of states of the visible units on which the machine is trained as a single, very high-order, disjunctive constraint. To perform search efficiently, the machine must reduce this constraint to a large set of first and second-order constraints, and to do this it must typically use extra "hidden" units that are not mentioned in the task specification.

8.1 An example

The approach and the key issues can be made clearer by considering in some detail a very simple example. Suppose an environment consisted of the following equiprobable vectors: {<0,0,0>,<0,1,1>,<1,0,1>,<1,1,0>}. In this case, there is an obvious rule that characterizes the set: the third element is the exclusive-OR of the first two. That is not the only such rule; it could also be phrased as the second being the exclusive-OR of the first and the third, or as the set of all triples of bits with even parity. The difference between these rules is in how the instances of the problem are specified. For the first rule, an instance would provide constraints on the first and second elements, and in finding a solution state that satisfied those constraints, the system could naturally be viewed as computing the exclusive-OR of the first two elements and representing the result in the third. With the third rule, any one or two of the elements might be constrained by an instance, and the system could be viewed as finding an even parity triple that satisfies the constraints of the instance.

This example is simple because the set of solution states is so small that a learning system of any significant size could simply produce an internal list of the solution states and enumerate them to find a solution state that fits any particular partial pattern. Interesting constraint satisfaction problems are rarely so compact. To learn to be an effective problem-solver, a system must have the ability to extract rules from a presentation of solution states; to extract regularities that tend to characterize membership in the set of solution states.

On the other hand, this example is non-trivial for one important reason. The state of any one element, by itself, provides *no information* about whether any other element should be one or zero. For each pair of elements, the solution set contains all four combinations of ones and zeros. It is for precisely this reason that Perceptrons were unable to compute the exclusive-OR function (Rosenblatt, 1961; Minsky & Papert, 1968). A single-level decision unit is unable to capture the distinguishing feature of all characterizations of this solution set: the states of two of the units, *taken together*, determines the third.

For the same reason, a Boltzmann Machine with only three interconnected units is unable to represent this solution set. Hypothesizing that the first element is a 1, by itself, does not constrain hypotheses about the second or the third sufficiently to solve the problem. However, notice the following decomposition of the set of solution states: $S_1 = \{\langle 0,0,0\rangle, \langle 0,1,1\rangle, \langle 1,0,1\rangle\}, S_2 = \{\langle 1,1,0\rangle\}$. S_1 alone is a partial description of the *inclusive*-OR function, and for that set there *are* constraints between pairs of units.[8] The hypothesis that the first unit is a 1 constrains the third unit also to be a one, and analogously for the second unit. S_1 can be represented by linking the first and second units to the third with positive weights, and providing the third unit with a negative bias, so that in absence of input from either of the other units, the third unit will tend to be off.

The network will now correctly find the elements of S_1, but fails on S_2, preferring $\langle 1,1,1\rangle$ over the correct state. By adding a hidden unit, however, this preference can be overridden. The hidden unit is given positive weights to the first two units, a negative weight to the third unit, and a bias low enough that both of the first two units must be on for the hidden unit to be likely to come on. The magnitudes of the weights are chosen so that the negative weight between the hidden unit and the third unit overrides the positive weights between the third unit and the first two. Figure 8 shows the resulting network.

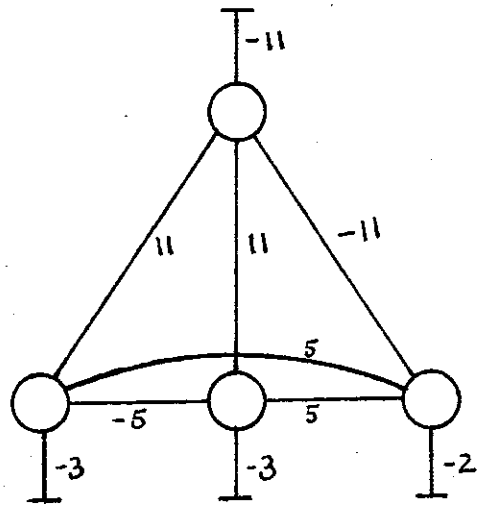


Figure 8: A Boltzmann Machine that computes the exclusive-OR function

It is worth emphasizing that connections in a Boltzmann Machine are *symmetric*; inside the machine there is no concept of "input lines" or "output lines." For example, it can cause confusion to think of the hidden unit in the exclusive-OR network as taking the states of the first two units as "input", and affecting the state of the third unit as "output." There is no preferred direction of causal flow through a Boltzmann Machine, there are only states and energies associated with those states. Units function as "inputs" when they are clamped by the

[8] There are other decompositions which work equally well.

environment and therefore cannot be changed by the network, and they function as "outputs" when they are not clamped, so with the pattern completion approach that we use in this paper, different instances of the same problem can completely redefine the notions of input and output.

