The Computational Power of Discrete Hopfield Nets with Hidden Units*

Pekka Orponen
Department of Computer Science
University of Helsinki, Finland†

Abstract

We prove that polynomial size discrete Hopfield networks with hidden units compute exactly the class of Boolean functions PSPACE/poly, i.e., the same functions as are computed by polynomial space-bounded nonuniform Turing machines. As a corollary to the construction, we observe also that networks with polynomially bounded interconnection weights compute exactly the class of functions P/poly, i.e., the class computed by polynomial time-bounded nonuniform Turing machines.

1 Introduction

We investigate the power of discrete Hopfield networks (Hopfield, 1982) as general computational devices. Our main interest is in the problem of Boolean function computation by symmetric networks of weighted threshold logic units; but for the constructions, we also need to consider asymmetric nets.

In our model of network computation, the input to a net is initially loaded onto a set of designated input units; then the states of the units in the network are updated repeatedly, according to their local update rules until the network (possibly) converges to some stable global state, at which

*A preliminary version of this work appears in *Proceedings of the 20th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science Vol. 700*, Springer-Verlag, Berlin Heidelberg, 1993, pp. 215–226.

†Address: P. O. Box 26, FIN–00014 University of Helsinki, Finland. E-mail: orponen@cs.helsinki.fi. Part of this work was done while the author was visiting the Institute for Theoretical Computer Science, Technical University of Graz, Austria.

point the output is read from a set of designated output units. We only consider finite networks of units with binary states, and with discrete-time synchronous dynamics (i.e., all the units are updated simultaneously in parallel). However, it is known that any computation on a synchronous network can be simulated on an asynchronous network where the updates are performed in a specific sequential order (Tchuente 1986; Bruck and Goodman 1988), or indeed even on a network where the update order is a priori totally undetermined (Orponen 1995).

Following the early work of McCulloch and Pitts (1943) and Kleene (1956) (see also Minsky 1972), it has been customary to think of finite (asymmetric) networks of threshold logic units as equivalent to finite automata (for recent work along these lines see, e.g., Alon *et al.* 1991; Horne and Hush 1994; Indyk 1995). However, in Kleene's construction for the equivalence, the input to a net is given as a sequence of pulses, whereas from many of the current applications' point of view it would be more natural to think of all of the input as being loaded onto the network in the beginning of a computation. (This is also the input convention followed in standard Boolean circuit complexity theory (Wegener 1987).) Of course, this view makes the network model *nonuniform*, in the sense that any single net operates on only fixed-length inputs, and to compute a function on an infinite domain one needs a *sequence* of networks.

Since a recurrent net of s units converging in time t may be "unwound" into a feedforward circuit of size $s \cdot t$, the class of Boolean functions computed by polynomial size, polynomial time asymmetric nets coincides with the class P/poly of functions computed by polynomial size Boolean circuits or, equivalently, polynomial time Turing machines with a polynomially bounded number of nonuniform "advice bits" (Karp and Lipton 1982; Balcázar *et al.* 1988)[1]. On the other hand, if computation times are not bounded, then a relatively straightforward argument shows that the class of functions computed by polynomial size asymmetric nets equals the class PSPACE/poly of functions computed by polynomial space bounded Turing machines with polynomially bounded advice. Parberry (1990) attributes this result to an early unpublished report by Lepley and Miller (1983), but for completeness we outline a proof in Section 3.

Thus, general asymmetric recurrent nets are fairly easy to characterize computationally, and turn out to be quite powerful. On the other hand, as pointed out by Hopfield (1982), networks with symmetric interconnections possess natural Liapunov functions, and are thus at least dynamically

[1] In a recent paper, Siegelmann and Sontag (1994) prove that also *bounded* size asymmetric recurrent nets with *real-valued* unit states and connection weights, and a saturated-linear transfer function compute in polynomial time exactly the functions in P/poly.

much more constrained. (The simple dynamics, and the generic form of the Liapunov functions are also what makes symmetric networks so attractive in many applications.) Specifically, Goles (1982) and Hopfield (1982) observed that under asynchronous updates, any symmetric net with no negative self-connections at the units always converges from any initial state to some stable final state. By analyzing the rate of decrease of the Liapunov functions, Fogelman *et al.* (1983) further showed that in a symmetric net of p units with no negative self-connections, and with integer connection weights w_{ij}, the convergence requires at most a total of

$$3 \sum_{i,j} |w_{ij}| = O(p^2 \cdot \max_{i,j} |w_{ij}|)$$

unit state changes. Under synchronous updates a similar bound holds also for nets with negative self-connections (Poljak and Sůra 1983; Goles *et al.* 1985; Bruck and Goodman 1988), but in this model the network may also converge to oscillate between two alternating states instead of a unique stable state.

