Physica 22D (1986) 260–275
North-Holland, Amsterdam

LEARNING SYMMETRY GROUPS WITH HIDDEN UNITS: BEYOND THE PERCEPTRON

Terrence J. SEJNOWSKI and Paul K. KIENKER
Biophysics Department, Johns Hopkins University, Baltimore, MD 21218, USA

and

Geoffrey E. HINTON
Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA 15213, USA

Learning to recognize mirror, rotational and translational symmetries is a difficult problem for massively-parallel network models. These symmetries cannot be learned by first-order perceptrons or Hopfield networks, which have no means for incorporating additional adaptive units that are hidden from the input and output layers. We demonstrate that the Boltzmann learning algorithm is capable of finding sets of weights which turn hidden units into useful higher-order feature detectors capable of solving symmetry problems.

1. Introduction

Interest in massively-parallel network models has recently increased because of the possibility that they may be capable of solving difficult problems in vision, speech and motor control [1–4]. Knowledge about the task domain can be "programmed" into a network by specifying the connectivity of the processing elements and the strengths of the connections, or weights. For example, networks have been designed to perform figure-ground separation in images [5] and to solve the traveling salesman problem [6]. It would be desirable to have an automatic learning procedure that could incorporate knowledge about the task domain into the weights given only examples of solutions. The network would then be able to generalize appropriately.

Learning in humans is not a single ability but a cluster of adaptive abilities. When we are introduced to someone we learn to associate their name with their face and voice. Faces, names and voices are domains in which we are already expert, and only the associations amongst them need be learned. It is more difficult to remember someone's name in a foreign country, where the faces, names and voices are unfamiliar. It is even more difficult to learn new domains, such as lip reading or interpreting medical X-ray photographs. Learning a new domain means learning new concepts and new internal representations, learning that normally requires long training.

The learning problems examined here are based on the categorization of symmetry groups in patterns presented to an input array. It is possible in this domain to classify the difficulty of problems and to get insight into possible solutions using our own knowledge of symmetries. Symmetries are commonly found in nature, such as the bilateral mirror symmetry of faces, and symmetry groups have been extensively studied in mathematics and physics. We will demonstrate a general learning algorithm that can discover mirror, rotational and translational symmetries in input patterns. These empirical results support the conjecture that the predicate order of a problem is a measure of its learnability [7, 8, 10].

0167-2789/86/$03.50

2. Perceptrons

The perceptron is a parallel machine that can learn to categorize input patterns [9]. The learning algorithm requires a training phase during which the teacher presents examples from each category and provides feedback about the correctness of the classification that the perceptron makes. The input patterns are normally represented by an array of units that can take on only binary values. The input units are connected to a set of feature analyzers that are then connected to a final set of decision units. Each decision unit computes the scalar product of the vector of outputs from the feature analyzers and the vector of weights on its incoming connections. The output of a decision unit is 1 if the scalar product exceeds a threshold, otherwise it is 0, as shown in fig. 1.

The feature analyzers are fixed in advance and do not learn. All the learning takes place in the weights from the feature analyzers to the decision units, so when a perceptron is viewed as a *learning device* the states of the feature analyzers can be viewed as a pre-processed input vector and the inputs are then directly connected to the outputs. This simple topology and simple decision rule has the advantage that there is a simple procedure for incrementally changing the weights which is guaranteed to converge on a set of weights that allows the decision units to correctly categorize the input patterns, *provided that such a set of weights exists*. Unfortunately, there are strong limitations on what can be learned by a single layer of modifiable weights [10]. However, these limitations can be overcome in several ways if we:

1) Use exponentially many feature analyzers so that all possible combinations of activity in the input array are explicitly represented in the feature analyzers.

2) Make the decision units use a more complex function for combining the outputs of the feature analyzers.

3) Extend the algorithm so that it also modifies the weights on the connections between the input array and the feature analyzers, so that the network can develop just those feature analyzers that are required for the task. This can also be viewed as introducing extra layers of adaptive units between the feature analyzers and the decision units if the input patterns are equated with the states of the feature analyzers rather than the input array of the perceptron.

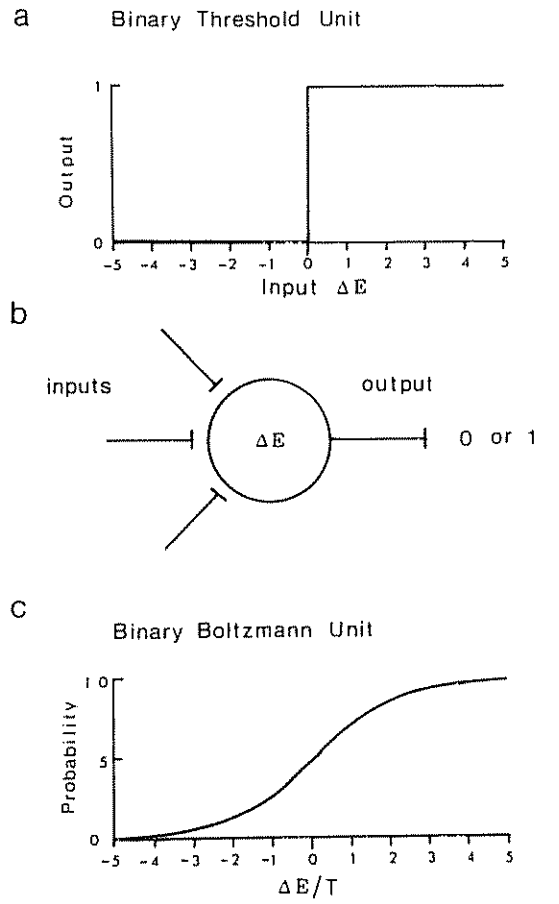

Fig. 1. Input–output decision rules for a) a binary threshold unit in a perceptron; and c) the probability of activating a binary Boltzmann unit as a function of total input ΔE, shown schematically in b). The temperature T is a scaling factor that determines the steepness of the sigmoid as a function of ΔE. Note that in the limit as $T \to 0$, the probability sigmoid in c) approaches the step function in a).

A key concept that Minsky and Papert [10] introduced in their influential analysis of perceptrons is the *order* of the problem, which roughly corresponds to the minimum number of input

units that carry any information relevant to the required output. (A more formal definition of predicate order is given in appendix A.) For example, the boolean OR function has order 1 because the value of a single input unit contains some information about the output. In contrast, the boolean EXCLUSIVE-OR (XOR) function has order 2 because each unit of the input array, considered in isolation, contains no information about the output value. The parity function of n binary input units is order n, and XOR is the special case with $n = 2$ (see section on XOR below).

Perceptrons are only capable of solving problems that are first order in their feature analyzers. They cannot, for example, learn to solve the XOR problem unless they are given appropriate hand-wired feature analyzers. This is also true of many other types of networks and learning algorithms that have been proposed, such as the network proposed by Hopfield [11] for associative memory. It can be proved that there exists no set of weights and thresholds for three binary units that allows a three-unit Hopfield network to store the four patterns (1 1 0), (1 0 1), (0 1 1), and (0 0 0) without storing other additional patterns. Higher-order problems can, however, be solved by introducing additional units that we call hidden units because they receive no external inputs and have no external outputs. As will be demonstrated, the XOR problem is a relatively easy second-order problem that can be solved if one hidden unit is added to the network.