The exclusive-OR example is a very simple illustration of a very hard general problem that a learning system must face: the task of discovering and representing the higher-order regularities of the solution set, those that do not manifest themselves as constraints on individual units or constraints between pairs of visible units. Hidden units are not directly constrained by the instances in a solution set and are thus available for reducing these higher-order constraints. Based on the environment and the structure of the network, a learning algorithm must discover distinctions that are helpful for solving the problem, as was the distinction between S_1 and S_2 above.

9 Bayesian inference

Bayesian inference suggests a general paradigm for perceptual interpretation problems. Suppose the probability associated with one unit represents the probability that a particular hypothesis, h, is correct. Suppose, also, that the "true" state of another unit is used to represent the existence of some evidence, e. Bayes theorem prescribes a way of updating the probability of the hypothesis $p(h)$ given the existence of new evidence e:

$$p(h|e) = \frac{p(h)\ p(e|h)}{p(h)\ p(e|h) + p(\bar{h})\ p(e|\bar{h})}$$

$$= 1/\left(1 + \frac{p(\bar{h})\ p(e|\bar{h})}{p(h)\ p(e|h)}\right)$$

$$= 1/\left(1 + e^{-\left(\ln \frac{p(h)}{p(\bar{h})} + \ln \frac{p(e|h)}{p(e|\bar{h})}\right)}\right)$$

where $\bar{h}$ is the negation of h.

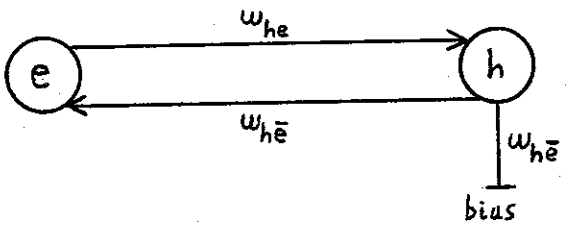
The Bayes rule has the same form as the decision rule in (5) if we identify the probability of the unit with the probability of the hypothesis. The bias of the unit implements the *a priori* likelihood ratio and the weight implements the effect of the evidence provided by the state of the other unit (assuming the temperature is fixed at 1):

$$bias_h = \ln \frac{p(h)}{p(\bar{h})} \qquad w_{he} = \ln \frac{p(e|h)}{p(e|\bar{h})}$$

Bayesian inference with one piece of evidence can therefore be implemented by units of the type we have been considering. There are, however, several difficulties with this simple formulation.

1. It provides no way for the negation of the evidence e to affect the probability of *h*.
2. It does not lead to symmetrical weights when two units affect each other since $p(e|h)/p(e|\bar{h})$ is generally not equal to $p(h|e)/p(h|\bar{e})$.
3. Although it can easily be generalized to cases where there are many *independent* pieces of evidence, it is much harder to generalize to cases where the pieces of evidence not independent of each other.

A diagrammatic representation of the way to solve the first difficulty is shown in Figure 9.



where $w_{h\bar{e}} = \ln \frac{p(\bar{e}|h)}{p(\bar{e}|\bar{h})}$

Figure 9: How to implement the required effect of an *off* unit. The effect of $\bar{e}$ is implemented by putting it into the bias term for *h*, and by subtracting an equal amount from the weighting coefficient from *e*, so that when *e* is in the true state the effect of the bias term on *h* is cancelled out.

The combined weight from *e* in Figure 9 is:

$$w_{total} = w_{he} - w_{h\bar{e}}$$

$$= \ln \frac{p(e|h)}{p(e|\bar{h})} - \ln \frac{p(\bar{e}|h)}{p(\bar{e}|\bar{h})}$$

$$= \ln \frac{p(e \wedge h)\,[1 - p(e) - p(h) + p(e \wedge h)]}{[p(e) - p(e \wedge h)]\,[p(h) - p(e \wedge h)]}$$

This equation is symmetrical in e and h, so in solving the problem of how to make the negation of e have the correct effect on h we have also solved the second problem — the required weights are now symmetrical. The more complicated weight in the resultant equation does not alter the fact that the probability of a hypothesis has the form of the Boltzmann distribution for a unit with two energy states.