Thus, in particular, symmetric networks with polynomially bounded weights converge in polynomial time. On the other hand, networks with exponentially large weights may indeed require an exponential time to converge, as was first shown by Goles and Olivos (1981) for synchronous updates, and by Haken and Luby (1988) for a particular asynchronous update rule. (The former construction was later simplified by Goles and Martínez (1989).) A network requiring exponential time to converge under an arbitrary asynchronous update order was first demonstrated by Haken (1989). The existence of networks with exponentially long asynchronous transients is now known to follow also from the general theory of local search for optimization problems (Schäffer and Yannakakis 1991).

In this paper, we prove that despite their constrained dynamics, *computationally* symmetric networks lose nothing of their power: specifically, symmetric polynomial size networks with unbounded weights can compute all functions in PSPACE/poly, and networks with polynomially bounded weights can compute all functions in P/poly. The idea, presented in section 4, is to start with the simulation of space-bounded Turing machines by asymmetric nets, and then replace each of the asymmetric edges by a sequence of symmetric edges whose behavior is sequenced by clock pulses. The appropriate clock can be obtained from, e.g., the symmetric exponential-transient network by Goles and Martínez (1989). Obviously, such a clock network cannot run forever (in this case it is not sufficient to have a network that simply oscillates between two states), but nevertheless the sequence of pulses it generates is long enough to simulate a polynomially space-bounded

computation or, in the case of polynomially bounded weights, a polynomially time-bounded one.

For more information on the computational aspects of recurrent threshold logic networks, or more generally automata networks, see, e.g., the survey articles and books by Floréen and Orponen (1994), Fogelman *et al.* (1987), Goles and Martínez (1990), Kamp and Hasler (1990), and Parberry (1990, 1994).

2 Preliminaries

Following Parberry (1990), we define a *(discrete) neural network* as a 6-tuple $N = (V, I, O, A, w, h)$, where V is a finite set of *units*, which we assume are indexed as $V = \{1, \ldots, p\}$; $I \subseteq V$ and $O \subseteq V$ are the sets of *input* and *output units*, respectively; $A \subseteq V$ is a set of *initially active units*, of which we require that $A \cap I = \emptyset$; $w : V \times V \to Z$ is the *edge weight matrix*, and $h : V \to Z$ is the *threshold vector*. The *size* of a network is its number of units, $|V| = p$, and the *weight* of a network is defined as its sum total of edge weights, $\sum_{i,j \in V} |w_{ij}|$. A *Hopfield net (with hidden units)* is a neural network N whose weight matrix is symmetric.

Given a neural network N, let us denote $|I| = n$, $|O| = m$; moreover, let us assume that the units are indexed so that the input units appear at indices 1 to n. The network then computes a partial mapping

$$f_N : \{0,1\}^n \to \{0,1\}^m$$

as follows. Given an input x, $|x| = n$, the states s_i of the input units are initialized as $s_i = x_i$. The states of the units in set A are initialized to 1, and the states of the remaining units are initialized to 0. Then new states s'_i, $i = 1, \ldots, p$, are computed simultaneously for all the units according to the rule

$$s'_i = \mathrm{sgn}(\sum_{j=1}^{p} w_{ij} s_j - h_i),$$

where $\mathrm{sgn}(t) = 1$ for $t \geq 0$, and $\mathrm{sgn}(t) = 0$ for $t < 0$. This updating process is repeated until no more changes occur, at which point we say that the network has *converged*, and the output value $f_N(x)$ can be read off the output units (in order). If the network does not converge on input x, the value $f_N(x)$ is undefined.

