Neural Networks, Vol. 2, pp. 59–67, 1989
Printed in the USA. All rights reserved.

0893-6080/89 $3.00 + .00
Copyright © 1989 Pergamon Press plc

ORIGINAL CONTRIBUTION

Relating Boltzmann Machines to Conventional Models of Computation

IAN PARBERRY AND GEORG SCHNITGER

The Pennsylvania State University

(*Received February 1988; revised and accepted June 1988*)

Abstract—*It is shown that clocked Boltzmann machines are not much more powerful than combinational circuits built from gates which compute Boolean threshold functions and their negations. More formally, any clocked Boltzmann machine can be simulated by a threshold circuit with running time greater by a constant factor and size greater by a polynomial.*

Keywords—Approximate language recognition, Boltzmann machine, Combinational circuit, Computational Complexity, Non-uniform model, Probabilistic computation, Scalable model, Threshold circuit.

1. INTRODUCTION

One of the interesting problems in artificial intelligence is the development of a plausible low-level model of cortical activity. Much research is currently aimed at determining whether these low-level models can emulate high-level cortical functions such as learning. The discovery of simple models which can mimic the brain may throw some light on the fundamental principles governing brain function, or at the very least may indicate how comparable systems may be implemented in silicon or act as inspiration for a new technology.

Connectionist models of the brain have recently regained popularity amongst researchers in artificial intelligence. The connectionist model (sometimes termed a neural network) is an interconnected set of simple neuron-like processing elements. One such model is the Boltzmann machine (see Ackley, Hinton, & Sejnowski, 1985; Hinton & Sejnowski, 1986; Hinton, Sejnowski, & Ackley, 1984) which can be described as a directed graph in which vertices represent processors, and edges represent (directed) links between processors. Each vertex is labeled with a *threshold value*, and each edge with a *weight*, both of which are integers. Each processor can be in one of two states, which are called *active* and *inactive*, and can change state as follows. At time t it computes the sum of the weights of the edges connecting it to its active neighbors and becomes active at time $t + 1$ with probability depending on the difference between that sum and its threshold. At the start of the computation a distinguished set of input vertices is held in either the active or inactive state to represent the input in binary. The output is similarly encoded in the states of a distinguished set of output vertices on completion of the computation.

A Boltzmann machine learns by modifying its edge-weights during the course of a "learning phase" in which inputs and their corresponding outputs are presented. In the experimental work published to date (e.g., Ackley, Hinton, & Sejnowski, 1985; Hinton & Sejnowski, 1986; Hinton, Sejnowski, & Ackley) once a function f has been learned by the Boltzmann machine, the performance of the learning algorithm is measured by fixing the weights and comparing the desired function f to the function computed by the Boltzmann machine *acting as a classical automaton*. It is instructive to characterize the functions which can be computed efficiently by the Boltzmann machine in this "classical mode," since if there is no weight assignment which allows efficient computation of f, then a Boltzmann machine cannot learn to compute f efficiently.

The resources of *running time* and *hardware* will be measured as functions of the size of the problem which the machines are to solve. It will be shown that the connection graph can be made acyclic, the

Research sponsored by the Air Force Office of Scientific Research, Air Force Systems Command, USAF, under grant number AFOSR 87-0400.

Research supported by NSF grant DCR-84-07256 to G.S.

Requests for reprints should be sent to Dr. I. Parberry, Department of Computer Science, 333 Whitmore Laboratory, The Pennsylvania State University, University Park, PA 16802.

machine can be made deterministic (that is, all random behaviour can be removed), and all edge-weights can be made equal to unity. That is, a Boltzmann machine can be reduced to a *combinational circuit.* Compared to the original network, the resulting circuit will have running time greater by a constant multiple and hardware requirement greater by a polynomial.

The remainder of this paper is broken up into six short sections. In the first section a formal definition of a Boltzmann machine is presented, and in the second section a formal definition of a computation of this machine is given. In the third section the model is augmented with conventional deterministic processors and random inputs. In the fourth section it is demonstrated that cycles can be removed from the connection graph of the machine. The fifth section shows how probabilism can be removed from the processors by using a standard sampling technique. The sixth section removes edge-weights, effectively reducing the model to a standard combination circuit. A preliminary version of this paper appears in Parberry and Schnitger (1987).