Systems which use Bayesian inference often make the assumption that pieces of evidence are independent. This assumption is motivated by the belief that it would be too difficult to discover all the dependencies and too expensive to store them even if they were known. Unfortunately, the independence assumption is hard to justify and it is typically a poor approximation in systems with many mutually interdependent hypotheses. This has led people working in the domain of medical diagnosis, for example, to reject Bayesian statistics in favor of more discrete, symbolic systems that use underlying causal models. Recently, however, Charniak (1983) has argued that the two approaches can be reconciled. It is possible to use Bayesian statistics and to do much better than the independence assumption by using a relatively small set of additional "causal factors" which capture the most significant non-independencies. These causal factors are analogous to our hidden units because they are not part of the definition of the task — which is to relate diseases to symptoms — but they are useful because they allow the higher-order structure of the relationships to be expressed.

The analogy can be taken further. Any practical medical diagnosis system that attempted to extend the applicability of Bayesian statistics by using underlying causal factors would probably get the appropriate factors from medical experts. It is tempting, however, to ask if there is an automatic procedure for deriving the factors from the data, or for fine-tuning the factors provided by experts. This is just what our learning algorithm does. Given some hidden units, it adjusts the strengths of their connections to the visible units and to each other so as to turn them into a useful set of "causal factors."

To summarize: The independence assumption keeps things simple, but it is likely to be wrong for complex problems. The full set of inter-dependencies (higher-order statistics) is much too large to acquire or to store. A good compromise is to focus on the most significant violations of independence and to use extra "causal"

factors to express them. The hidden units of a Boltzmann machine are just like these causal factors, and the learning algorithm can discover effective ways of using them.

10 Sequences

There is a serious obstacle which appears to prevent Boltzmann machines from modeling sequential symbol processing: At thermal equilibrium, there are no consistent sequences, and so a Boltzmann machine of the kind we have described is unable to produce sequential behavior. It can only respond to environmental changes, it cannot generate internal sequences.

It is tempting, when confronted with this problem to abandon the symmetry assumption and to use asymmetrical weights to encode sequential behavior. Sequences then correspond to continuous paths through phase-space. This is certainly a possibility (Hopfield, 1983), but it is incompatible with the central idea of performing constraint-satisfaction searches by relaxing to equilibrium under the influence of sustained input.[9] If *systematic* asymmetries are used to encode sequential information, the whole power of the search technique and of the resultant learning algorithm is destroyed.

What is at issue here is not just a minor modification. Systems with symmetric weights form a very interesting class of computational device because their dynamics is governed by an energy function.[10] This is what makes it possible to analyze their behavior and to use them for iterative constraint-satisfaction. They form an important natural class precisely because they *lack* the ability to go through regular sequences. In their influential exploration of Perceptrons, Minsky & Papert (1968; p 231) concluded that: "Multilayer machines with loops clearly open up all the questions of the general theory of automata." This statement is very plausible but recent developments suggest that it may be misleading because it ignores the symmetric case and it seems to have led to the general belief that it would be impossible to find an interesting generalization of perceptrons, and in particular a generalized perceptron convergence theorem for multi-layered networks.

An alternative to simply mixing in some asymmetric links in order to allow sequences is to separate out the symmetric and asymmetric components. A system could be composed of a number of internally symmetric modules that are asymmetrically connected to one another. Each module could then perform constraint-

---

[9] It is, of course, possible to *represent* sequences in a symmetric Boltzmann machine. If several different groups of units are dedicated to the different temporal pieces of a sequence, the states of these groups can represent the "fillers" of these temporal "slots," and the weights can be set so as to create energy minima for particular combinations of temporal slot fillers. It is then possible to recall the rest of a sequence from some fraction of it. However, this ability to recreate a *static* "spatial" representation of a sequence is quite different from the ability to proceed through the sequence. It does not require any temporal regularity in the internal states of the machine.