Networks like the perceptron in which there is only one layer of modifiable weights avoid the most difficult issue in learning – constructing appropriate internal representations. In a perceptron, *all* the units have their behavior specified externally. The behavior of the feature analyzers is a pre-wired function of the input patterns, and the required behavior of the decision units is specified by the teacher. This means that there are no "hidden" units that are free to represent novel features that would help with the required discrimination. As soon as hidden units are introduced, learning becomes much harder because the learning procedure must decide what the hidden units should represent. Minsky and Papert suggested that the limitation of the perceptron learning algorithm to a single layer of modifiable weights could not be overcome. We show here that at least one generalization of the perceptron can learn to use hidden units to solve difficult problems.

3. Boltzmann machines

Boltzmann machines were designed to solve a class of optimization problems in vision called constraint-satisfaction problems [3, 12, 13]. The processing units in the machine have binary values like that of the perceptron, but all links between the units have reciprocal connections with the same weight. The symmetric connectivity enables the convergence of the networks to be analyzed using techniques from physics [11]. In particular, it is possible to define the energy of the network as

$$E = -\tfrac{1}{2}\sum_{i}\sum_{j\neq i} w_{ij}s_i s_j, \tag{1}$$

where s_i is the state of the ith binary unit and w_{ij} is the weight between the ith and jth units. (In this paper the weights can only take on integer values.) The binary threshold rule governing the binary units in the perceptron is deterministic, but in a Boltzmann machine the decision rule is probabilistic:

$$p_i = P(\Delta E_i) = \frac{1}{1 + e^{-\Delta E_i/T}}, \tag{2}$$

where p_i is the probability for the ith unit to be in state 1, $P(x)$ is a sigmoidal probability function (fig. 1), T is a parameter analogous to temperature and is a measure of the noise introduced into the decision [14], and the total input to the unit is

$$\Delta E_i = \sum_j w_{ij}s_j. \tag{3}$$

Thresholds are taken into account by giving each

unit a connection to a unit that always has value 1, the "true" unit. The value of the weight to the true unit is a bias, or the negative of the threshold, to that unit.

The energy function of a network is a cost function that must be minimized to solve the constraint-satisfaction problem. The probabilistic decision rule, which is applied asynchronously, allows the network to escape from local energy minima. Simulated annealing (gradually lowering the temperature T) can be used to find good states of low energy [15]. In this paper simulated annealing will not be used to find the globally optimal solution, but to rapidly reach equilibrium at the temperature where learning takes place. In all the simulations reported here the learning temperature was chosen arbitrarily to be $T = 10$.

The idea of implementing constraints as interactions between stochastic processing elements was proposed by Moussouris [16] who discussed the identity between Boltzmann distributions and Markov random fields [17]. The idea of using simulated annealing to find low energy states in parallel networks has been investigated by several different groups. Geman and Geman [18] established limits on the allowable speed of the annealing schedule and showed that simulated annealing can be very effective for removing noise from images. Hinton and Sejnowski [3] showed how the use of binary stochastic elements could solve some problems with other relaxation techniques, in particular the problem of learning the weights. Smolensky [19] has been investigating a similar framework he calls "harmony theory", and Cerny [20] has applied simulated annealing to multiprocessors. Paretto [21], Amit et al. [22, 23], and Toulouse et al. [24] have studied the properties of networks that have random weights.

4. The Boltzmann learning algorithm

The goal of the learning algorithm is to produce a network which correctly categorizes input patterns according to the following procedure: the input units are "clamped" to a particular pattern while the network relaxes into a state of low energy in which the output units have the correct values. The input and output units will be called "visible" units to distinguish them from the hidden units. The difficult part of learning is to decide how to use the hidden units to help achieve the required behavior of the visible units. The statistical structure of the input–output mapping must be captured by the weights between the units, and only the first-order statistics of the input pattern can be captured by direct connections between input and output units. The role of the hidden units is to capture higher-order statistical relationships and this can be accomplished if significant underlying features can be found that have strong, regular relationships with the patterns on the visible units. The hard part of learning is to find the sets of weights which turn the hidden units into useful feature detectors.

The Boltzmann learning algorithm is closely related to the EM algorithm for estimating the parameters of exponential distributions to fit incomplete data [25] and an earlier algorithm proposed by Baum for estimating the parameters in a hidden Markov chain that has been successfully used for speech recognition [26]. These are maximum likelihood methods that work by adjusting parameters to increase the probability that the observed data were generated by the underlying model. The Boltzmann learning algorithm, which is also a maximum likelihood method, has the advantage that it can be easily implemented in a parallel network of simple processing units. It is derived from a measure of how effectively the weights in a network are being used for modeling the structure of the environment [3, 13]. In this paper the environment will always have the structure of a mapping between a set of input units and a set of output units. Each step in the algorithm depends on estimating the gradient of this measure with respect to the weights and making incremental changes to the weights to reduce the discrepancy between the model and the environment.

We assume that the environment clamps a particular vector over the input units and keeps it there long enough for the network to reach thermal equilibrium with this vector as a boundary condition. We also assume that there is no structure in the sequential order of the environmentally clamped vectors. Therefore the complete structure of the ensemble of input–output mappings can be specified by giving the probability distribution of each of the 2^v vectors over the v visible units. Note that this probability distribution does not depend on the weights in the network because the environment clamps the visible units.

A particular set of weights can be said to constitute a perfect model of the structure of the environment if it leads to exactly the same probability distribution of visible output vectors when the visible input vectors are clamped. Let $P^-(O_\alpha|I_\beta)$ be the probability distribution of obtaining an output state O_α when the input is clamped to state I_β and the network has been allowed to reach equilibrium. The goal of the learning algorithm is to find weights which match the desired input–output mapping $P^+(O_\alpha|I_\beta)$ to the actual one, $P^-(O_\alpha|I_\beta)$, but this will not in general be possible since there are exponentially many possible mappings and there are at most only $n(n-1)/2$ symmetrical weights and n thresholds among n units. However, it may be possible to do well with relatively few weights if the mappings contain regularities that can be expressed in the weights.

An information-theoretic measure [27] of the distance between the two probability distributions is given by

$$G = \sum_\alpha \sum_\beta P^+(O_\alpha|I_\beta) P(I_\beta) \ln \frac{P^+(O_\alpha|I_\beta)}{P^-(O_\alpha|I_\beta)}, \qquad (4)$$

where $P(I_\beta)$ is the probability distribution over input vectors. G is never negative and is zero only if the desired and actual probability distributions of the mappings are identical. The G-measure is sometimes called the asymmetric divergence or information gain because it reflects the distance *from* the output probabilities produced by the model *to* the environmental distribution. Minimizing G is also equivalent to maximizing the log of the likelihood of generating the correct mappings. The likelihood ratio is weighted by the actual probability of occurrence of that mapping, which gives greater weight to events that occur more frequently.

The network's model of the required mapping can be improved by changing the weights so as to reduce G. At first sight this would seem to require global information about the network, since changing a single weight will change the probabilities of many global states in ways that depend on all the other weights in the network. Fortunately, this information is available locally; that is, the gradient of G with respect to w_{ij} depends only on the behavior of the ith and jth units. It can be shown that

$$\frac{\partial G}{\partial w_{ij}} = -\frac{1}{T}\left[P_{ij}^+ - P_{ij}^-\right], \qquad (5)$$

where P_{ij}^+ is the probability, averaged over all mappings, that the ith and jth units are both on when the inputs and outputs of the network are clamped, and P_{ij}^- is the corresponding co-occurrence probability when only the inputs to the network are clamped [13].