2. THE BOLTZMANN MACHINE

It is natural to focus on Boltzmann machines which are, in neural networks terminology, *scalable*, that is, can be scaled to problems of any size without too great an increase in running time or hardware. Minor modifications to the standard definition of Boltzmann machines must be made in order to render such a study possible. A Boltzman machine will be considered to be an infinite family $B = \langle B_1, B_1, \ldots \rangle$ of finite machines, one for each input size. Each finite machine B_n consists of:

1. A directed graph $G_n = (V_n, E_n)$. V_n is a set of *vertices* or *processors*, and $E_n \subseteq V_n \times V_n$ is a set of *edges*.
2. A distinguished set of *input processors*, $I_n \subseteq V_n$.
3. A distinguished set of *output processors*, $O_n \subseteq V_n$.
4. A distinguished set of *initially-active processors*, $A_n \subseteq V_n, A_n \cap I_n = \{\}$.
5. A threshold assignment, $h_n: V_n \to \mathbf{Z}$, which assigns a *threshold* to every processor.
6. A weight assignment, $w_n: V_n \times V_n \to \mathbf{Z}$, which assigns a *weight* to every edge. It is useful to adopt the convention that if $(u, v) \notin E_n$, then $w_n(u, v)$ is defined to be zero.
7. A temperature function $\tau_n: V_n \times \mathbf{N} \to \mathbf{N}$. τ_n gives for each processor and time a "temperature." This "temperature" can be varied with time in order to perform simulated annealing (after Kirkpatrick, Gelatt, & Vecchi, 1983).

Each finite Boltzmann machine (for a fixed number of inputs) will be depicted using circles for vertices, and lines connecting them for edges. The direction of communication along these edges may be assumed to be from top to bottom, unless explicitly indicated by the presence of arrowheads. Threshold values will be placed inside the appropriate vertices, and weights alongside the appropriate edges. Each vertex will be named using numbers and letters if necessary, the name appearing next to the relevant vertex. Figure 1 shows a simple two-processor machine.

A *computation* of B on an input x consisting of n bits (write $x \in \{0, 1\}^n$) is defined as follows. At time $t = 0$ the input processors of B_n are placed into states which encode x. That is, the processors in I_n are numbered consecutively, and the ith such processor is placed in the active state if the ith bit of x is one. The processors of A_n are also placed in the active state. All other processors are placed in the inactive state. The processors are then allowed to change state according to the following rules. All computations are *clocked*, that is, there is a global clock available to aid in synchronization (in order to avoid race conditions and the effects of spurious behavior during state-change). Three modes of computation are prevalent in the literature:

1. Sequential operation. A single processor is chosen at random to be updated during each clock cycle. All other processors maintain their state.
2. Parallel operation. Each processor v is updated with some probability p_v during each clock cycle, and maintains its state otherwise.
3. Synchronous parallel operation. Every processor is updated during every clock cycle.

The state of an individual processor v is updated as follows. Define the output of processor v at time t, written $U_n(v, t)$, to be 1 if v is active at time t and 0 if it is inactive. Define the input to processor v at time $t > 0$, written $W_n(v, t)$, to be the sum of the

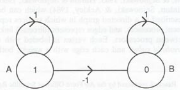

FIGURE 1. A Boltzmann machine for $n = 1$ with two processors, labelled "A" and "B." $I_n = \{A\}$, $O_n = \{B\}$ and $A_n = \{\}$.

weights of edges connecting it to processors which were active in the previous step, that is,

$$W_n(v, t) = \sum_{u \in V_n} w_n(u, v) U_n(u, t - 1)$$

Then v is active at time t with some probability $p(W_n(v, t) - h_n(v), v, t)$. Typically, in the literature the activation probability function

$$p(\Delta, v, t) = \frac{1}{1 + e^{-\Delta/\tau_n(v,t)}}$$

is used, but this is not crucial to the results to be discussed in this paper. Note that although input processors are permitted to participate in the computation, they can be "clamped" (as is preferred in much of the literature) by giving each input processor a unit threshold value and a unit-weight self-loop, presuming that all other edges are out-going.