For simplicity, we consider from now on only networks with a single output unit; the extensions to networks with multiple outputs are straightforward. The *language recognized* by a single-output

network N, with n input units, is defined as

$$L(N) = \{x \in \{0,1\}^n \mid f_N(x) = 1\}.$$

Given a language $A \subseteq \{0,1\}^*$, denote $A^{(n)} = A \cap \{0,1\}^n$. We consider the following complexity classes of languages:

$$\text{PNETS} = \{A \subseteq \{0,1\}^* \mid \text{ for some polynomial } q, \text{ there is for each } n \text{ a network of size at most } q(n) \text{ that recognizes } A^{(n)} \},$$

$$\text{PNETS(symm)} = \{A \subseteq \{0,1\}^* \mid \text{ for some polynomial } q, \text{ there is for each } n \text{ a Hopfield net of size at most } q(n) \text{ that recognizes } A^{(n)} \},$$

$$\text{PNETS(symm, small)} = \{A \subseteq \{0,1\}^* \mid \text{ for some polynomial } q, \text{ there is for each } n \text{ a Hopfield net of weight at most } q(n) \text{ that recognizes } A^{(n)} \}.$$

Let $\langle x, y \rangle$ be some standard pairing function mapping pairs of binary strings to binary strings (see, e.g., Balcázar *et al.* (1988, p. 7)). A language $A \subseteq \{0,1\}^*$ belongs to the nonuniform complexity class PSPACE/poly (Karp and Lipton 1982; Balcázar *et al.* 1987; Balcázar *et al.* 1988, p. 100), if there exist a polynomial space bounded Turing machine M and an "advice" function $f : N \to \{0,1\}^*$, such that for some polynomial q and all $n \in N$, $|f(n)| \leq q(n)$, and for all $x \in \{0,1\}^*$,

$$x \in A \quad \Leftrightarrow \quad M \text{ accepts } \langle x, f(|x|) \rangle.$$

The class P/poly is defined analogously, using polynomial time instead of space bounded Turing machines. The class P/poly can also be characterized as the class of languages recognized by polynomial size-bounded sequences of feedforward Boolean circuits (Balcázar *et al.* 1988, p. 111). This includes circuits using threshold logic gates, as any threshold function on k variables can be implemented as an AND/OR/NOT-circuit of size $O(k^2 \log^2 k)$ and depth $O(\log k)$ (Parberry 1994, p. 173).

3 Simulating Turing Machines with Asymmetric Nets

Simulating space-bounded Turing machines with asymmetric neural nets is fairly straightforward.

Theorem 3.1 PNETS = PSPACE/poly.

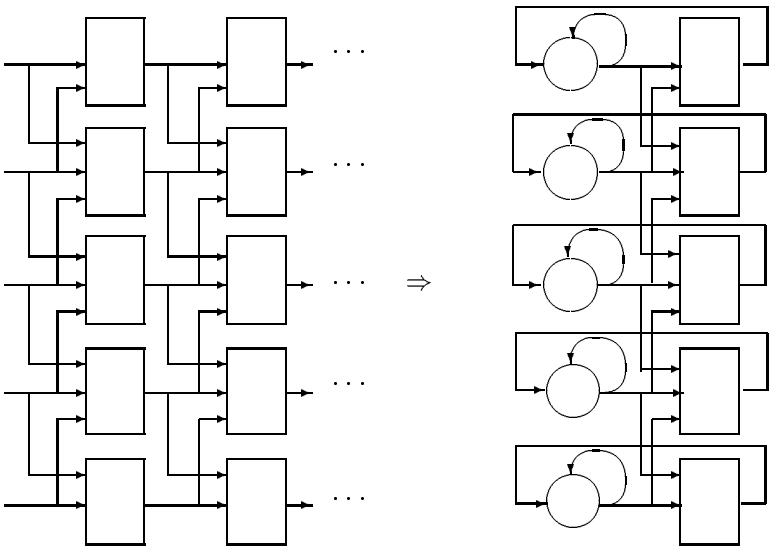

Figure 1: Simulation of a space bounded Turing machine by an asymmetric recurrent net.