Because the system is Markov and therefore ergodic, the ensemble-average probabilities in eq. (5) can be approximated by time-average probabilities in thermal equilibrium. A fast annealing schedule was used to reach approximate equilibrium: (2@40, 2@35, 2@30, 2@25, 2@20, 2@16, 2@14, 2@12, 14@10), where 2@40 means that each of the units was probed twice at a temperature of 40. Co-occurrence statistics were accumulated during the last 5@10 iterations and the weights were updated after several input patterns were presented (typically 5). Several different ways to collect the statistics are discussed in appendix B. The difference

$$\Delta = P_{ij}^+ - P_{ij}^- \qquad (6)$$

can be used to estimate the gradient of G, but because the average is very inaccurate, only the sign was used to decide to change the weight by $+1$ if $\Delta > 0$ and -1 if $\Delta < 0$. Using this gradient descent procedure the weights tended to become large because that is one way to improve the performance on patterns already learned. Unfortunately, large weights cause high energy barriers so that the network is unable to approach equilibrium in the time allowed. To counteract this tendency, the weights were made to decay by 1 with a small probability that was proportional to the magnitude of the weight (typically around 0.0001 times the magnitude of the weight).

Except in the XOR problem, the input units were not connected directly to the output units but communicated only indirectly through a layer of hidden units. The output units were interconnected but there were no interconnections amongst the hidden units. Allowing connections between hidden units did not significantly affect the performance, but made it more difficult to interpret the solutions.

5. Unlearning

Crick and Mitchison [28] have suggested that a form of reverse learning might occur during REM sleep in mammals. A simulation of reverse learning was performed by Hopfield et al. [29] who independently had been studying ways to improve the associative storage capacity of simple networks of binary processors. In their algorithm an input is presented to the network as an initial condition and the system evolves by falling into a nearby energy minimum. However, not all local energy minima represent stored information. In creating the desired minima, they accidentally create other spurious minima and to eliminate these they use "unlearning": The learning procedure is applied with reverse sign to the states after starting from random initial conditions. Following this procedure the performance of the system in accessing states was found to be improved.

There is an interesting relationship between the reverse learning proposed by Crick and Mitchison and Hopfield et al. and the form of the learning algorithm which we derived by considering how to minimize an information theoretic measure of the discrepancy between the environmental structure and the network's internal model [3]. The two phases of our learning algorithm resemble the learning and unlearning procedures: positive Hebbian learning occurs in phase$^+$ during which information in the environment is captured by the weights; during the testing phase$^-$ the system samples states according to their Boltzmann distribution and Hebbian learning occurs with a negative coefficient.

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

6. XOR

The XOR problem will be used to illustrate several general properties of solutions produced by the Boltzmann learning algorithm. This problem is small enough that it can be run many times and all possible solutions can be catalogued. The average learning curve for 20 runs is shown in fig. 2. Two measures were used to assess the performance of a network. For the fast cooling schedule given above

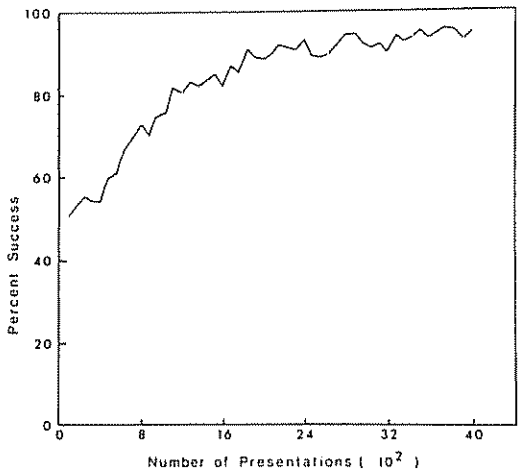

Fig. 2. Performance of the XOR network, averaged over 20 runs, as a function of the number of individual input presentations. The performance with a fast annealing schedule reached 96% within 2,000 presentations. With a slower annealing schedule the same weights gave a virtually perfect performance.

the typical performance was 96% correct at $T = 10$; when a much slower cooling schedule was used (255 iterations down to $T = 4$) the performance was virtually perfect. (A Boltzmann machine cannot be perfect at finite temperature.)

Two graphical ways to represent a network are illustrated in fig. 3. In the first, more traditional form each reciprocal connection of the network is represented as an arc and the value of the weight is represented by a number on the arc. The same network is shown in a form where each weight is represented as a black or a white square, and the connection is implicitly indicated by the position of the square within the unit icon. In the example shown, the hidden unit, in the center, has two excitatory weights to the input units, which can be considered the "receptive field" of the unit. The hidden unit in turn excites the output unit. However, this was not the only way that the hidden unit can be used to solve the problem. Eight different classes of solution were found. Some of these solutions were more common than others, as indicated in table I. All of the runs were started from zero weights, which biased the search in weight space toward some classes of solutions.

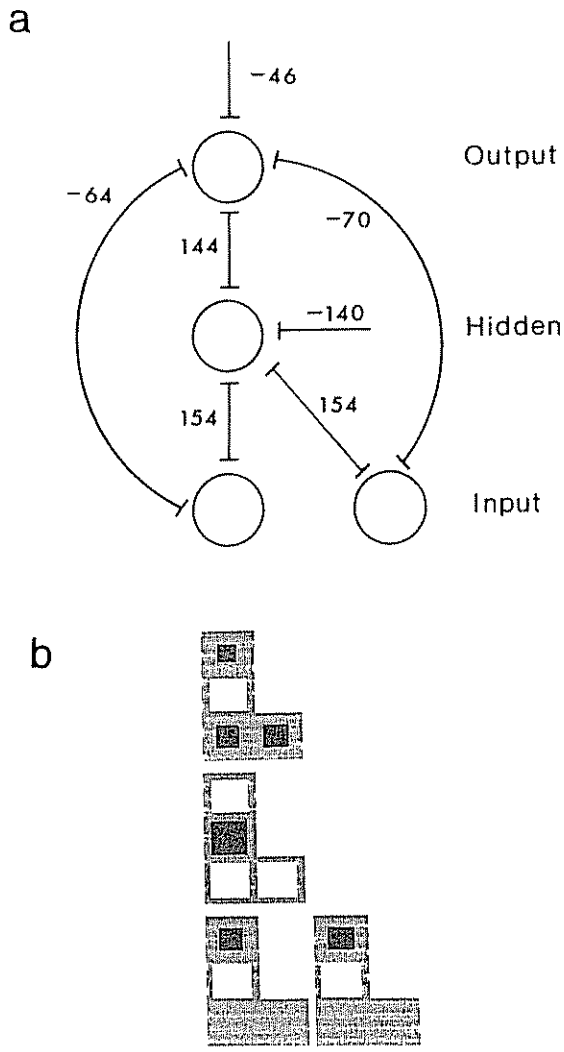

Fig. 3. Two ways of representing a network that solves the XOR problem as found by the Boltzmann learning algorithm. a) Conventional representation: The circles represent the two input units in the bottom layer, the output unit in the top layer, and the hidden unit in the middle layer. The lines between the units are labeled with the weight of the reciprocal link between each pair of units. The single line drawn to a unit is a link to the true unit, and the value represents the bias to that unit. b) A more compact self-similar representation of the same network as in a). Each unit has the same "L" shape as the shape formed by all four units. The squares within each unit represent weights, and the position of a square within the unit matches the position within the network of the unit to which the weight connects. White squares are excitatory weights, black squares are inhibitory weights, and the area of the square is proportional to the magnitude of the weight. The bias to a unit is shown in the position occupied by that unit in the network.