It may be assumed that there is some predefined termination convention for Boltzmann machines. That is, for each n there is some finite time $T(n)$ at which the computation of B_n on an input of size n is deemed to be completed. At this time the output of B is encoded in the states of the output processors of B_n (analogously to the way that the input was encoded in the states of the input processors). The running time of B is then said to be $T(n)$. Note that the exact details of the termination condition are not crucial to this definition. $T(n)$ can be taken to be the worst-case running time over all inputs of size n provided there is some method of maintaining the output for up to $T(n)$ steps should the computation terminate at some earlier point for any particular input. This can be achieved with careful design and self-loops. B is said to have *size* (at most) $Z(n)$ if for all $n \geq 1$, $|E_n| \leq Z(n)$. Size will be used as a measure of hardware requirements. It is reasonable to assume that $|E_n| \geq |V_n|$ (which is the case for all but degenerate machines) so that the number of edges is a reasonable measure of size to within a small polynomial. Also assume that the absolute values of the edge weights (and therefore the thresholds) are bounded above by a polynomial in $Z(n)$, for concreteness, $Z(n)^c$ for some $c \in \mathbf{N}$. Thus $Z(n)$ is, to within a small polynomial, a good measure of the number of bits required to describe B_n. This is a reasonable assumption because there is experimental evidence (Hinton & Sejnowski, 1986) that the performance of learning algorithms is enhanced when the weights are kept small.

3. APPROXIMATE LANGUAGE RECOGNITION

Consider Boltzmann machines which have a single output, that is $|O_n| = 1$ for all n. If $x \in \{0, 1\}^n$, B is said to *accept* x if the computation of B_n on input x terminates with the output of the output processor equal to 1, and to *reject* x otherwise. A *language* L is a set of strings of zeros and ones. B is said to *recognize* a language L if it can determine whether or not an input x belongs to L with probability of error bounded away from 0.5. That is, it recognizes L iff there is a real number $\epsilon > 0$ such that:

(i) for all $x \in L$, the probability that B accepts x is $\geq 0.5 + \varepsilon$.
(ii) for all $x \notin L$, the probability that B rejects x is $\geq 0.5 + \varepsilon$.

This is often called *two-sided bounded-error probabilism* and has a well-studied analogue for sequential computation.

Two-sided bounded-error probabilism appears at first to be an unnatural choice; for example, a language recognizer which answers correctly with probability 0.6 is not very reliable. However, by repeating a computation many times and taking the consensus, the probability of correctness can be increased to $0.5 + \mu$ for arbitrarily large $\mu < 0.5$, using a standard result from sequential complexity theory.

Lemma 1 (Chernoff, 1952): In a sequence of N independent Bernoulli trials each with probability p of success, the probability that at least m trials succeed, denoted $B(m, N, p)$, has the property

$$B(m, N, p) \leq \left(\frac{Np}{m}\right)^m \left(\frac{N - Np}{N - m}\right)^{N-m}$$

provided $m \geq Np$.

Taking $p = 0.5 - \varepsilon$ and $m = N/2$ in Lemma 1 it follows that the probability of more than half of the N trials failing is given by

$$B(N/2, N, 0.5 - \varepsilon) \leq (1 - 4\varepsilon^2)^{N/2}$$

Thus if

$$N = 2\frac{\log \lambda}{\log(1 - 4\varepsilon^2)}$$

trials are made and the majority decision taken, the probability of failure is reduced to λ, for any $0 < \lambda < \varepsilon$. For example, a Boltzmann machine with only 60% chance of making the correct decision can be used to obtain 99.9% certainty with 339 trials, regardless of the size of the input.

Note that the above results hold equally well for Boltzmann machines which compute functions. Let $f: \{0, 1\}^n \rightarrow \{0, 1\}^m$. A Boltzmann machine is said to compute f if for all inputs x and for all i, the machine terminates with the ith bit of its output equal to the ith bit of $f(x)$ with probability $0.5 + \varepsilon$. Given a machine with m outputs, make m copies of it and take one bit of output from each. Each independent copy can then be treated as a language recognizer. In comparison to the original machine, the new composite machine has identical running time and size at most quadratically larger.

4. AN AUGMENTED BOLTZMANN MACHINE

It is convenient to augment the Boltzmann machine with deterministic processors of two different types. An *upper-threshold* processor, v, is active at time t iff $W_n(v, t) \geq h_n(v)$. A *lower-threshold* processor, v, is active at time t iff $W_n(v, t) \leq h_n(v)$. A deterministic processor will be depicted as a circle containing the symbol "≤" for a lower-threshold and "≥" for an upper-threshold processor, followed by the threshold value. It is easy to produce processors which compute the Boolean AND and Boolean OR of the states of a set of processors, and the Boolean negation of a single processor. An AND processor is an upper-threshold processor with threshold k and edges of weight 1 from k other processors. An "AND" processor is depicted as a circle containing the symbol "&." An OR processor is an upper-threshold processor with threshold 1 and edges of weight 1 from k other processors. An "OR" processor is depicted as a circle containing the symbol "V." A negation processor is a lower-threshold processor with threshold 0 and an edge of weight 1 from another processor. A negation processor is depicted as a circle containing the symbol "¬."