[10] One can easily write down a similar energy function for asymmetric networks, but this energy function does not govern the behavior of the network when the links are given their normal causal interpretation.

satisfaction searches, with the asymmetric inputs to the module acting as boundary conditions that determine the particular problem it has to solve at any moment. Given an architecture of this kind, it might be appropriate to use very different kinds of description at different time-scales. In the short term, the modules would perform parallel iterative constraint-satisfaction searches. In the longer term, the result of each search could be viewed as a single step in a strictly serial process, with each search setting up the boundary conditions for the next.

The idea of a coarse-grained, sequential description that is implemented by a series of parallel constraint-satisfaction searches seems to fit quite well with the idea of a production system architecture in which all the heavy computational work is done by a parallel "recognition process" that decides which rule best fits the current state of working memory.

One of the major difficulties in implementing this kind of matching is to ensure that there is a consistent set of bindings between the variables in a rule and the constants and variables of the instances in working memory. Discovering a consistent set of bindings is called unification and it requires a rather flexible matching process. Newell (1978) has suggested that this kind of flexible matching acts as a sequential bottleneck in people. Hinton (1981) describes a parallel network that is capable of performing the equivalent process in object-recognition.[11] This network has the interesting property that it can only settle on one match at a time. The settling process performs a large parallel search among alternative "rules" and "bindings" but the only way to ensure consistency of the bindings is to settle on a single match.

There are clearly many difficult problems in implementing production systems in a collection of Boltzmann machine modules, but we feel that this "discrete symbolic" approach may be more fruitful than trying to model sequences by continuous paths through state space, and it may help to explain how a collection of massively parallel, rather noisy modules can behave like a machine that manipulates discrete symbols according to formal rules.

11 The relationship to the brain

One of the main reasons for studying Boltzmann machines is that they bear some resemblances to brains. The hope is that by studying a simple and idealized machine that is in the same general class of computational device as the brain, we can gain insight into the principles that underlie biological computation (especially the kinds of computation that occur in mammalian neocortex). This is clearly a vain hope if there are

[11] In object recognition the problem is to find the viewpoint-independent object-model that best fits the current collection of viewpoint-dependent features. The object-models are like rules and the viewer-centered features are like instances in working memory. The viewing transform that relates viewer-centered and object-centered features is like the set of variable bindings.

irreconcilable differences between the cortex and Boltzmann machines that are crucial to the way Boltzmann machines compute. In this section we discuss some of the most obvious differences.

11.1 Binary states and action potentials

Neurons are complex biochemical entities and the simple binary units studied here are not meant to be literal models of cortical neurons. However, two of our key assumptions, that the binary units change state asynchronously and that they use a probabilistic decision rule, seem closer to reality than a model with synchronous, deterministic updating.

The energy gap for a binary unit has a role similar to that played by the membrane potential for a neuron: both are the sum of the excitatory and inhibitory inputs and both are used to determine the output state. However, neurons produce action potentials — brief spikes that propagate down axons — rather than binary outputs. When an action potential reaches a synapse, the signal it produces in the postsynaptic neuron rises to a maximum and then exponentially decays with the time constant of the membrane (typically around 5 msec for neurons in cerebral cortex). The effect of a single spike on the postsynaptic cell body may be further broadened by electrotonic transmission down the dendrite to the cell body.

The energy gap represents the summed input from all the recently active binary units. If the average time between updates is identified with the average duration of a postsynaptic potential then the binary pulse between updates can be considered an approximation to the postsynaptic potential. Although the shape of a single binary pulse differs significantly from a postsynaptic potential, the sum of a large number of stochastic pulses is independent of the shape of the individual pulses and depends only on their amplitudes and durations. Thus for large networks having the large fan-ins typical of cerebral cortex (around 10,000) the binary approximation may not be too bad.

11.2 Implementing temperature in neurons

What significance could the probabilistic decision rule in Eq. (5) have for neurons, and in particular, what does the temperature correspond to and how can it be controlled? The membrane potential of a neuron is graded, but if it exceeds a fairly sharp threshold an action potential is produced followed by a refractory period lasting several msec during which another action potential cannot be elicited. If Gaussian noise is added to the membrane potential, then even if the total synaptic input is below threshold, there is a finite probability that the membrane potential will reach threshold.

The amplitude of the Gaussian noise will determine the width of the sigmoidal probability distribution for the neuron to fire during a short time interval, and it therefore plays the role of temperature in the model.