Proof (outline). To prove the inclusion PNETS $\subseteq$ PSPACE/poly, observe that given (a description of) a neural net, it is possible to simulate its behavior *in situ*. Hence, there exists a universal neural net interpreter machine M that given a pair $\langle x, N \rangle$ simulates the behavior of net N on input x in linear space. Let then language A be recognized by a polynomial size bounded sequence of nets (N_n). Then $A \in$ PSPACE/poly via the machine M and advice function $f(n) = N_n$.

To prove the converse inclusion, let $A \in$ PSPACE/poly via a machine M and advice function f. Let the space complexity of M on input $\langle x, f(|x|) \rangle$ be bounded by a polynomial $q(|x|)$. Without loss of generality (see, e.g., Balcázar *et al.* 1988) we may assume that M has only one tape, halts on any input $\langle x, f(|x|) \rangle$ in time $c^{q(|x|)}$, for some constant c, and indicates its acceptance or rejection of the input by printing a 1 or a 0 on the first square of its tape.

Following the standard simulation of Turing machines by combinational circuits (Balcázar *et al.* 1988, pp. 106–112), it is straightforward to construct for each n a feedforward threshold logic circuit that simulates the behavior of M on inputs of length n. (More precisely, the circuit simulates computations $M(\langle x, f(n) \rangle)$, where $|x| = n$.) This circuit consists of $c^{q(n)}$ "layers" of $O(q(n))$ parallel wires, where the tth layer represents the configuration of the machine M at time t (Figure 1, left). Every two consecutive layers of wires are interconnected by an intermediate layer of $q(n)$ constant-size subcircuits, each implementing the local transition rule of machine M at a single position of the simulated configuration. The input x is entered to the circuit along input wires; the advice

string $f(n)$ appears as a constant input on another set of wires; and the output is read from the particular wire at the end of the circuit that corresponds to the first square of the machine tape.

One may now observe that the interconnection patterns between layers are very uniform: all the local transition subcircuits are similar, with a structure that depends only on the structure of M, and their number depends only on the length of x. Hence we may replace the exponentially many consecutive layers in the feedforward circuit by a single transformation layer that feeds back on itself (Figure 1, right). (As can be seen in the figure, we now use an explicit layer of units to represent the configuration of machine M, with small positive self-connections to maintain the representation between successive transformations.) The size of the recurrent network thus obtained is only $O(q(n))$. When initialized with input x loaded onto the appropriate input units, and advice string $f(n)$ mapped to the appropriate initially active units, the network will converge in $O(c^{q(n)})$ update steps, at which point the output can be read off the unit corresponding to the first square of the machine tape.

□

4 Simulating Asymmetric Nets with Symmetric Nets

Having now shown how to simulate polynomial space-bounded Turing machines by polynomial size asymmetric nets, the remaining problem is how to simulate the asymmetric edges in a network by symmetric ones. This is not possible in general, as witnessed by the different convergence behaviors of asymmetric and symmetric nets. However, in the special case of *convergent* computations on asymmetric nets the simulation can be effected.

Theorem 4.1 PNETS(symm) = PSPACE/poly.

Proof. Because PNETS(symm) $\subseteq$ PNETS, and by the previous theorem PNETS $\subseteq$ PSPACE/poly, it suffices to prove the inclusion PSPACE/poly $\subseteq$ PNETS(symm).

Given any $A \in$ PSPACE/poly, there is by Theorem 3.1 a sequence of polynomial size asymmetric networks recognizing A. Rather than show how this specific sequence of networks can be simulated by symmetric networks, we shall show how to simulate the convergent computations of an *arbitrary* asymmetric network of n units and e edges of nonzero weight on a symmetric network of $O(n+e)$ units and $O(n^2)$ edges.