The model is also augmented with a set of *random imputs*, $R_n \subseteq V_n$, $R_n \cap I_n = \{\}$, $R_n \cap A_n = \{\}$, each of which is independently assigned a state (with the appropriate probability of being active) at the start of each computation. A random input is depicted using a circle containing the word "random" and the probability that it is initially active.

5. REMOVAL OF CYCLES

A Boltzmann machine may have cycles (sometimes called feedback loops) in its connection graph. A standard technique from Savage (1972) (used more recently in Goldschlager & Parberry, 1986, and Parberry & Schnitger, 1986, in press) can be used to remove these cycles. For each machine with cycles it is possible to produce a combinational circuit which has the same input–output behavior (in particular, which recognizes the same language) with a polynomial increase in size and no increase in time.

Let B be a Boltzmann machine of size $Z(n)$ and running time $T(n)$. The acyclic machine $\hat{B}$ consists of $T(n) + 1$ "snapshots" of B, one at each point in time. As a consequence, $\hat{B}$ will have size $O(T(n)Z(n))$ and running time $T(n)$. If $T(n)$ is restricted to be at most polynomial in $Z(n)$, then the increase in size is at most a polynomial. This restriction to machines with running time not much larger than size is reasonable in the light of the fact that much can be achieved within polynomial time with polynomial size.

It is instructive to consider the synchronous parallel case first. This is the easiest since the operation of a combinational circuit is in a sense synchronous parallel. More formally, $\hat{B} = \langle \hat{B}_1, \hat{B}_2, \ldots \rangle$ where $\hat{G} = (\hat{V}_n, \hat{E}_n)$. For every vertex $v \in V_n$ there is a vertex representing v at time t, called $(v, t) \in \hat{V}_n$, for $0 \leq t \leq T(n)$. For every edge $(u, v) \in E_n$ there is an edge from the vertex representing u at time $t - 1$ to the vertex representing v at time t. The thresholds, weights, input processors, output processors and initially-active processors are mapped to the new machine in a natural fashion.

$$\hat{V}_n = \{(v, t) | v \in V_n, 0 \leq t \leq T(n)\}$$

$$\hat{E}_n = \{((u, t-1), (v, t)) | (u, v) \in E_n, 1 \leq t \leq T(n)\}$$

$$\hat{h}_n(v, t) = h_n(v) \text{ for } 0 \leq t \leq T(n)$$

$$\hat{w}_n((u, t-1), (v, t)) = w_n(u, v)$$

$$\hat{I}_n = \{(u, 0) | u \in I_n\}$$

$$\hat{O}_n = \{(u, T(n)) | u \in O_n\}$$

$$\hat{A}_n = \{(u, t) | u \in A_n, 0 \leq t \leq T(n)\}$$

$$\hat{\tau}_n((v, t), t) = \tau_n(v, t) \text{ for } 0 \leq t \leq T(n), v \in V_n.$$

For example, the acyclic machine in Figure 2 is equiv-

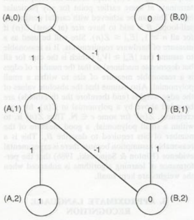

FIGURE 2. A Boltzmann machine without cycles equivalent to the machine in Figure 1, when the latter is run for two steps. $I_n = \{(A, 0)\}$, $O_n = \{(B, 2)\}$ and $A_n = \{\}$.

alent to the cyclic machine in Figure 1 when the latter is run for two steps.

In the case of a sequential Boltzmann machine, the same construction as above is used, with the addition of circuitry which ensures that only one processor in each level has the opportunity to change state. At most $\lceil \log Z(n) \rceil$ random inputs are added for each level, each random input with probability 0.5 of being active. The extra circuitry shown in Figure 3 is also added for each processor v at time t. The square box in that figure represents a circuit whose output is true if the sequence of new random inputs encodes the binary representation of v. The construction of such a circuit in constant depth and polynomial size is left as a trivial exercise for the reader. The resulting circuit has size $O(T(n)Z(n))$, and its depth needs to be extended to at most $O(T(n))$ in the case where $Z(n)$ is not a power of 2. In the case of a parallel Boltzmann machine, a similar modification is made to ensure that processor (v, t) is allowed to change state with probability p_v (see Figure 4).