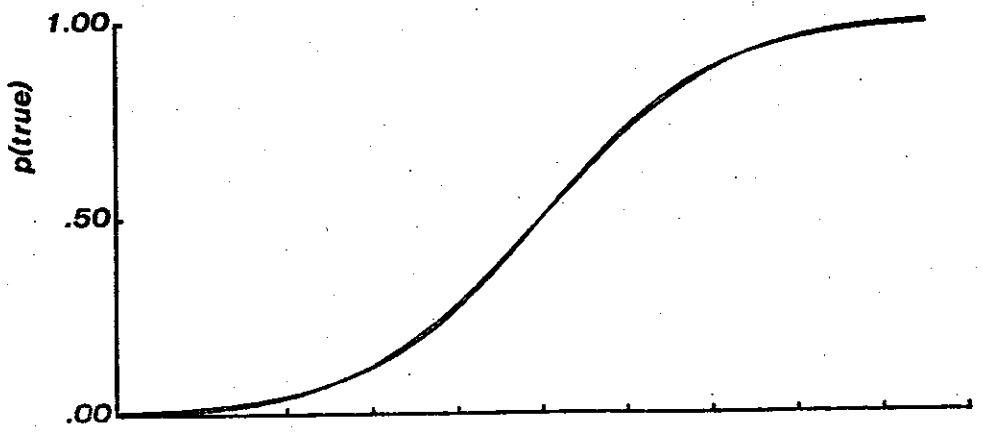


Figure 10: Two superimposed curves. One is the sigmoid function of Figure 1. The other is the cumulative area under a Gaussian, which gives the probability that the threshold of a unit will be exceeded by the sum of its energy gap and some Gaussian noise with mean 0. If the standard deviation of the Gaussian is chosen appropriately, the two curves never differ by more than about 1%.

Surprisingly, a cumulative Gaussian is a very good approximation to the required probability distribution (Figure 10). Intracellular recordings from neurons show that there is stochastic variability in the membrane potential of most neurons, in part due to fluctuations in the transmitter released by presynaptic terminals. Other sources of noise may also be present and could be controlled by cellular mechanisms.

In the visual cortex of primates single neurons respond to the same visual stimulus with different sequences of action potentials on each trial (Sejnowski, 1981). In order to measure a repeatable response the spike trains are typically averaged over 10 trials. The result, called the post-stimulus time histogram, gives the probability for a spike to occur as a function of the time after the onset of the stimulus. However, this averaging procedure throws out all information about the variance of the noise, so that there is no way to determine whether the noise varies systematically during the response to the stimulus or perhaps on a longer time scale while the stimulus is being attended.

Unlike the bulk of the brain, which is composed of many morphologically different nuclei, the cerebral cortex

is relatively uniform in structure. Different areas of cerbral cortex are specialized for processing information from different sensory modalities, such as visual cortex, auditory cortex, and somatosensory cortex, and other areas are specialized for motor functions; however, all of these cortical areas have a similar anatomical organization and are more similar to each other in cytoarchitecture than they are to any other part of the brain. Many problems in vision, speech recognition, associative recall, and motor control can be formulated as searches. The similarity between different areas of cerebral cortex suggests that the same kind of massively parallel searches may be performed in many different cortical areas.

It may be difficult to arrange for all neurons in a network to have the same amplitude of noise. How sensitive are our statistical results to the assumption that all binary units are making decisions at the same temperature? The effect of variations in the temperature was tested in large-scale simulations of a problem reported in (Sejnowski, Hinton, Kienker, & Schumacher, 1984). Random variations in the temperature from unit to unit of up to 25% did not significantly affect the annealing and equilibrium solutions. The effect of variations in temperature on learning has not been studied.

11.3 Asymmetry and time-delays

In a Boltzmann Machine of the kind we have presented all connections are symmetrical. It is very unlikely that this assumption is strictly true of neurons in cerebral cortex. However, if the constraints of a problem are inherently symmetrical and if the network on average approximates the required symmetrical connectivity, then random asymmetries in a large network will be reflected as an increase in the Gaussian noise in each unit. Systematic asymmetries may have other purposes. For example, some interneurons are thought to have only inhibitory links to other neurons, and these could serve as automatic gain controls to keep a network within a narrow operating range of firing rates.