Table I
Summary of eight different ways that a single hidden unit can be used in an XOR network. The desired input–output mapping is shown on the left side of the table and each column on the right side shows the state of the hidden unit for one solution to the problem. The Boltzmann learning algorithm was run 339 times starting each run with zero weights. Each of the codes was found at least once and the fraction of the total number of runs of each type is listed beneath each column

Inputs		Output	Hidden unit							
0	0	0	0	1	0	1	0	1	0	1
0	1	1	1	0	1	0	0	1	0	1
1	0	1	1	0	0	1	1	0	0	1
1	1	0	1	0	0	1	0	1	1	0
Percentage of runs:			4%	54%	16%	1%	16%	1%	7%	0.3%

Thus, several different networks were found that could solve the problem, and the pattern of weights found on a particular run depended on the particular path through weight space taken by the probabilistic algorithm. In each solution the hidden unit was used to select one of the inputs as different from the rest; in a sense, the hidden unit was used to represent a second-order fact about the input pattern, and this intermediate representation carried enough information to solve the second-order XOR problem. In the following examples, the hidden units also "discover" intermediate representations that allow the network to solve a higher-order problem, although in those cases it is not as obvious what representations have been found or how many there are.

7. Mirror symmetries

Consider a square $N \times N$ array of randomly generated binary patterns that have an axis of reflection symmetry, as shown in fig. 4. For every axis of symmetry, there are $2^{N^2/2}$ different input patterns. The task is to teach a network to correctly categorize these patterns according to the axis of symmetry. To make the problem more difficult, imagine that the arrangement of the pixels is randomly scrambled by fiber optics, so that the symmetry is not apparent. How difficult a problem is this? This is a second-order problem in the sense of Minsky and Papert because single pixels by themselves carry no information about the solution to the problem, but information can be extracted from pairs of pixels that are related by the mirror symmetries. In terms of predicates, the solution can be written as a conjunction of second-order disjunctions (see Valiant [7] for a general discussion). For example, the predicate for horizontal mirror symmetry is:

$$\varphi_{\mathrm{H}} = (a_1 b_1 + \bar{a}_1 \bar{b}_1)(a_2 b_2 + \bar{a}_2 \bar{b}_2) \ldots, \qquad (7)$$

where a_i and b_i represent pixels in the array that are mirror reflections of each other with respect to the horizontal axis.

Input patterns were generated with three axes of symmetry, horizontal, vertical and one of the diag-

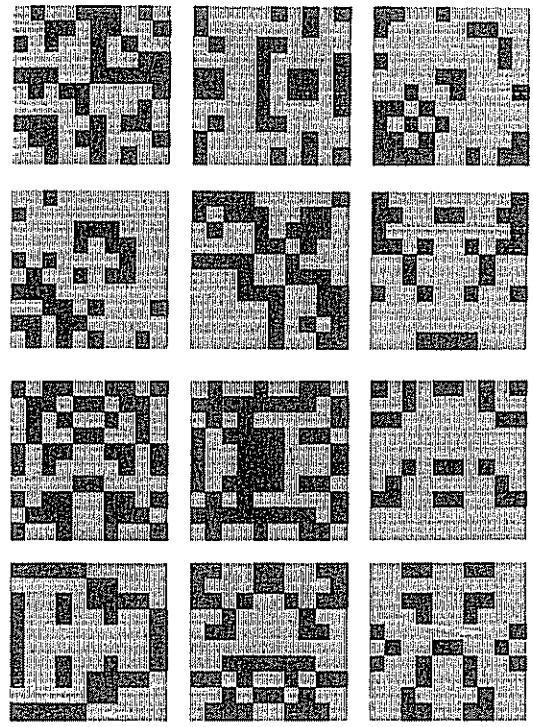

Fig. 4. Sample input patterns for the 10×10 mirror symmetry problem. The dark squares represent input units that are clamped to value 1 and the light squares are units that are clamped to value 0

onals, with each pixel having a probability of 0.4 of being on. Patterns with more than one axis of symmetry were rejected. Each unit in the input array was connected to each of the hidden units, which were in turn connected to each of three output units representing the three axes of mirror symmetry. The hidden units were not connected with one another. The weights were updated after every 5 randomly generated input patterns. Learning curves for arrays of size 4×4 and 10×10 with 12 hidden units each are shown in fig. 5. The larger array took longer and had poorer overall performance; however, the number of possible input patterns for the large array was about 2^{42} times larger than for the small array.

The pattern of weights for the 4×4 array is shown in fig. 6, and for the 10×10 array in fig. 7. Different runs with the same size of array had different patterns of weights, but in all runs, for all sizes of arrays, the patterns of weights shared the same qualitative properties: The weights of each hidden unit were either symmetric or antisymmetric about one or more of the three axes. For many hidden units the weights were antisymmetric about two of the axes and symmetric around the one axis that had an output weight of opposite sign from the other two axes. The weight to the symmetric axis was excitatory if the bias of the hidden unit was negative, or inhibitory if the bias was positive. The spatial distribution of the weights from the input units within the receptive field varied considerably amongst hidden units, with many types of geometric features represented: linear features, circular features, and isolated pixels. The relationships amongst the receptive fields of the hidden units were important; however, the hidden units have no direct connections, so the joint relationships amongst the hidden units must be mediated by feedback connections from the output units.

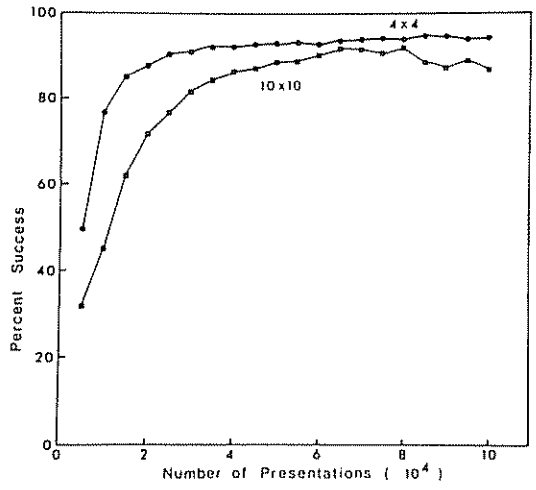

Fig. 5. Learning curves showing the performance for the 4×4 and 10×10 mirror symmetry problems with 12 hidden units each, using the fast annealing schedule. With slow annealing the performance at 100,000 presentations was 98.2% for the 4×4 problem and 90% for the 10×10 problem.

In nearly every run all the weights to some hidden units were much smaller than average, and they made no apparent contribution to the solution of the problem. The reason for this is not clear, but one possible explanation may be related to the tendency for the weights to decay. When the learning has reached steady-state, the size of each weight is determined by a balance between the tendency to decay and the learning gradient; therefore, weights to hidden units that make no significant contribution tend to get smaller. A unit that has small weights may find it more difficult to establish a direction in weight space that is not already occupied by some other hidden unit. In a sense, the hidden units tend to repel each other from the "territory" they cover in the space of input patterns.