6. REMOVAL OF RANDOMNESS

It is possible to replace each processor of the Boltzmann machine with a small number of deterministic processors and random inputs. A processor with threshold-value k, and s inputs with weights $w_1, \ldots, w_s$ can be replaced by the circuit in Figure 5. The circuit makes use of an *equality processor*. An equality processor v is active at time t iff $W_n(v, t) = h_n(v)$. An equality processor can be constructed from an upper-threshold, a lower-threshold and an AND processor as in Figure 6. Each (probabilistic) processor with threshold value k and s inputs in a Boltzmann machine with size $Z(n)$ is replaced by $2Z(n)^2 + 1$ random inputs, equality processors and AND processors, and a single OR processor. The key is to provide a random input for each possible probability.

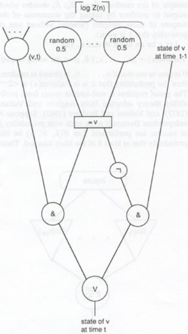

FIGURE 3. Extra circuitry required to ensure sequential operation of a combinational circuit.

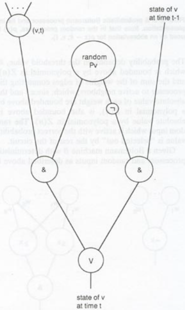

FIGURE 4. Extra circuitry required to ensure parallel operation of a combinational circuit.

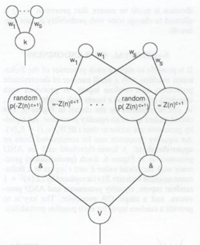

FIGURE 5. A probabilistic Boltzmann processor and its implementation. Note that in the random processors, $p(x)$ is used as an abbreviation for $p(x - k, v, t)$.

The probability depends on the threshold value, k, which is bounded above by a polynomial in $Z(n)$, and the sum of the weights of edges connecting the processor to active neighbors, which, since s and the absolute value of each weight are bounded above by a polynomial in $Z(n)$, is also bounded above in absolute value by a polynomial in $Z(n)$. The random input which is active with the correct probability value is "selected out" by the rest of the circuit.

Given a Boltzmann machine B with deterministic processors and random inputs as described above it is possible to produce a new machine $\hat{B}$ which is deterministic and recognizes the same language as B with only polynomially more processors and time greater by at most a constant multiple, using a well-known technique due to Adleman (1978). Suppose $B = \langle B_1, B_2, \ldots \rangle$ is such a Boltzmann machine which recognizes L with error probability $0.5 - \varepsilon$, where $\varepsilon > 0$ is a real number. If B_n is a finite Boltzmann machine with n inputs, and r is a fixed string of zeros and ones of the appropriate size, let $B_n(r)$ be the machine obtained by fixing the random-inputs according to r. That is, if the ith bit of r is 1, then the ith random input is placed in A_n. Suppose cn such strings are picked at random, where $c > (1 - 2\varepsilon)(\log_e 2)/\varepsilon^2$, choosing each random input independently with the appropriate probability. It will be demonstrated that there is a choice of such an $r_1, \ldots, r_{cn}$ such that the deterministic machine $\hat{B}_n$ depicted in Figure 7 recognizes the same language as B_n with no error. $\hat{B}_n$ consists of a copy of each of $B_n(r_1), \ldots, B_n(r_{cn})$. Each of those subcircuits is a sample of the random circuit B_n; $\hat{B}_n$ decides which output to produce by taking the consensus of the outputs of those samples (assume without loss of generality that c is even).

Let x be an input of size n. Let

$$\text{Failures}(x) = \{(r_1, \ldots, r_{cn}) \mid \hat{B}_n \text{ gives the wrong output}\}$$

It is easy to see that if $r_1, \ldots, r_{cn}$ is picked at random, then the probability that it is in Failures(x) is $< 2^{-n}$. The proof requires a well-known result from probability theory adopted from Angluin and Valiant (1977), and Valiant and Brebner (1982). Suppose N independent Bernoulli trials each with probability p of success are performed. Let $B(k, N, p)$ be the probability that at least k of the trials succeed. Then:

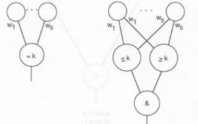

FIGURE 6. An equality processor and its implementation.

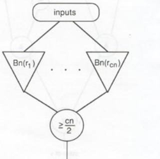

FIGURE 7. A deterministic machine $\hat{B}_n$ constructed from cn copies of B_n.

Lemma 2: If $k = Np(1 + \beta)$ for some $0 \leq \beta \leq 1$, then $B(k, N, p) \leq e^{-0.5\beta^2 Np}$.