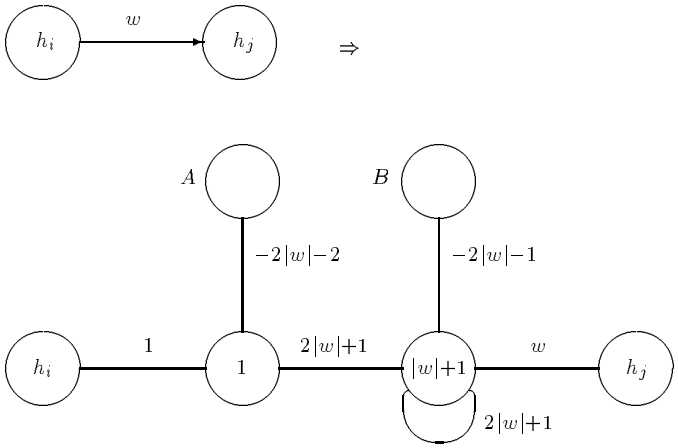

Figure 2: A sequence of symmetric edges simulating an asymmetric edge of weight w.

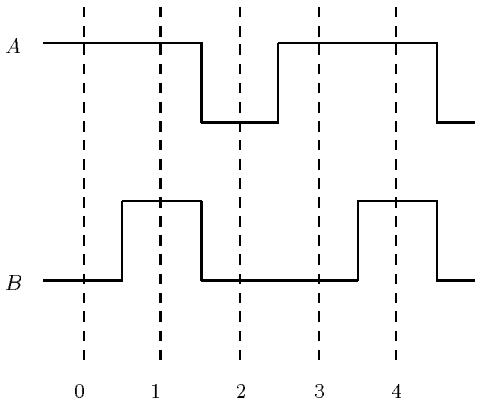

Figure 3: The clock pulse sequence used in the edge simulation of Figure 2.

The construction combines two network "gadgets": a simplified version of a mechanism due to Hartley and Szu (1987) for simulating an asymmetric edge by a sequence of symmetric edges and their interconnecting units, whose behavior is coordinated by a system clock (Figures 2, 3); and a binary counter network due to Goles and Martínez (1989; see also Goles and Martínez 1990, pp. 88–95) that can count up to 2^n using about $3n$ units and $O(n^2)$ symmetric edges (Figure 4). An important observation here is that any convergent computation by a network of n units has to terminate in 2^n synchronous update steps, because otherwise the network repeats a configuration and goes into a cycle; hence, the exponential time counter network can be used to provide a sufficient number of clock pulses for the simulation to be performed.

Let us first consider the gadget for a symmetric simulation of an asymmetric edge of weight w from a unit i to a unit j (Figure 2). Here the idea is that the two units inserted between the units i and j in the symmetric network function as locks in a canal, permitting information to move only from left to right. The locks are sequenced by clock pulses emanating from the units labeled A and B, in cycles of period three as presented in Figure 3.

At time 0 clock A is on, setting the first intermediate unit to zero, and clock B is off, permitting the state of the second intermediate unit to influence the state computation at unit j. At time 1 clock B turns on, clearing the second intermediate unit at time 2 (note that the connection from unit j is not strong enough to turn this unit back on). This will make the state of unit j indeterminate at time 3. At time 2, clock A turns off, permitting a new state to be copied from unit i to the first intermediate unit at time 3 (i.e., just before the state of unit i becomes indeterminate). At time 3, clock A turns on again, clearing the first intermediate unit at time 4; but simultaneously at time 4 the new state is copied from the first to the second intermediate unit, from where it can then influence the computation of the new state of unit j at time 5.

The next question is how to generate the clock pulses A and B. It is not possible to construct a symmetric clock network that runs forever: at best such a network can end up oscillating between two states, but this is not sufficient to generate the period 3 pulse sequences required for the previous construction. However, Figure 4 presents the first two stages in the construction of a $(3n-4)$-unit symmetric network with a convergence time of more than 2^n (actually, $2^n + 2^{n-1} - 3$) synchronous update steps. (For the full details of the construction, see (Goles and Martínez 1989).) The idea here is that the n units in the upper row implement a binary counter, counting from all 0's to all 1's (in the figure, the unit corresponding to the least significant bit is to the right). For each "counter" unit added to the upper row, after the two initial ones, two "control" units are