To see why random asymmetry acts like gaussian noise, consider a symmetrical network in which pairs of units are linked by two equal one-way connections, one in each direction. Now, perform the following operation on all pairs of these one-way connections: Remove one of the connections and double the strength of the other. Provided the choice of which connection to remove is made randomly, this operation will not alter the *expected* value of the input to a unit from the other units. On average, it will "see" half as many other units, but with twice the weight. So if a unit has a large fan-in it will be able to make a good unbiased estimate of what its total input would have been if the links had not been cut. However, the use of fewer, larger weights will increase the variance of the energy gap, and will thus act like added noise.

The idea that a large fan-in is needed to reduce the effects of random asymmetries has an interesting consequence for artificially produced systems of this kind. There may be a trade-off between symmetry and fan-in. In systems like the brain where connections are *grown* it may be hard to ensure symmetry, so a large

fan-in may be essential (Cortical pyramidal cells typically receive between 10^3 and 10^5 input connections). In artificial systems it may be hard to achieve very high fan-ins, but this may not be essential provided the connections are symmetrical.

The analysis of the effects of time-delays is somewhat more complex, but simulations suggest that time-delays act like added noise, and preliminary mathematical results (Venkataraman Venkatasubramanian, personal communication) show that this is true to first order provided the fan-in is large and the individual weights are small compared with the energy gaps.

The main difficulty in the treatment of both asymmetry and time-delays is that it is hard to know how they will interact with the learning algorithm. It is quite possible, for example, that a network which starts with random asymmetry will develop systematic asymmetry.

12 Conclusion

The application of statistical mechanics to constraint-satisfaction searches raises a great many issues that we have only mentioned in passing. Some of these issues are discussed in greater detail elsewhere: Hinton and Sejnowski (1983) describe the relation to more conventional relaxation techniques; Fahlman, Hinton and Sejnowski (1983) compare Boltzmann machines with some alternative parallel schemes, and discuss some of the knowledge representation issues; Geman and Geman (1983) describe a very similar model, developed independently, and discuss its relationship to Markov random fields.

The two main ideas that led to the Boltzmann Machine are that noise can help with search, and that Boltzmann distributions make it possible to assign credit on the basis of *local* information in a non-linear network. It is interesting that a similar approach can be derived from entirely different considerations. While investigating how to perform computation reliably with unreliable components, Von Neumann was led to the following conclusion:

> All of this will lead to theories [of computation] which are much less rigidly of an all-or-none nature than past and present formal logic. They will be of a much less combinatorial, and much more analytical, character. In fact, there are numerous indications to make us believe that this new system of formal logic will move closer to another discipline which has been little linked in the past with logic. This is thermodynamics, primarily in the form it was received from Boltzmann, and is that part of theoretical physics which comes nearest in some of its aspects to manipulating and measuring information.
>
> — John Von Neumann, *Collected Works* Vol 5, pg 304

References

Berliner, H.J., & Ackley, D.H. The QBKG system: Generating explanations from a non-discrete knowledge representation. *Proceedings of the National Conference on Artificial Intelligence AAAI-82,* Pittsburgh, PA, August 1982, 213-216.

Binder, K. (Ed.) *The Monte-Carlo Method in Statistical Physics.* New York: Springer-Verlag, 1978.

Charniak, E. The Bayesian basis of common sense medical diagnosis. *Proceedings of the National Conference on Artificial Intelligence AAAI-83,* Washington, DC, August 1983, 70-73.

Derthick, M. Learning in Boltzmann Machines and why it's slow. Technical report CMU-CS-84-120, Pittsburgh, PA: Carnegie-Mellon University, 1982.

Fahlman, S.E. The Hashnet Interconnection Scheme. Technical report CMU-CS-80-125, Carnegie-Mellon University, June 1980.

Fahlman, S.E., Hinton, G.E., & Sejnowski, T.J. Massively parallel architectures for AI: NETL, Thistle, and Boltzmann Machines. *Proceedings of the National Conference on Artificial Intelligence AAAI-83,* Washington, DC, August 1983, 109-113.

Feldman, J.A. A connectionist model of visual memory. In G.E. Hinton & J.A. Anderson (Eds.) *Parallel Models of Associative Memory.* Hillsdale, NJ: Erlbaum, 1981.

Feldman, J.A., & Ballard, D.H. Connectionist models and their properties. *Cognitive Science,* 1982, *6,* 205-254.