Proof: The proof follows from Lemma 1. □

If r is picked at random, the probability that it fails is $0.5 - \varepsilon$. Without loss of generality, assume that $\varepsilon \leq 1/4$. If cn independent Bernoulli trials are performed to pick $r_1, \ldots, r_{cn}$, where $c > (1 - 2\varepsilon)(\log_e 2)/\varepsilon^2$, and take $N = cn$, $p = 0.5 - \varepsilon$, $\beta = 2\varepsilon/(1 - 2\varepsilon)$, $k = cn/2$, then by Lemma 2 the probability that there are at least $cn/2$ failures out of cn trials is

$$B(cn/2, cn, 0.5 - \varepsilon) \leq e^{-0.5cn(0.5-\varepsilon)} < 2^{-n}.$$

Therefore if $r_1, \ldots, r_{cn}$ are picked at random, the probability that it is in $\bigcup_x \text{Failures}(x)$ is less than one (since there are only 2^n possible strings x of length n). Hence there must be at least one choice of cn strings $r_1, \ldots, r_{cn}$ that make $\hat{B}_n$ work correctly for all inputs of size n. Therefore $\hat{B}$ recognizes the same language as B.

7. BOLTZMANN MACHINES AND THRESHOLD CIRCUITS

To complete the result, it is sufficient to observe that all edge-weights can be made equal to one. This can be achieved easily by first finding an equivalent circuit with positive edge-weights, and then replacing each edge of weight w with w edges of weight 1. A method for getting rid of negative edge-weights while squaring the size and increasing the running time by a constant factor appears in Parberry and Schnitger (1986, in press). A much better construction requiring only a linear increase in size was suggested to the first author by M. S. Paterson in 1986 and is implicit in a result described by Godbeer (1987). Each edge with negative weight w from processor i to processor j can be replaced with an edge of weight 1 from processor i to a new negation processor, and an edge of weight $-w$ from there to processor j, which has its threshold increased by the absolute value of w. The correctness of this construction can be deduced from the simple observation that for all $x \in \{0, 1\}$,

$$x = 1 - \bar{x} \qquad (*)$$

where $\bar{x}$ denotes the complement of x in $\{0, 1\}$: Consider an upper-threshold processor with threshold k, and inputs with weights $w_1, \ldots, w_m$ from m processors with outputs $x_1, \ldots, x_m$ respectively (that is, $x_i = 1$ iff the ith processor is active). It becomes active iff

$$\sum_{i=1}^{m} w_i x_i \geq k.$$

That is, by (*) above,

$$\sum_{i=1}^{m-1} w_i x_i - w_m \bar{x}_m \geq k - w_m.$$

Thus negative edge-weights can be removed from the machine while at most doubling the size and depth, and increasing the thresholds by a polynomial amount.

8. CONCLUSION

It has been shown that Boltzmann machines are in a sense equivalent to unbounded fan-in circuits built from gates which compute Boolean threshold functions. Unbounded fan-in threshold circuits have attracted a great deal of attention in the recent literature (see, e.g., Chandra, Stockmeyer, & Vishkin, 1984; Parberry & Schnitger, 1986, in press; Reif, 1987). It is perhaps surprising that so many of the key features of the Boltzmann machine (such as probabilism and the ability to perform simulated annealing) are unimportant. However, having shown that Boltzmann machines are weaker than they appear, two comments must be made. Firstly, Boltzmann machines may be more powerful than threshold circuits by up to a constant multiple in speed and a polynomial in size. This latter advantage may be of some use should the construction of Boltzmann machines become technologically feasible. Secondly, threshold circuits are extremely powerful. For example, they can multiply two integers (Chandra, Stockmeyer, & Vishkin, 1984), approximate convergent rational power series, perform integer and polynomial division, fast Fourier transform, polynomial interpolation, Chinese remaindering, compute elementary symmetric functions, and banded and triangular Toeplitz matrix inverses (Reif, 1987) in constant depth and polynomial size. Thus Boltzmann machines are certainly more powerful than classical sequential or parallel models of computation.

Many assumptions about Boltzmann machines have been made in this paper. Some of these are self-evidently reasonable, while others may be the subject of some controversy. In the interests of clarity, a list of the important assumptions follows.

1. *The computational paradigm*. The Boltzmann machine is formalized as an automaton which recognizes formal languages. That is, the Boltzmann machine is used as a computer which takes a bit-string input and gives a yes/no answer in response.
2. *Scalability*. Only scalable Boltzmann machines are considered. They embody an algorithm which can be applied to inputs of any size. The resources required for a computation are measured as a function of input size. This is achieved by defining a Boltzmann machine to be an infinite family of finite machines, each of which is to deal with inputs of a particular size. This type

of definition is standard in the study of conventional networks and circuits built from two-input Boolean functions (see, e.g., Russo, 1981; Parberry, 1987; Pippenger, 1979).