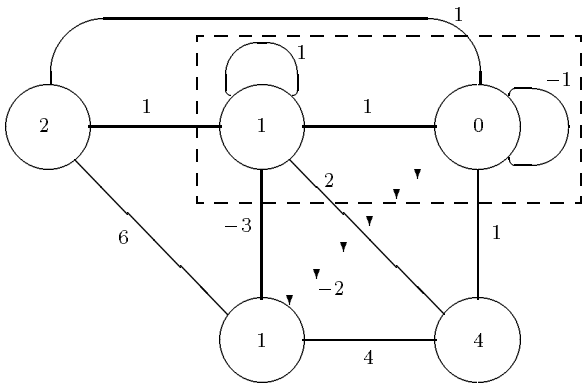

Figure 4: The first two stages in the construction of a binary counter network (Goles and Martínez 1989).

added to the lower row. The purpose of the latter is to first turn off all the "old" units, when the new counter unit is activated, and from then on balance the input coming to the old units from the new units, so that the old units may resume counting from zero[2].

It is possible to derive from such a counter network a sufficient number of the A and B pulse sequences by means of the delay line network presented in Figure 5. Here the unit at the upper left corner is some sufficiently slow oscillator; since we require pulse sequences of period three, this could be the second counter unit in the preceding construction, which is "on" for four update steps at a time. (Thus, a 2^{n+1}-counter suffices to sequence computations of length up to $2^n - 1$.) The delay line operates as follows: when the oscillator unit turns on, it "primes" the first unit in the line; but nothing else happens until the oscillator turns off. At that point the "on" state begins to travel down the line, one unit per update step, and the pulses A and B are derived from the appropriate points in the line.

The value W used in the construction has to be chosen so large that the states of the units in the underlying network have no effect back on the delay line. It is sufficient to choose W larger than the total weight of the underlying network. Similarly, the weights and thresholds in the counter network have to modified so that the connections to the delay line do not interfere with the counting. Assuming that $W \geq 3$, it is here sufficient to multiply all the weights and thresholds

[2]Following the construction by Goles and Martínez (1989), we have made use of one negative self-connection in the counter network. If desired, this can be removed by making two copies of the least significant unit, both with threshold 0, interconnected by an edge of weight -1, and with the same connections to the rest of the network as the current single unit. All the other weights and thresholds in the network must then be doubled.

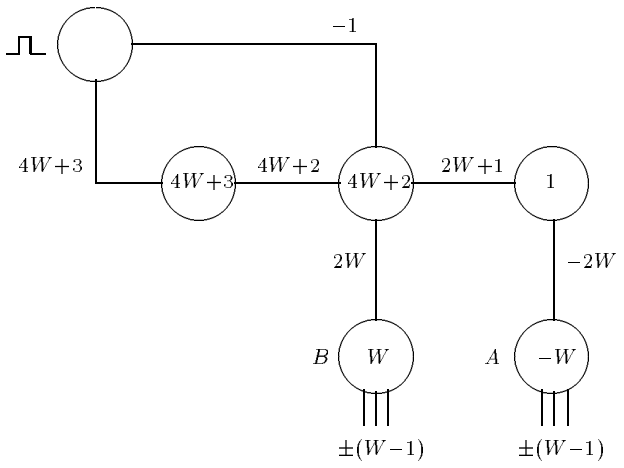

Figure 5: A delay line for generating clock pulses from the binary counter network in Figure 4.

by $6W$, and then subtract one from each threshold.

□

Concerning the edge weights in the above constructions, one can see that in the network implementing the machine simulation (Figures 1, 2), the weights actually are bounded by some constant that depends only on the simulated machine M; in the delay line, the weights are proportional to the total weight of the underlying network; and the weights in the counter network (Figure 4) are proportional to the length of the required simulation and, less significantly, to the weight of the delay line. Thus, we obtain as a corollary to the construction that if the simulated Turing machine (or, more generally, asymmetric network) is known to converge in polynomial time, then it is sufficient to have polynomially bounded weights in the simulating symmetric network. Formulating this in terms of nonuniform complexity classes, we obtain:

Corollary 4.2 PNETS(symm, small) = P/poly.

Consequently, anything that can be computed by asymmetric networks, or nonuniform Turing machines, in polynomially bounded time can also be computed by polynomial weight symmetric networks, with their guaranteed good convergence properties.