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. Unpublished manuscript, 1983.

Grimson, W.E.L. *From Images to Surfaces.* Cambridge, MA: MIT Press, 1981.

Hinton, G.E. Relaxation and its role it vision. PhD Thesis, University of Edinburgh, 1977; Described in: *Computer Vision,* D.H. Ballard and C.M. Brown (Eds.) Englewood Cliffs, NJ: Prentice-Hall, 1982, 408-430.

Hinton, G.E. Implementing semantic networks in parallel hardware. In G.E. Hinton & J.A. Anderson (Eds.) *Parallel Models of Associative Memory.* Hillsdale, NJ: Erlbaum, 1981a.

Hinton, G. E. A parallel computation that assigns canonical object-based frames of reference. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence,* Vol 2. Vancouver BC, Canada. August 1981b.

Hinton, G.E., & Anderson, J.A. *Parallel Models of Associative Memory.* Hillsdale, NJ: Erlbaum, 1981.

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

Hinton, G.E., & Sejnowski, T.J. Optimal perceptual inference. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition.* Washington, DC, June 1983, 448-453. (b)

Hinton, G.E., Sejnowski, T.J., & Ackley, D.H. Boltzmann Machines: Constraint satisfaction networks that learn. Technical report CMU-CS-84-119, Carnegie-Mellon University, May 1984.

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

Hopfield, J.J. Collective processing and neural states. To appear in: *Modeling and analysis in Biomedicine.*

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.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A. & Teller, E. *Journal of Chemical Physics*, 1953, *6*, 1087.

Minsky, M. L. A framework for representing knowledge. In *The Psychology of Computer Vision*, P. Winston (Ed.) NY: McGraw-Hill, 1975.

Minsky, M., & Papert, S. *Perceptrons.* Cambridge, MA: MIT Press, 1968.

Newell, A. Harpy, Production Systems and Human Cognition. Technical report CMU-CS-78-140, Pittsburgh, PA: Carnegie-Mellon University, 1978.

Newell, A. Intellectual issues in the history of artificial intelligence. Technical report CMU-CS-82-142, Pittsburgh, PA: Carnegie-Mellon University, 1982.

Newell, A. & Simon, H.A. *Human Problem Solving.* Englewood Cliffs, NJ: Prentice-Hall, 1972.

Ratcliff, R. A theory of memory retrieval. *Psychological Review*, 1978, *85*(2), 59-108.

Renyi, A. *Probability Theory.* Amsterdam: North-Holland, 1962.

Rosenblatt, F. *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms.* Washington, DC: Spartan, 1961.

Rumelhart, D. E. and Zipser, D. Competitive Learning. *Cognitive Science*, (in press).

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

Sejnowski, T. J., Skeleton filters in the brain. In: G. E. Hinton & J. A. Anderson (Eds.) *Parallel Models of Associative Memory.* Hillsdale, NJ: Erlbaum, 1981.

Smolensky, P., Schema selection and stochastic inference in modular environments. *Proceedings of the National Conference on Artificial Intelligence AAAI-83*, Washington, DC, August 1983, 109-113.

Terzopoulos, D., Multiresolution computation of visible-surface representations. PhD. Thesis, Cambridge, MA: MIT, 1984.

Waltz, D.L. Understanding line drawings of scenes with shadows. In *The Psychology of Computer Vision*, P. Winston (Ed.) NY: McGraw-Hill, 1975.

Winston, P.H., *Artificial Intelligence.* (2nd edition) Reading, MA: Addison-Wesley, 1984.

Appendix: Derivation of the learning algorithm

When a network is free-running at equilibrium the probability distribution over the visible units is given by

$$P'(V_\alpha) = \sum_\beta P'(V_\alpha \wedge H_\beta) = \frac{\sum_\beta e^{-E_{\alpha\beta}/T}}{\sum_{\lambda\mu} e^{-E_{\lambda\mu}/T}} \tag{11}$$

where V_α is a vector of states of the visible units, H_β is a vector of states of the hidden units, and $E_{\alpha\beta}$ is the energy of the system in state $V_\alpha \wedge H_\beta$

$$E_{\alpha\beta} = -\sum_{i<j} w_{ij} s_i^{\alpha\beta} s_j^{\alpha\beta}$$

Hence,

$$\frac{\partial e^{-E_{\alpha\beta}/T}}{\partial w_{ij}} = \frac{1}{T} s_i^{\alpha\beta} s_j^{\alpha\beta} e^{-E_{\alpha\beta}/T}$$