3. *Non-uniformity*. Non-uniform Boltzmann machines are studied. That is, each finite machine in the infinite family may look radically different. In contrast, in a uniform machine, a description of each finite n-input machine in the family should be easily computable from n (see, e.g., Ruzzo, 1981, for classical circuits, Parberry & Schnitger, 1986, in press, for Boltzmann machines). The source of non-uniformity in this paper is the nonconstructive proof of the existence of a deterministic machine corresponding to a random one.
4. *Unbounded Fan-in*. The Boltzmann machine is viewed as an *unbounded fan-in* computer. That is, the fan-in of the connection graph is not required to be fixed (independent of n). Unbounded fan-in circuits are already well-studied in the literature (see, e.g., Chandra, Stockmeyer, & Vishkin, 1984; Furst, Saxe, & Sipser, 1984; Stockmeyer & Vishkin, 1984).
5. *Fixed Weights*. Boltzmann machines whose weights and thresholds do not vary with time are studied. Variable weights are crucial for the learning algorithms in Ackley, Hinton, and Sejnowski (1985) and Hinton and Sejnowski (1986), but weights are kept fixed for computing the learned function. Different behavior may result from allowing the network to continue modifying its weights.
6. *Clocked Operation*. The network is assumed to be *clocked*, that is, the existence of a global clock which has a large enough period to avoid race conditions and the effects of spurious behavior during state-changes is assumed. Unclocked (asynchronous) networks of threshold processors are studied by Alon (1985) and Lepley and Miller (1984).
7. *Resources*. The resources of running time and hardware are considered. The number of connections, which we call the *size* of the network, is used as a measure of hardware, and the number of clock ticks as a measure of time.
8. *Termination*. It is assumed that there is some termination convention in which all computations on inputs of size n are finished after $T(n)$ steps, at which time the output can be determined. A typical termination condition is the convergence of the network to a steady-state. Hopfield (1982) shows that if the connection graph is undirected (i.e., for all $n \in \mathbf{N}$ and $u, v \in V_n$, $w_n(u, v) = w_n(v, u)$) and has no self-loops (i.e., for all $n \in \mathbf{N}$ and $v \in V_n$, $w(v, v) = 0$), then a deterministic network under sequential operation must reach a steady-state. Poljak and Sura (1983) show that the same type of network under fully parallel operation either reaches a steady-state or a cycle of length two. Many problems relating to the termination of deterministic networks are NP-complete (see Godbeer, 1987; Lipscomb, 1987; Luby, 1985; Porat, 1987).
9. *Small Weights*. It is assumed that the absolute values of the edge weights (and therefore the thresholds) are bounded above by a polynomial in the size of the machine. This is a fairly stringent requirement. Any polynomial-size deterministic network with arbitrary real weights can be realized with integer weights of polynomially many bits (Hong, 1987; Muroga, 1971; Muroga, Toda, &, Takasu 1961) without increase in size or time (note that this is exponentially larger than required). It follows from this observation that all weights can be made equal to unity if size is increased by a polynomial and running time by a constant multiple.
10. *Fast Computation*. One would expect that only networks of polynomial size would have a chance of being implemented. Furthermore, it is reasonable to expect that only networks which terminate in polynomial time will be useful. Therefore it is reasonable to study only computations for which the running time is bounded above by a polynomial in the size of the network. Note that some polynomial size networks must terminate in polynomial time. It is easy to show using the technique of Hopfield (1982) that if the connection graph is undirected (i.e., for all $n \in \mathbf{N}$ and $u, v \in V_n$, $w_n(u, v) = w_n(v, u)$) then a polynomial-size deterministic network with polynomial weights under sequential operation must reach a steady-state in polynomial time. It is also a relatively easy matter to extend the argument of Poljak and Sura (1983) to show that the same type of network under fully parallel operation will reach a steady-state or a cycle of length two in polynomial time.
11. *Probabilism*. Machines with *two-sided bounded-error probabilism* are studied. These machines have probability of being correct bounded away from 0.5.

REFERENCES

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

Adleman, L. (1978). Two theorems on random polynomial time. In *Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science* (pp. 75–83). Washington, DC: IEEE Computer Society Press.

Alon, N. (1985). Asynchronous threshold networks. *Graphs and Combinatorics*, **1**, 305–310.