The result also implies that large (i.e., superpolynomial) weights are essential for the computational power of polynomial size symmetric networks if and only if P/poly $\neq$ PSPACE/poly. (The condition P/poly = PSPACE/poly is known to have the unlikely consequence of collapsing the "polynomial time hierarchy" to its second level (Karp and Lipton 1982).) In asymmetric networks

large weights are not needed; in fact even bounded weights suffice, as can be seen by conceptually replacing each threshold logic unit by a corresponding AND/OR/NOT subcircuit.

5 Conclusion and Open Problems

We have characterized the classes of Boolean functions computed by asymmetric and, more interestingly, symmetric polynomial size recurrent networks of threshold logic units under a synchronous update rule. When no restrictions are placed on either computation time or the sizes of interconnection weights, both of these classes of networks compute exactly the class of functions PSPACE/poly. If interconnection weights are limited to be polynomial in the size of the network, the class of functions computed by symmetric networks reduces to P/poly. This limitation has no effect on the computational power of asymmetric nets. Although we have considered here only networks with discrete synchronous dynamics, it can be shown that any computation on such a network can also be performed on a slightly larger network with a totally asynchronous dynamics (Orponen 1995).

Some of the open problems suggested by this work are the following. In the original associative memory model proposed by Hopfield (1982), all the units are used for both input and output, and no hidden units are allowed. Although this is a somewhat artificial restriction from the function computation point of view, characterizing the class of mappings computed by such networks would nevertheless be of some interest in the associative memory context.

Of more general interest would be the study of the *continuous-time* version of Hopfield's network model (Hopfield 1984; Hopfield and Tank 1985). It will be an exciting broad research task to define the appropriate notions of computability and complexity in this model, and attempt to characterize its computational power.

Acknowledgment

I wish to thank Mr. Juha Kärkkäinen for improving on my initial attempts at simplifying the Hartley/Szu network, and suggesting the elegant construction presented in Figure 2.

References

Alon, N., Dewdney, A. K., and Ott, T. J. 1991. Efficient simulation of finite automata by neural nets. *Journal of the ACM 38*, 495–514.

Anderson, J. A., and Rosenfeld, E. (eds.) 1988. *Neurocomputing: Foundations of Research.* The MIT Press, Cambridge, MA.

Balcázar, J. L., Díaz, J., and Gabarró, J. 1987. On characterizations of the class PSPACE/poly. *Theoret. Comput. Sci. 52*, 251–267.

Balcázar, J. L., Díaz, J., and Gabarró, J. 1988. *Structural Complexity I.* Springer-Verlag, Berlin Heidelberg.

Bruck, J., and Goodman, J. W. 1988. A generalized convergence theorem for neural networks. *IEEE Trans. Information Theory 34*, 1089–1092.

Floréen, P., and Orponen, P. 1994. Complexity Issues in Discrete Hopfield Networks. Report A–1994–4, University of Helsinki, Dept. of Computer Science. To appear in (Parberry t.a.).

Fogelman, F., Goles, E., and Weisbuch, G. 1983. Transient length in sequential iterations of threshold functions. *Discr. Appl. Math. 6*, 95–98.

Fogelman Soulié, F., Robert, Y., and Tchuente, M. 1987. *Automata Networks in Computer Science: Theory and Applications.* Manchester University Press.

Goles, E. 1982. Fixed point behavior of threshold functions on a finite set. *SIAM J. Alg. Discr. Methods 3*, 529–531.

Goles, E., Fogelman, F., and Pellegrin, D. 1985. Decreasing energy functions as a tool for studying threshold networks. *Discr. Appl. Math. 12*, 261–277.

Goles, E., and Martínez, S. 1989. Exponential transient classes of symmetric neural networks for synchronous and sequential updating. *Complex Systems 3*, 589–597.

Goles, E., and Martínez, S. 1990. *Neural and Automata Networks.* Kluwer Academic, Dordrecht.

Goles, E., and Olivos, J. 1981. The convergence of symmetric threshold automata. *Info. and Control 51*, 98–104.

Haken, A. 1989. Connectionist networks that need exponential time to converge. Unpublished manuscript, 10 pp. University of Toronto, Dept. of Computer Science, January 1989.