Differentiating (11) then yields

$$\frac{\partial P'(V_\alpha)}{\partial w_{ij}} = \frac{\frac{1}{T}\sum_\beta e^{-E_{\alpha\beta}/T} s_i^{\alpha\beta} s_j^{\alpha\beta}}{\sum_{\lambda\mu} e^{-E_{\lambda\mu}/T}} - \frac{\sum_\beta e^{-E\alpha\beta/T} \frac{1}{T} \sum_{\lambda\mu} e^{-E_{\lambda\mu}/T} s_i^{\lambda\mu} s_j^{\lambda\mu}}{\left(\sum_{\lambda\mu} e^{-E_{\lambda\mu}/T}\right)^2}$$

$$= \frac{1}{T}\left[\sum_\beta P'(V_\alpha \wedge H_\beta) s_i^{\alpha\beta} s_j^{\alpha\beta} - P'(V_\alpha) \sum_{\lambda\mu} P'(V_\lambda \wedge H_\mu) s_i^{\lambda\mu} s_j^{\lambda\mu}\right]$$

This derivative is used to compute the gradient of the G-measure

$$G = \sum_\alpha P(V_\alpha) \ln \frac{P(V_\alpha)}{P'(V_\alpha)}$$

where $P(V_\alpha)$ is the clamped probability distribution over the visible units and is independent of w_{ij}. So

$$\frac{\partial G}{\partial w_{ij}} = -\sum_\alpha \frac{P(V_\alpha)}{P'(V_\alpha)} \frac{\partial P'(V_\alpha)}{\partial w_{ij}}$$

$$= -\frac{1}{T}\sum_\alpha \frac{P(V_\alpha)}{P'(V_\alpha)}\left[\sum_\beta P'(V_\alpha \wedge H_\beta) s_i^{\alpha\beta} s_j^{\alpha\beta} - P'(V_\alpha) \sum_{\lambda\mu} P'(V_\lambda \wedge H_\mu) s_i^{\lambda\mu} s_j^{\lambda\mu}\right]$$

Now,

$$P(V_\alpha \wedge H_\beta) = P(H_\beta|V_\alpha) P(V_\alpha),$$
$$P'(V_\alpha \wedge H_\beta) = P'(H_\beta|V_\alpha) P'(V_\alpha),$$

and

$$P'(H_\beta|V_\alpha) = P(H_\beta|V_\alpha) \tag{12}$$

Equation (12) holds because the probability of a hidden state given some visible state must be the same in equilibrium whether the visible units were clamped in that state or arrived there by free-running. Hence,

$$P'(V_\alpha \wedge H_\beta)\frac{P(V_\alpha)}{P'(V_\alpha)} = P(V_\alpha \wedge H_\beta)$$

Also,

$$\sum_\alpha P(V_\alpha) = 1$$

Therefore,

$$\frac{\partial G}{\partial w_{ij}} = -\frac{1}{T}[p_{ij} - p'_{ij}]$$

where

$$p_{ij} \stackrel{\text{def}}{=} \sum_{\alpha\beta} P(V_\alpha \wedge H_\beta) s_i^{\alpha\beta} s_j^{\alpha\beta}$$

and

$$p'_{ij} \stackrel{\text{def}}{=} \sum_{\lambda\mu} P'(V_\lambda \wedge H_\mu) s_i^{\lambda\mu} s_j^{\lambda\mu}$$

as given in (9).

The Boltzmann Machine learning algorithm can also be formulated as an input-output model. The visible units are divided into an input set I and an output set O, and an environment specifies a set of conditional probabilities of the form $P(O_\beta|I_\alpha)$. During the "training" phase the environment clamps both the input and output units, and p_{ij}s are estimated. During the "testing" phase the input units are clamped and the output units and hidden units free-run, and p'_{ij}s are estimated. The appropriate G measure in this case is

$$G = \sum_{\alpha\beta} P(I_\alpha \wedge O_\beta) \ln \frac{P(O_\beta|I_\alpha)}{P'(O_\beta|I_\alpha)}$$

Similar mathematics apply in this formulation and $\partial G/\partial w_{ij}$ is the same as before.