Angluin, D., & Valiant, L. (1977). Fast probabilistic algorithms for Hamiltonian circuits and matchings. In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing* (pp. 30–41). New York: ACM Press.

Chandra, A. K., Stockmeyer, L. J., & Vishkin, U. (1984). Constant depth reducibility. *SIAM Journal on Computing*, **13**, 423–439.

Chernoff, H. (1952). A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, **23**, 493–507.

Furst, M., Saxe, J. B., & Sipser, M. (1984). Parity, circuits and the polynomial time hierarchy. *Mathematical Systems Theory*, **17**, 13–27.

Godbeer, G. (1987). *The computational complexity of the stable state problem for connectionist models*. Unpublished master's thesis, University of Toronto, Ontario, Canada.

Goldschlager, L. M., & Parberry, I. (1986). On the construction of parallel computers from various bases of Boolean functions. *Theoretical Computer Science*, **43**, 43–58.

Hinton, G. E., & Sejnowski, T. J. (1986). Learning and relearning in Boltzmann machines. In *Parallel distributed processing: Explorations in the microstructure of cognition* (Vol. 1, pp. 282–317). Cambridge, MA: The MIT Press.

Hinton, G. E., Sejnowski, T. J., & Ackley, D. H. (1984). *Boltzmann machines: Constraint satisfaction networks that learn*. (Tech. Rep. CMU-CS-84-119). Pittsburgh, PA: Carnegie-Mellon University.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, **79**, 2554–2558.

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

Hong, J. (1987). *On connectionist models*. Unpublished manuscript, Department of Computer Science, University of Chicago.

Lepley, M., & Miller, G. (1984). *Computational power for networks of threshold devices in an asynchronous environment*. Unpublished manuscript, Department of Mathematics, The Massachusetts Institute of Technology.

Lipscomb, J. (1987). *On the computational complexity of finding a connectionist model's stable state vectors*. Unpublished master's thesis, University of Toronto, Ontario, Canada.

Luby, M. (1985). A simple parallel algorithm for the maximal independent set problem. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing* (pp. 1–10). New York: ACM Press.

Muroga, S., Toda, I., & Takasu, S. (1961). Theory of majority decision elements. *Journal of the Franklin Institute*, **271**, 376–418.

Muroga, S. (1971). *Threshold logic and its applications*. New York: John Wiley and Sons.

Parberry, I. (1987). An improved simulation of space and reversal bounded deterministic Turing machines by width and depth bounded uniform circuits. *Information Processing Letters*, **24**, 363–367.

Parberry, I., & Schnitger, G. (1986). Parallel computation with threshold functions [Preliminary Version]. In *Proceedings of the Structure in Complexity Theory Conference*, Lecture Notes in Computer Science (Vol. 223, pp. 272–290). New York: Springer-Verlag.

Parberry, I., & Schnitger, G. (1987). Relating Boltzmann machines to conventional models of computation. In *Proceedings of the 2nd International Symposium on Methodologies for Intelligent Systems* (pp. 347–354). New York: Elsevier.

Parberry, I., & Schnitger, G. (1988). Parallel computation with threshold functions *Journal of Computer and System Sciences*, **36**(3), 278–302.

Pippenger, N. (1979). On simultaneous resource bounds. In *Proceedings of the 20th Annual IEEE Symposium on Foundations of Computer Science* (pp. 307–311). Washington, DC: IEEE Computer Society Press.

Poljak, S., & Sura, M. (1983). On periodical behavior in societies with symmetric influences. *Combinatorica*, **3**, 119–121.

Porat, S. (1987). Stability and looping in connectionist models with asymmetric weights. (Tech. Rep. TR 210). Rochester, NY: University of Rochester.

Reif, J. (1987). On threshold circuits and polynomial computation. In *Proceedings of the 2nd Structure in Complexity Theory Conference* (pp. 118–125). Washington, DC: IEEE Computer Society Press.

Ruzzo, W. L. (1981). On uniform circuit complexity. *Journal of Computer and System Sciences*, **22**, 365–383.

Savage, J. E. (1972). Computational work and time on finite machines. *Journal of the Association for Computing Machinery*, **19**, 660–674.

Stockmeyer, L., & Vishkin, U. (1984). Simulation of parallel random access machines by circuits. *SIAM Journal on Computing*, **13**, 409–422.

Valiant, L. G., & Brebner, G. J. (1982). A scheme for fast parallel communication. *SIAM Journal on Computing*, **11**, 350–361.