Haken, A., and Luby, M. 1988. Steepest descent can take exponential time for symmetric connection networks. *Complex Systems 2*, 191–196.

Hartley, R., and Szu, H. 1987. A comparison of the computational power of neural networks. In: *Proc. of the 1987 Internat. Conf. on Neural Networks, Vol. 3.* IEEE, New York. Pp. 15–22.

Hopfield, J. J. 1982. Neural networks and physical systems with emergent collective computational abilities. *Proc. Nat. Acad. Sci. USA 79*, 2554–2558. Reprinted in (Anderson and Rosenfeld 1988, pp. 460–464).

Hopfield, J. J. 1984. Neurons with graded response have collective computational properties like those of two-state neurons. *Proc. Nat. Acad. Sci. USA 81*, 3088–3092. Reprinted in (Anderson and Rosenfeld 1988, pp. 579–583).

Hopfield, J. J., and Tank, D. W. 1985. "Neural" computation of decisions in optimization problems. *Biol. Cybern. 52*, 141–152.

Horne, B. G., and Hush, D. R. 1994. Bounds on the complexity of recurrent neural network implementations of finite state machines. In: *Advances in Neural Information Processing Systems 6* (ed. J. D. Cowan, G. Tesauro, J. Alspector). Morgan Kaufmann, San Francisco, Ca. Pp. 359–366.

Indyk, P. 1995. Optimal simulation of automata by neural nets. In: *Proc. 12th Annual Symposium on Theoretical Aspects of Computer Science (Munich, Germany, March 1995).* Lecture Notes in Computer Science 900. Springer-Verlag, Berlin. Pp. 337–347.

Kamp, Y., and Hasler, M. 1990. *Recursive Neural Networks for Associative Memory.* John Wiley & Sons, Chichester.

Karp, R. M., and Lipton, R. J. 1982. Turing machines that take advice. *L'Enseignement Mathématique 28*, 191–209.

Kleene, S. C. 1956. Representation of events in nerve nets and finite automata. In: *Automata Studies* (ed. C. E. Shannon, J. McCarthy). Annals of Mathematics Studies n:o 34. Princeton Univ. Press, Princeton, NJ. Pp. 3–41.

Lepley, M., and Miller, G. 1983. Computational power for networks of threshold devices in an asynchronous environment. Unpublished manuscript, 6 pp., Massachusetts Inst. of Technology, Dept. of Mathematics, 1983.

McCulloch, W. S., and Pitts, W. 1943. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys. 5*, 115–133. Reprinted in (Anderson and Rosefeld 1988, pp. 18–27).

Minsky, M. L. 1972. *Computation: Finite and Infinite Machines.* Prentice-Hall, Englewood Cliffs, NJ.

Orponen, P. 1995. Computing with truly asynchronous threshold logic networks. Manuscript submitted for publication, 18 pp. Technical University of Graz, Institute for Theoretical Computer Science, March 1995.

Parberry, I. 1990. A primer on the complexity theory of neural networks. In: *Formal Techniques in Artificial Intelligence: A Sourcebook* (ed. R. B. Banerji). Elsevier – North-Holland, Amsterdam. Pp. 217–268.

Parberry, I. 1994. *Circuit Complexity and Neural Networks.* The MIT Press, Cambridge, Ma.

Parberry, I. (ed.) t.a. *The Computational and Learning Complexity of Neural Networks: Advanced Topics.* The MIT Press, Cambridge, Ma., to appear.

Poljak, S., and Sůra, M. 1983. On periodical behaviour in societies with symmetric influences. *Combinatorica 3*, 119–121.

Schäffer, A. A., and Yannakakis, M. 1991. Simple local search problems that are hard to solve. *SIAM J. Computing 20*, 56–87.

Siegelmann, H. T., and Sontag, E. D. 1994. Analog computation via neural networks. *Theoretical Computer Science 131*, 331–360.

Tchuente, M. 1986. Sequential simulation of parallel iterations and applications. *Theoretical Computer Science 48*, 135–144.

Wegener, I. 1987. *The Complexity of Boolean Functions.* John Wiley & Sons, Chichester, and B. G. Teubner, Stuttgart.