In all of the mirror symmetry problems thus far the symmetry axes were fixed. A more difficult problem is to categorize mirror input patterns when the axes of symmetry were allowed to translate. For example, if the edges of the 10×10 array are identified so that it has the topology of a torus, then the vertical axis can be positioned between any of the 10 pairs of adjacent columns. It is then no longer possible to solve the problem with receptive fields of arbitrary geometry: the only translationally invariant patterns with the required symmetries are stripes and checkerboards. These are

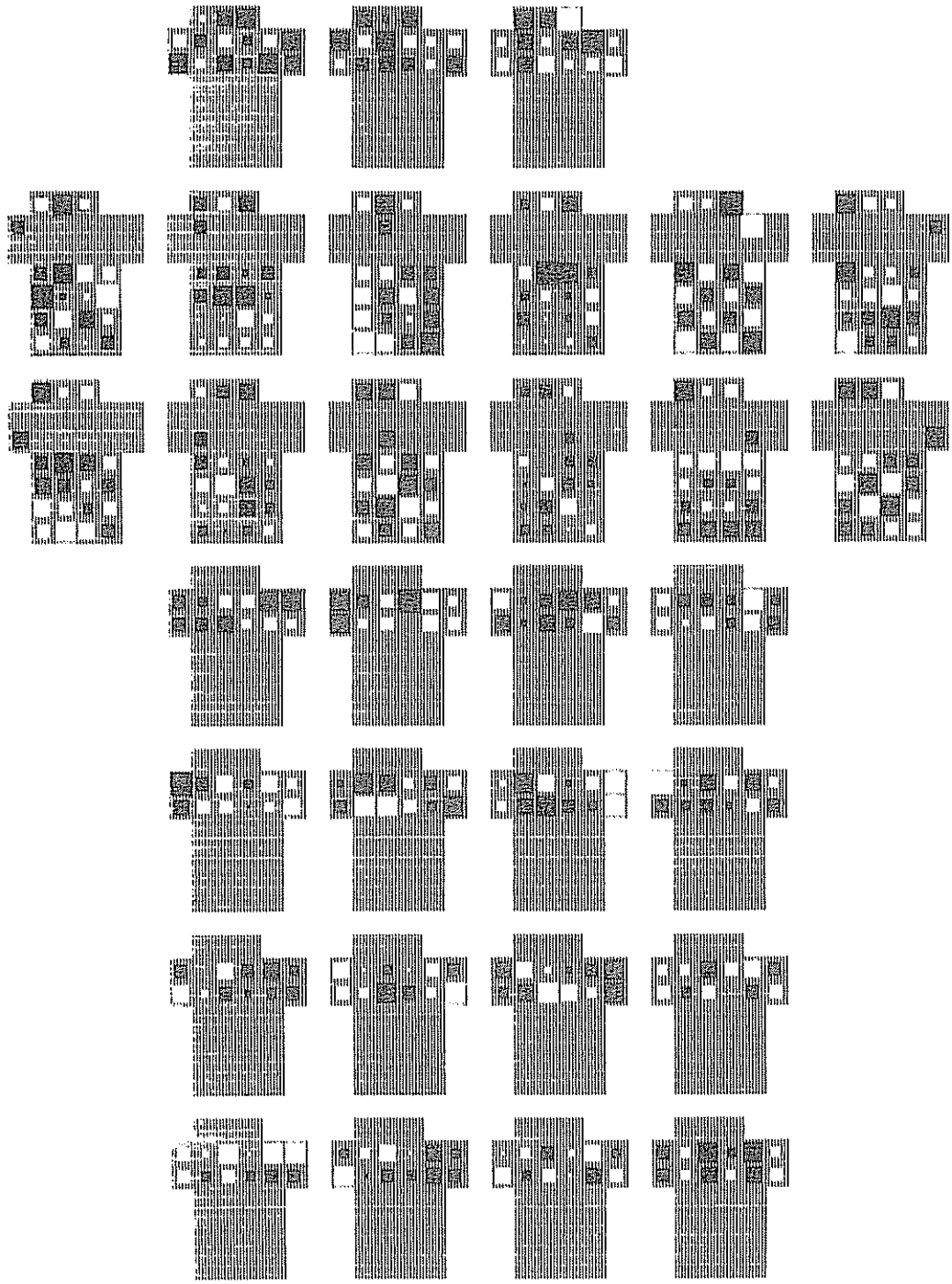

Fig. 6. Self-similar representation of the weights found by the Boltzmann learning algorithm for the 4×4 mirror symmetry problem. The 16 input units are arranged in a square array below, the 3 output units are in the top layer, and the 12 hidden units are between them. The output units from left to right represent the horizontal, vertical and right diagonal mirror symmetries respectively. (See fig. 3 for an explanation of the graphical representation used for weights.)

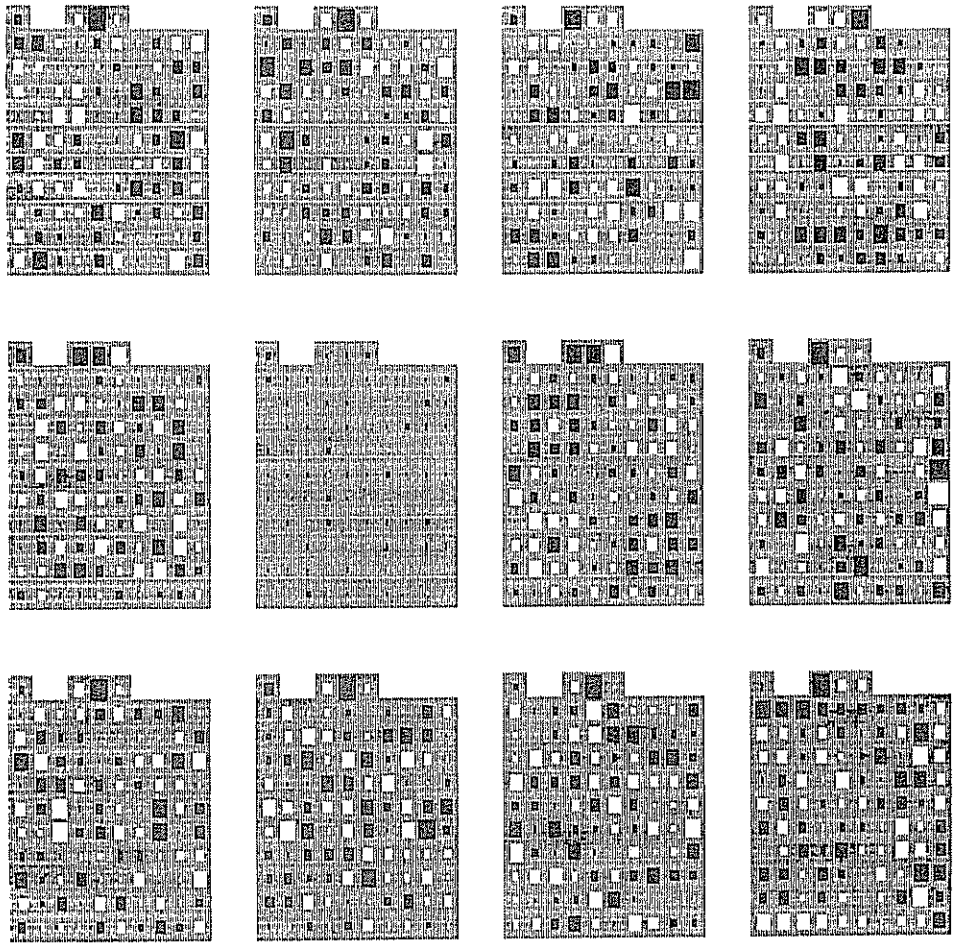

Fig. 7. Receptive field patterns formed by the weights to the hidden units found by the Boltzmann learning algorithm for the 10×10 mirror symmetry problem. Only the weights to the 12 hidden units are shown, with the weights from the input units shown in the 10×10 array, the weights to the 3 output units shown above the array, and the bias shown on the upper left corner. The output units from left to right represent the horizontal, vertical and right diagonal mirror symmetries respectively.

precisely the patterns of weights found by the learning algorithm, as shown in fig. 8.

8. T–C problem

A third-order problem is one that cannot be solved without using intermediate units that examine triples of input units. Minsky and Papert ([10], p. 102) describe a third-order problem that requires the discrimination between two 5-unit templates for a "T" and a "C" shape that can occur in any position in an array and in any 90° orientation, as shown in fig. 9. We have used networks with the same connectivity as that used for the mirror symmetry problem and have explored how the learning scales with the size of the array and the number of hidden units.

The learning algorithm was able to solve the T–C problem using finite arrays with the topology of a torus; the weights are shown in fig. 10 for a 6×6 array with 24 hidden units. The way that the

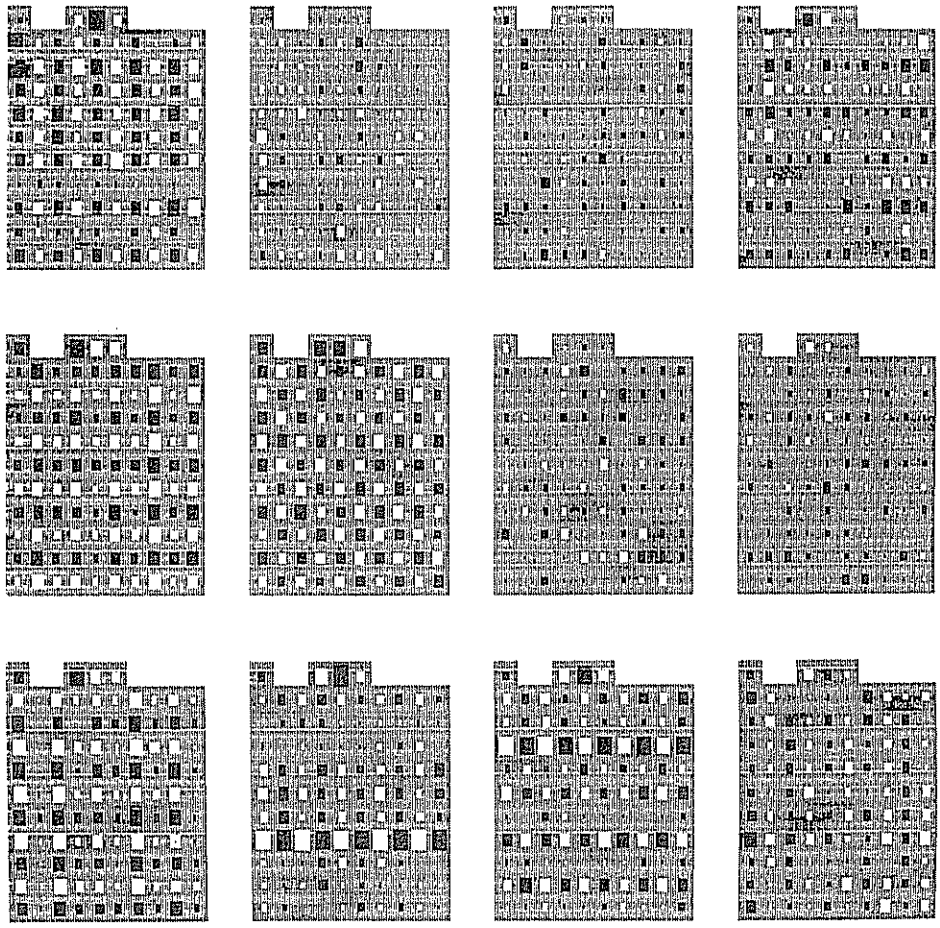

Fig. 8. Receptive field patterns formed by the weights to the hidden units found by the Boltzmann learning algorithm for the 10×10 mirror symmetry problem with translated symmetry axes. Only the weights to the 12 hidden units are shown, with the weights from the input units shown in the 10×10 array, the weights to the 3 output units shown above the array, and the bias shown on the upper left corner. The output units from left to right represent the horizontal, vertical and right diagonal mirror symmetries respectively.

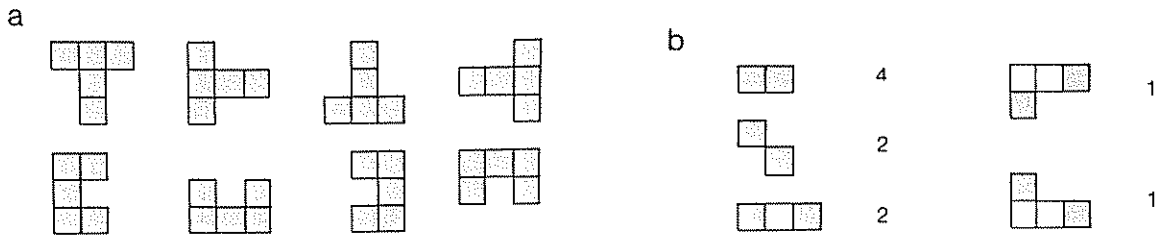

Fig. 9. a) Eight input templates for the T–C problem. The "T" is shown in the top row and the "C" in the bottom row, each in four different orientations. The templates can be translated to any position in the square array, with the edges of the array identified so that the topology is that of a torus. b) Distance between point-pair spectra is the same for the "T" and the "C" patterns allowing for all possible 90° rotations of the templates. For example, there are 4 second order pairs of adjacent units in both templates. The lowest order spectra that distinguish the two rotation- and translation-invariant templates are point-triples [10].

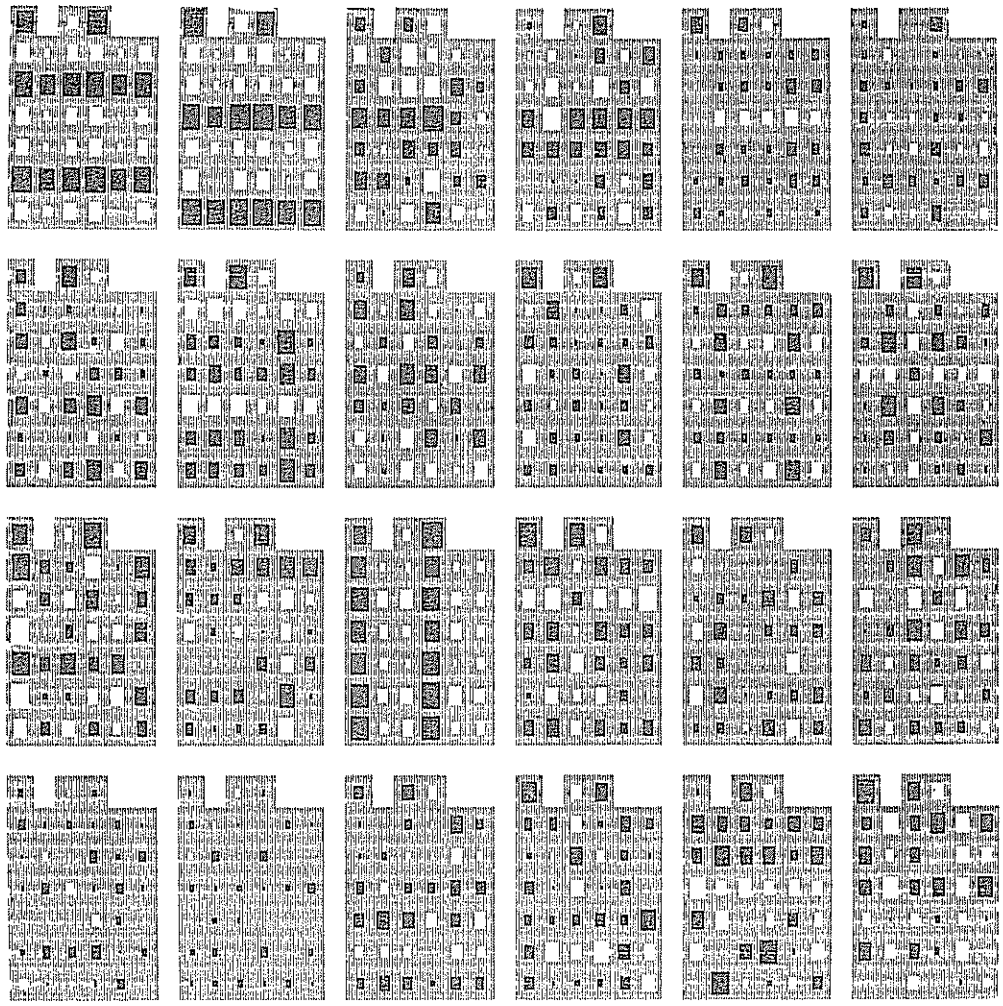

Fig. 10. Receptive field patterns formed by the weights to the hidden units found by the Boltzmann learning algorithm for the 6×6 T–C problem. Only the weights to the 24 hidden units are shown. The bias is shown on the upper left corner, and the two weights to the two output units are shown on the top; the left output unit represents the "T" and the right output unit represents the "C".

problem was solved by the network was less obvious than in the case of mirror symmetries. One common feature was a large inhibitory weight flanked by two smaller excitatory weights, sometimes arranged in triples that alternated between one row of inhibitory weights and two rows of excitatory weights. This was an effective combination because the "T" has two rows of three contiguous units in it, whereas "C" only has one and thus a 'T" would be more likely to turn off the hidden unit.

Learning curves for arrays of size 4×4 through 10×10 with 24 hidden units are shown in fig. 11. As the size of the array increased, a longer learning time was required to reach steady state, and the performance became worse. Especially for the larger arrays there was a long period at the beginning of the learning when the performance was at

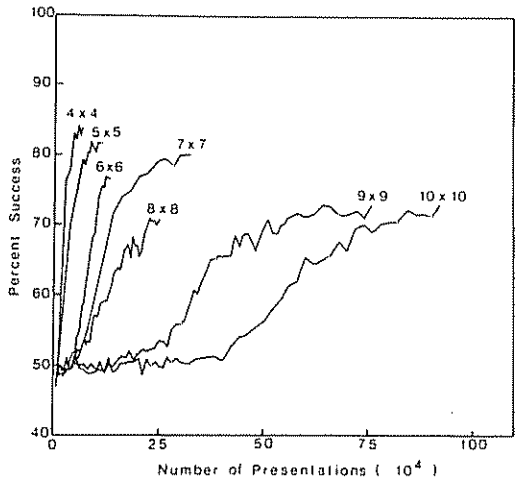

Fig. 11. Dependence of the rate of learning on the size of the input array for the T–C problem. Learning curves are shown (in some cases terminated before reaching an asymptote) for arrays of size 4×4 to 10×10, each with 24 hidden units.

chance level, and only gradually did patterns develop in the weights. This probably indicates that a random search for particular combinations of weights must occur before gradient descent can be effective.

The number of hidden units also affected the rate of learning and the performance. As shown in fig. 12, learning with only 6 hidden units was slow and performance was poor. There was significant variation from run to run so that the rates of learning and performances of 12 and 24 hidden units were similar, but were better for 48 hidden units. The shape of the curve for 6 hidden units resembled the shape of the slowest learning curves in fig. 11, which suggests that performance on the larger arrays could have been substantially improved with more hidden units.

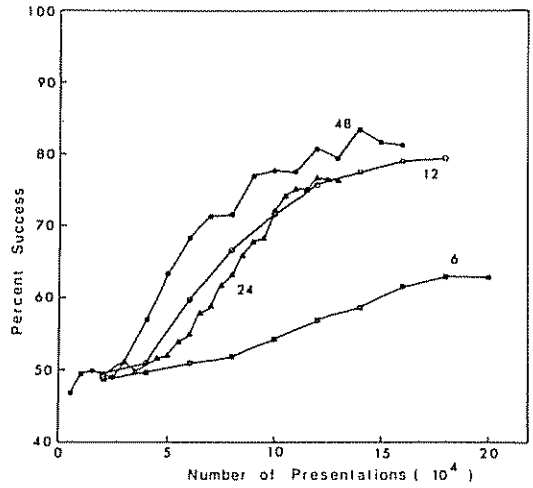

Fig. 12. Dependence of the rate of learning on the number of hidden units for the T–C problem for a 6×6 input array. The number of hidden units varied from 6 to 48, as indicated beside each learning curve.

9. Discussion

We have empirically explored the Boltzmann learning algorithm in several problems that perceptrons can only solve if they are hand-wired for each task. If the networks are not hand-wired these problems require adaptive hidden units between the input and output layers. With a sufficient number of hidden units, good performance was achieved for second- and third-order problems, even though the number of input presentations required was for some problems excessively large.

True generalization was achieved by the learning algorithm in the mirror symmetry problem, since for large arrays the number of possible input patterns was so large that virtually every input pattern presented was unique. In contrast, the T–C problem was much more difficult to learn even though the total number of T–C patterns was much smaller and memorization is a possible strategy for solving the problem. The difficulty of the problem for the learning algorithm is therefore correlated better with the order of the problem as defined by Minsky and Papert than with the absolute number of patterns to be categorized. The order of a problem is likely to be a good measure of its degree of difficulty for the Boltzmann learning algorithm.

For large arrays the T–C problem would have been substantially easier to solve if the receptive fields of the hidden units were made smaller than the entire input array, just as the receptive fields

of ganglion cells are arranged on the retina. A spatially restricted receptive field gives the hidden unit considerable information about the topology of the search space, greatly reducing the number of combinations of weights from the input units that need to be searched and reducing the need to make the receptive field consistent over the entire array. Restricting the connectivity in a way that is known to help in the solution of the problem puts knowledge about the task domain into the network. However, the goal of this study was to see whether the learning algorithm was capable of finding internal representations which could solve the problem without outside intervention. In none of the simulations for the T–C problem did any hidden unit narrow its receptive field to cover less than the entire input array.

Several other learning algorithms have recently been proposed which are also capable of solving higher-order learning problems. Barto and Andersen [30] have presented a stochastic learning algorithm for layered adaptive networks that does not require relaxation to equilibrium. This algorithm also requires many input presentations to solve higher-order problems, although simulations on a sequential machine run faster because fewer iterations are needed per presentation. Rumelhart, Hinton and Williams [31] have introduced a promising learning algorithm for layered networks that is deterministic and relies on back-propagation of errors. We will compare the performance of these learning algorithms in a later paper.

At least some of the virtues of the perceptron are preserved in the Boltzmann machine, which also overcomes some of its limitations. We anticipate that network architectures with hidden units will be useful in other domains where learning algorithms are needed.

Acknowledgements

This research was supported by grants from the System Development Foundation and grants to T.J.S. from the National Science Foundation, General Electric Corporation, Exxon Education Foundation, Allied Corporation Foundation, Westinghouse, and Smith, Kline & French Laboratories.

Appendix A

Order of a predicate

Minsky and Papert [10] defined a concept of order for predicates which is useful for analyzing the computational power of parallel machines. Let R be the set of atomic propositions and φ be a predicate over R. Define the support $S(\varphi)$ as the number of different atomic propositions in φ.

Definition. The order of φ is defined as the smallest number k for which there exists a basis set of predicates $\{\Phi_a\}$ satisfying:

$$S(\Phi_a) \leq k,$$
$$\varphi \in L(\Phi_a),$$

where $L(\Phi_a)$ is the set of all predicates that can be expressed as disjunctions, or linear combinations, of the basis set $\{\Phi_a\}$.

The motivation behind this definition is that in a parallel machine it may be desirable to implement complex predicates as combinations of simpler ones, and the order of the predicate is an indication of the minimum complexity required for the basis set. The basis set may be considered the internal representations that are used by the machine to implement the predicate; however, the order of a predicate is a property only of that predicate and does not depend on the particular choice of internal representations.

Appendix B

Collecting statistics

There are several ways to compute estimates of the ensemble averages needed in the Boltzmann learning algorithm. A straightforward way to estimate the ensemble average

$$P_{ij}^{+} = \sum_{\alpha} \sum_{\beta} P^{+}(O_\alpha | I_\beta) s_i^{\alpha\beta} s_j^{\alpha\beta}$$

is to reach equilibrium and accumulate the time average, eq. (5). Since the states are Markov processes [18, 32] the system is ergodic and the time average should converge to the ensemble average.

A better estimate for the ensemble average may be obtained using information from the summed inputs to the units as well as their outputs. For example, the time average can be rewritten as

$$P_{ij}^{+} \approx \langle s_i | s_j \rangle_t^{+} \langle s_j \rangle_t^{+}.$$

The first term can be estimated by averaging P_i given $s_j = 1$, and the latter term can be estimated by averaging P_j.

References

[1] J.A. Feldman and D.H. Ballard, Connectionist models and their properties, Cog. Sci. 6 (1982) 205–254.

[2] D.H. Ballard, G.E. Hinton and T.J. Sejnowski, Parallel visual computation, Nature 306 (1983) 21–26.

[3] G.E. Hinton and T.J. Sejnowski, Optimal perceptual inference, Proc. IEEE Computer Society Conf. Computer Vision & Pattern Recognition, Washington, D.C., (1983) 448–453.

[4] T. Poggio, V. Torre and C. Koch, Computational vision and regularization theory, Nature 317 (1985) 314–319.

[5] T.J. Sejnowski and G.E. Hinton, Separating figure from ground with a Boltzmann machine, in: Vision, Brain & Cooperative Computation, M.A. Arbib and A.R. Hanson, eds. (MIT Press, Cambridge, 1985).

[6] J.J. Hopfield and D. Tank, "Neural" computation of decision in optimization problems, Biol. Cybernetics (1985).

[7] L.G. Valiant, A theory of the learnable, Communications of the ACM, 27 (1984) 1134–1142.

[8] L.G. Valiant, Learning disjunctions of conjunctions, Proc. Ninth Intl. Joint Conf. Artif. Intell. (Morgan Kauffmann, Los Altos, CA, 1985) pp. 560–566.

[9] F. Rosenblatt, Principles of Neurodynamics (Spartan, New York, 1959).

[10] M. Minsky and S. Papert, Perceptrons (MIT Press, Cambridge, 1969).

[11] J.J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, Proc. National Academy of Sciences USA 79 (1982) 2554–2558.

[12] S.E. Fahlman, G.E. Hinton and T.J. Sejnowski, Massively-parallel architectures for AI: NETL, THISTLE and Boltzmann Machines, Proc. National Conf. Artificial Intelligence, Washington, D.C. (1983) 109–113.

[13] D.H. Ackley, G.E. Hinton and T.J. Sejnowski, A learning algorithm for Boltzmann machines, Cognitive Science 9 (1985) 147–169.

[14] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller and E. Teller, Equation of state calculations by fast computing machines, J. Chem. Phys. 21 (1953) 1087–1092.

[15] S. Kirkpatrick, D.D. Gelatt and M.P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.

[16] J. Moussouris, Gibbs and Markov random systems with constraints, J. Statis. Physics, 10 (1974) 11–33.

[17] R. Kindermann and J.L. Snell, Markov random fields and their applications (American Mathematical Society, Providence, 1980).

[18] S. Geman and D. Geman, Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images, IEEE Transactions on Pattern Analysis and Machine Intelligence 6 (1984) 721–741.

[19] P. Smolensky, Schema selection and stochastic inference in modular environments, Proc. Natl. Conf. on Artif. Intel., Washington, D.C. (1983) 378–382.

[20] V. Cerny, Multiprocessor system as a statistical ensemble: a way towards general-purpose parallel processing and mind computers?, preprint, Institute of Physics and Biophysics, Camenius University, Bratislava, Czechoslovakia (1983).

[21] P. Paretto, Collective properties of neural networks: a statistical physics approach, Biological Cybernetics 50 (1984) 51–62.

[22] D.J. Amit, H. Gutfreund and H. Sompolinsky, Spin-glass models of neural networks, Phys. Rev. A 32 (1985) 1007–1018.

[23] D.J. Amit, H. Gutfreund and H. Sompolinsky, Storing infinite numbers of patterns in a spin-glass model of neural networks, Phys. Rev. Letters 55 (1985) 1530–1533.

[24] G. Toulouse, S. Dehaene and J.-P. Changeux, Spin glass model of learning by selection, Proc. National Academy of Sciences USA 83 (1986) 1695–1698.

[25] A.P. Dempster, N.M. Laird and D.B. Rubin, Maximum likelihood from incomplete data via the EM algorithm, J. Roy. Stat. Soc. B39 (1977) 1–38.

[26] L.R. Bahl, F. Jelinek and R.L. Mercer, A maximum likelihood approach to continuous speech recognition, IEEE Trans. Pattern Analysis and Machine Intelligence, 5 (1983) 179–190.

[27] S. Kullback, Information theory and statistics (Wiley, New York, 1959).

[28] F. Crick and G. Mitchison, The function of dream sleep, Nature 304 (1983) 111–114.

[29] J.J. Hopfield, D.I. Feinstein and R.G. Palmer, "Unlearning" has a stabilizing effect in collective memories, Nature 304 (1983) 158–159.

[30] A.G. Barto and C.W. Andersen, Structural learning in connectionist systems, Proc. Seventh Annual Conf. Cognitive Science Soc. (1985).

[31] D.E. Rumelhart, G.E. Hinton and R.J. Williams, Learning internal representations by error propagation, in: Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1: Foundations, D.E. Rumelhart and J.L. McClelland, eds. (MIT Press, Cambridge, 1986).

[32] K. Binder, The Monte-Carlo method in statistical physics (Springer, New York, 1978).